Relazione progetto machine learning

Martino Pettinari 866496, Davide Creati, 869274

Abstract

Nella relazione abbiamo cercato di riassumere tutti gli step che ci hanno permesso di arrivare all'obiettivo del progetto ovvero addestrare 2 modelli scelti e valutarne la correttezza e precisione. Nel processo si includono scelte stilistiche, assunzioni o ipotesi fatte durante lo svolgimento.

1 Introduzione e obiettivi

Nel corso degli ultimi anni, il tennis ha acquisito una crescente popolarità in Italia, grazie al talento e alle imprese di sportivi italiani che hanno portato questo paese sui campi internazionali. Questo fenomeno ha suscitato un crescente interesse e coinvolgimento nel mondo dello sport, con sempre più persone che seguono i nostri atleti nei tornei più prestigiosi del circuito ATP. In questo contesto, il presente lavoro progettuale si propone di esplorare il potenziale di alcuni modelli di machine learning nel contesto della previsione dei risultati delle partite di tennis.

Per questo studio, abbiamo selezionato esclusivamente dati provenienti dai quattro tornei del Grande Slam del circuito ATP. Il circuito professionistico ATP è composto da molteplici tornei che si svolgono durante l'arco dell'anno. Ognuno di questi tornei da punteggi differenti, in base all'importanza del torneo. Questa scelta è stata motivata dalla rilevanza di questi eventi. Utilizzare dati provenienti esclusivamente dai Grand Slam, ci ha permesso di concentrarci su eventi di prestigio e di rilevanza internazionale, durante i quali, i giocatori, offrono prestazioni al massimo delle loro capacità dato che, la posta in gioco (rappresentata dai punti assegnati), crea un incentivo. Per la realizzazione del progetto abbiamo utilizzato solo i dati dei tornei Grand Slam (ovvero quelli con il maggior punteggio) avvenuti nel 2023.

L'obiettivo centrale di questo progetto è stato sviluppare 3 modelli di machine learning in grado di prevedere l'esito di ogni singola partita disputata durante il primo Grand Slam del 2024 (Australian Open), utilizzando i dati dei 4 Grand Slam del 2023 come addestramento per i modelli.

2 Dati e pulizia dataset

I dati utilizzati per lo sviluppo del progetto sono stati scaricati da `http://tennis-data.co.uk/2023/2023.xlsx` e da `blue`http://tennis-data.co.uk/2024/2024.xlsx. Il dataset che abbiamo utilizzato contiene tutti i match del circuito ATP disputati nel 2023 e 2024. L'obiettivo principale di questa fase è stato analizzare i dati.

Di seguito sono riportati tutti i valori presenti all'interno del dataset, prima della pulizia e della riduzione dello spazio di input.

ATP	Location	Tournament	Date	Series	Court
Surface	Round	Best of	Winner	Loser	WRank
LRank	WPts	LPts	W1	L1	W2
L2	W3	L3	W4	L4	W5
L5	Wsets	Lsets	Comment	B365W	B365L
PSW	PSL	MaxW	MaxL	AvgW	AvgL

Dall'analisi esplorativa dei dati è emerso che il dataset contiene molte informazioni, riassumento gli attributi riportati sopra, in tabella, si hanno:

- Giocatori che disputano il match e posizionamento di ogni giocatore nel rank del circuito ATP;
- Dettagli in merito alla partita (tipologia di superficie, luogo del campo, stadio, luogo di disputa del il match, etc.);
- Risultato della partita con punteggi per ogni set giocato;
- Quote dei principali siti di scommesse (come B365, Pinnacle, ...).

L'elenco rappresenta in modo riassuntivo le diverse tipologie di dati che compongono i dataset. In particolare, le informazioni dei siti di scommessa, sono dei dati che racchiudono molte statistiche delle partite.

Una volta analizzati e compresi i dati, ci siamo dedicati alla pulizia dei dataset. Per renderlo utilizzabile per il training dei modelli è stato necessario eseguire più step di pulizia sui dati, questo processo si è composto nel seguente modo:

1. Esclusione delle righe che non fossero all'interno del perimetro. Poichè la scelta è stata quella di utilizzare solo i dati dei 4 Grand Slam annuali, abbiamo eliminato tutte le righe del dataset che non fossero relative ai match di questi 4 tornei;
2. Rimozione delle informazioni poco rilevanti, come ad esempio il luogo di disputa della partita o la data. Oltre a ciò abbiamo rimosso le colonne in riferimento a dati che non sono disponibili finche il match non si è concluso e quindi non utilizzabili per l'apprendimento del modello. Un esempio sono i risultati di ogni set disputato di orni partita;
3. Eliminazione righe con valori null, essendo pochi valori la scelta è ricaduta nell'eliminare quelle righe per evitare di inserire dei valori poco accurati. Nel caso in cui la colonna "rank" contenesse il valore nullo, veniva messo il valore del rank massimo trovato + 1. Questo perché si è supposto che se un giocatore non avesse assegnato un rank, allora fosse "scarso" e non dovesse rientrare nella lista.

4. Mixing dei dati, questo è stata una scelta effettuata per evitare che il modello sviluppasse un pattern fisso di apprendimento. Il dataset di partenza posiziona il vincitore come primo giocatore, per evitare ciò è bastato dividere in due il dataset, nel secondo caso invertire le colonne e mischiare la seconda metà con la prima;

5. Trasformazione di tutte le stringhe del dataset in codifiche numeriche. Per realizzare ciò è stata necessaria la trasformazione da stringa a categoria, per le colonne interessate per poi trasformare le categorie in intero tramite un package dedicato. Questo è un passaggio fondamentale per permettere ai modelli un apprendimento corretto ma anche, per poter standardizzare il dataset prima delle fasi di apprendimento.

Concludendo, attraverso una serie di passaggi di pulizia e preparazione dei dati, siamo stati in grado di rendere il dataset pronto per l'addestramento dei modelli di machine learning. Questo processo ci ha permesso di ottenere un dataset coerente e bilanciato, pronto per essere utilizzato nell'addestramento dei modelli per l'analisi dei match di tennis nei tornei del Grand Slam.

3 Analisi dei dati

Per analizzare i dati, prima di iniziare la fase di addestramento dei modelli, abbiamo iniziato osservando i dati grezzi e successivamente utilizzando una matrice di correlazione e la PCA come strumenti per comprendere meglio la struttura e le relazioni tra le variabili.

3.1 Matrice di correlazione

Questa matrice mostra il grado di correlazione lineare tra tutte le coppie di features presenti nel dataset. Il possibile range di valori di correlazione può variare da da -1 a 1, dove i valori prossimi a 1 indicano una forte correlazione positiva mentre, i valori prossimi a -1 indicano una forte correlazione negativa, infine, i valori prossimi a 0 indicano una debole correlazione.

Analizzando la matrice di correlazione (escludendo la diagonale principale), osservabile in 1, si può notare come alcune variabili presentano una correlazione lineare forte, come ad esempio "B1" e "PS1" che presentano una correlazione negativa di 0.97. Questi dati indicano i valori dei siti di scommesse per la vincita del giocatore 1, un'alta correlazione denota una coerenza tra i dati di previsione ottenuti dalle statistiche di questi siti. Un esempio di correlazione moderata è tra "B1" e "Target" che, essendo di valore 0.22, denota come la previsione delle statistiche tenda ad avere una correlazione con il target effettivo che si vuole predire. Dall'altro lato, si notano correlazioni negative, tra le feature che indicano le statistiche per il giocatore uno, in correlazione con quelle del giocatore 2.

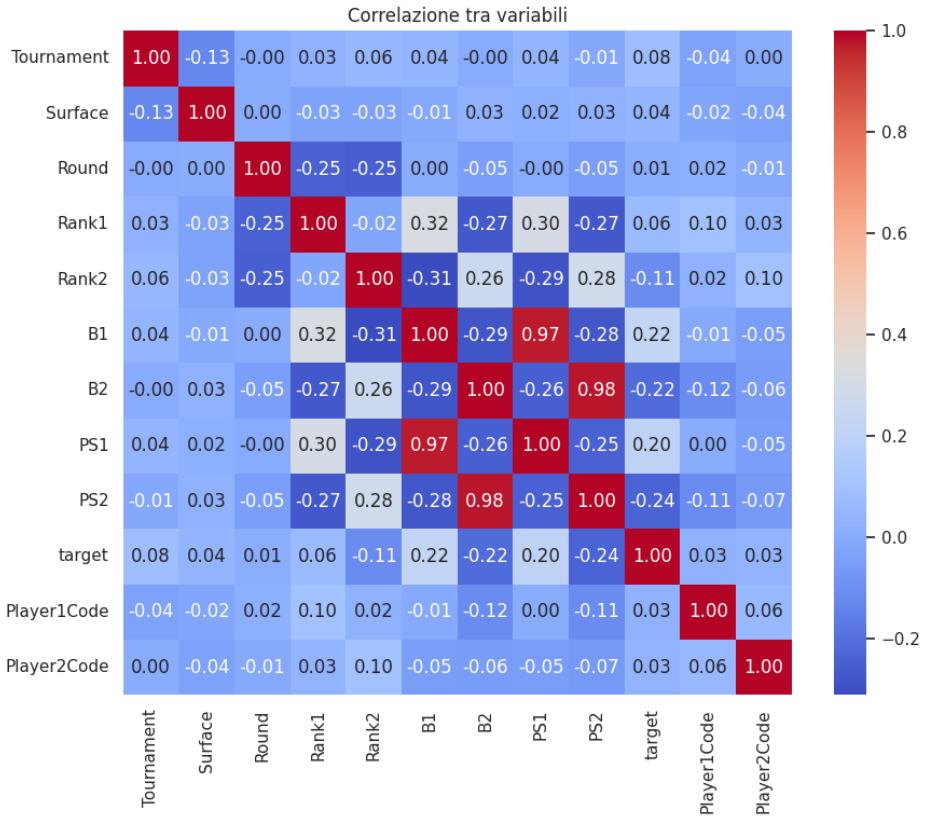

Figure 1: Correlazione dati tra features

3.2 Principal Component Analysis

Abbiamo impiegato la PCA per isolare le componenti fondamentali che spiegano la maggior parte della varianza del dataset. È importante notare che, durante l'applicazione della PCA, abbiamo scelto di utilizzare solo i dati numerici del dataset, escludendo i dati categorici. Questa decisione è stata motivata dal fatto che i dati categorici, anche se trasformati in forma numerica, possono introdurre rumore e distorsioni nei calcoli della PCA, in quanto non hanno un significato numerico intrinseco e possono portare a interpretazioni errate della struttura dei dati. In questo modo, abbiamo potuto concentrare l'analisi sulle caratteristiche numericamente rilevanti del dataset, garantendo una maggiore precisione e affidabilità nei risultati ottenuti. La PCA ci ha quindi consentito di visualizzare le informazioni contenute nelle variabili originali in un nuovo spazio con dimensioni ridotte, preservando al massimo la variazione dei dati e ottimizzando la nostra capacità di previsione dei risultati delle partite di tennis. Mediante questa analisi, siamo stati in grado di identificare e rimuovere le variabili meno influenti, contribuendo a ottimizzare ulteriormente il nostro modello predittivo.

Il grafico che si può osservare nell'immagine 2, plot ha una forma a "gomito", con un tratto iniziale ripido seguito da una curva più graduale. Questo è un buon segno, in quanto indica che le prime PC catturano una porzione significativa della variabilità totale del dataset. In base a questo grafico, si potrebbe decidere di estrarre le prime 3/4 PC. Questo perché catturano una porzione significativa della variabilità totale e sono relativamente facili da

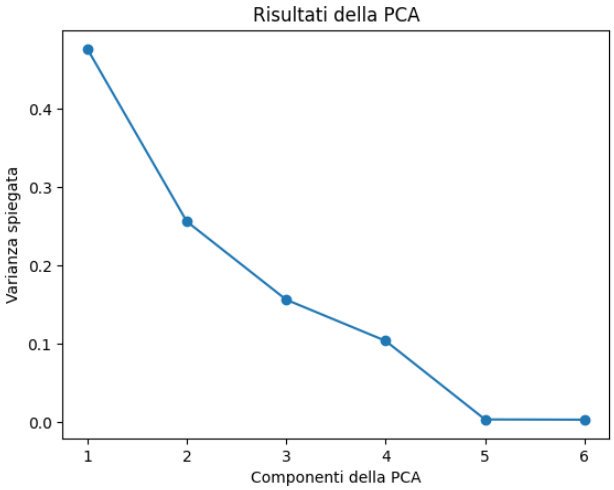

Figure 2: PCA eseguito sul dataset di training

interpretare.

Grazie a queste analisi preliminari è possibile cominciare con la fase di scelta dei modelli e di addestramento avendo migliore comprensione della struttura del dataset e delle relazioni tra le variabili.

4 Modelli utilizzati

Nella realizzazione del progetto, ci siamo concentrati su 3 modelli di machine learning. Il primo modello su cui abbiamo focalizzato il nostro lavoro sono le reti neurali, scelte per la loro capacità di gestire relazioni complesse nei dati. Per quanto riguarda il secondo modello, abbiamo esaminato diverse alternative e valutato le prestazioni di vari modelli. Dopo una serie di test, abbiamo identificato gli alberi decisionali come la nostra seconda scelta. Abbiamo optato per questi due modelli perché, essendo molto diversi tra loro, ci hanno consentito di garantire un risultato progettuale accurato, affidabile e in grado di soddisfare gli obiettivi del progetto. Da notare che abbiamo utilizzato anche Naive Bayes come terzo modello, completando così l'insieme dei tre modelli di machine learning utilizzati nel progetto.

Nella scelta di un modello per un progetto di machine learning finalizzato alla previsione dei vincitori delle partite di tennis, l'adozione di una rete neurale può offrire diversi vantaggi significativi. Le reti neurali sono particolarmente adatte a questo tipo di compito poiché sono in grado di catturare relazioni complesse e non lineari tra le variabili di input, come le caratteristiche dei giocatori, il loro ranking, il tipo di superficie del campo e altre variabili pertinenti. La capacità delle reti neurali di apprendere in modo automatico e adattativo dai dati disponibili consente loro di adattarsi a una vasta gamma di dati eterogenei e di individuare pattern nascosti che potrebbero non essere facilmente identificabili con altri approcci di modellazione. Inoltre, le reti neurali sono in grado di gestire grandi quantità di dati senza problemi, il che è cruciale considerando la vastità e la complessità dei dati

disponibili nel contesto dei match di tennis. Infine, la flessibilità delle reti neurali consente di esplorare diversi tipi di architetture e di ottimizzare il modello in base alle esigenze specifiche del progetto, garantendo una soluzione su misura per la previsione accurata dei vincitori delle partite di tennis.

Gli alberi di decisione sono una ottima scelta dato che sono facili da interpretare e visualizzare, ciò ci ha permesso di comprendere il comportamento generale del modello. Questa trasparenza è particolarmente importante quando si lavora con dati relativamente semplici come quelli delle partite di tennis, poiché consente di identificare facilmente i fattori più influenti nel determinare i vincitori delle partite. In definitiva, gli alberi di decisione offrono un equilibrio tra prestazioni e facilità d'uso che, ci ha motivati a sceglierli come secondo modello per la previsione dei vincitori delle partite di tennis.

Inoltre, gli alberi di decisione sono in grado di gestire sia dati categorici che numerici senza necessitare di trasformazioni complesse dei dati. Nel contesto del progetto, molte delle variabili impiegate sono categoriche, come il tipo di superficie. Gli alberi, essendo di natura categorica, si adattano bene a questo tipo di dati e sono in grado di apprendere relazioni non lineari tra le variabili. Oltre a ciò, riescono a gestire relazioni complesse tra le variabili, anche in caso in cui possono esserci variabili correlate che influenzano l'esito della partita.

Infine, Naive Bayes, è una scelta appropriata per la previsione dei risultati delle partite di tennis in quanto si basa su principi statitici che osservano e predicono in base alla probabilità degli eventi. Grazie alla sua natura statistica, questo modello è in grado di analizzare in modo efficace le probabilità e le relazioni tra le variabili presenti nei dati, offrendo così un approccio robusto per la classificazione degli eventi reali.

4.1 Reti neurali

Per approcciare la previsione dei risultati delle partite di tennis, abbiamo adottato un modello di rete neurale artificiale. Le reti neurali sono modelli ispirati al funzionamento del cervello umano e vengono molto utilizzate nell'ambito del machine learning per risolvere problemi di classificazione e regressione.

Il modello di rete neurale utilizzato per la predizione delle partite di tennis è stato sviluppando usando il framework Keras, il quale permette una rapida e semplice creazione del modello. La rete è stata progettata tramite una fase di tuning dei parametri, durante la quale si è cercato di ottimizzare l'architettura della rete e degli iperparametri. Per la realizzazione del modello abbiamo utilizzati strati densamente connessi (Dense).

4.1.1 Tuning iperparametri

Come accennato precedentemente, per la ricerca dei parametri ottimali abbiamo utilizzato un grid search che, ci ha permesso di testare varie combinazioni, per ottenere l'architettura con gli iperparametri che potesse ottimizzare la rete sui dati di training. Come già menzionato, per identificare i parametri ottimali abbiamo eseguito una ricerca per trovare i migliori iperparametri, tramite una grid search. Questa procedura ci ha consentito di esplorare diverse combinazioni di iperparametri al fine di individuare l'architettura della rete neurale

che massimizzasse le prestazioni sui dati di addestramento. Per fare ciò il primo step è stato definire gli iperparametri sul quale eseguire la ricerca:

```
param_grid = {
            'batch_size': [10, 15, 20],
            'epochs': [10, 15, 20],
            'firstLayer': [44, 55, 110],
            'secondLayer': [22, 44],
            "thirdLayer": [12, 22],
            'learning_rate': [0.01],
            "oneMoreLayer": [0, 1]}
```

La "param_grid" ci ha permesso di definire i valori che abbiamo utilizzato per eseguire la grid search sulla rete neurale. Tra questi valori "batch_size" e "epochs" sono di numero ridotto dato che, abbiamo notato come, aumentando questi numeri, non ci fosse un miglioramento di accuratezza in fare di training ma il rischio di overfitting sulla rete. Inoltre la ricerca è stata fatta con una architettura con 2 o 3 livelli nascosti (in base al valore della variabile "oneMoreLayer" ovvero una variabile tecnica che ci ha permesso di testare la rete con stessi iperparametri ma livelli differenti).

Per ottimizzare i parametri tramite una grid search Una volta definito il perimetro dei valori sulla quale eseguire la grid search, abbiamo creato tutte le combinazioni possibili di questi valori e addestrato il modello per ognuna di queste possibilità. Infine, per ogni modello siamo passati alla valutazione dell'accuratezza: Facendo ciò, abbiamo salvato il modello e gli iperparametri con accuratezza migliore. Nonostante esistano librerie dedicate, abbiamo preferito implementare manualmente le logiche della grid search, in modo da avere un maggiore controllo dell'algoritmo. Di seguito viene riportato la parte di codice che si occupa della grid search per la rete neurale:

```
best_accuracy = 0
best_params = {}
best_model = None

for params in product(*param_grid.values()):
    model = create_model(params[2], params[3],
        params[4], params[5])
    model.fit(X_train, y_train, batch_size=params[0],
        epochs=params[1], verbose=0)

    y_pred = model.predict(X_test)
    StratifiedKFold(n_splits=n_fold, shuffle=True, random_state=42)
    y_pred = correctValue(y_pred)
    accuracy = accuracy_score(y_test, y_pred)

    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_params = params
        best_model = model
```

(Nel codice originale ci sono delle istruzioni di "print" che abbiamo omesso dato che sono utili solo quando viene avviato lo script). Il codice esegue un ciclo su ogni combinazione di iperparametri ed istanza un modello, tramite il metodo "create_model". Una volta fatto ciò, si esegue una predizione sui valori di test e successivamente si salva il modello, solo nel caso in cui il valore di accuratezza sia il migliore incontrato fino ad ora.

Per concludere l'analisi sulla ricerca, analizziamo il metodo "create_model":

```
def create_model(firstLayer=11, secondLayer=11,
    thirdLayer=11, learning_rate=0.1, oneMoreLayer=0):
    model = Sequential()
    model.add(Dense(firstLayer, input_dim=X_train.shape[1],
        activation='relu'))
    model.add(Dense(secondLayer, activation='relu'))
    if(oneMoreLayer == 1):
        model.add(Dense(thirdLayer, activation='relu'))
    model.add(Dense(2, activation='softmax'))
    model.compile(loss="binary_crossentropy",
        optimizer=Adam(learning_rate=learning_rate),
        metrics=['accuracy'])
    return model
```

Nel corpo del metodo viene istanziato il modello, in base ai parametri passati al metodo. Alcuni valori, come le funzioni di attivazione, la funzione di "loss" oppure la logica di ottimizzazione sono stati scelti a priori, in base alla tipologia del problema che stiamo affrontando. Di seguito, vengono elencate le decisioni riguardo ai parametri che sono stati scelti, illustrando le considerazioni adottate. I parametri sono:

- Funzioni di attivazione: all'interno della rete sono state utilizzate due funzioni di attivazione, ReLu e Softmax. Per il livello di input e quelli nascosti, abbiamo scelto di utilizzare la funzione di attivazione ReLU. Questa funzione, essendo caratterizzata da semplicità computazionale e dalla capacità di limitare il problema della scomparsa del gradiente, ci ha permesso di ottenere una rapida convergenza durante l'addestramento della rete. Per il livello di output, la scelta è stata di utilizzare la funzione Softmax ovvero, una delle scelte più comuni per problemi di classificazione. La funzione converte gli output della rete in una distribuzione probabilistica sulle classi, garantendo che la somma delle probabilità sia pari a uno. Ciò ci ha permesso di classificare come vincitore il giocatore con la probabilità maggiore.

- Funzione di perdita: la scelta è stata di utilizzare la Binary Cross-Entropy, in quanto, stiamo affrontando un problema di classificazione binaria con due classi da prevedere. Questa funzione è stata selezionata poiché è progettata per contesti di classificazione binaria.

- Algoritmo di ottimizzazione: abbiamo adottato Adam con valore del tasso di apprendimento ottenuto tramite grid search. Questa scelta si è rivelata efficace nel migliorare la convergenza dell'addestramento e nell'ottimizzare le prestazioni complessive del modello, adattando dinamicamente il tasso di apprendimento durante il processo di ottimizzazione.

Questo approccio ci ha consentito di ottimizzare il modello, esplorando diverse architetture della rete. Per la realizzazione della grid search, abbiamo utilizzato l'accuratezza come metrica principale per valutare le varie combinazioni di parametri, ad ogni iterazione, garantendo così una ricerca efficiente e mirata delle migliori configurazioni.

4.1.2 Valutazione prestazioni

Dopo aver analizzato l'architettura e fatto alcune considerazioni, possiamo passare all'analisi dei risultati che il test fatto emergere. Tenendo a mente le considerazioni fatto sopra, bisogna considerare i numeri ottenuti come il risultato di più addestramenti eseguiti a parità di iperparametri e training set. La matrice di confusione mostra buoni valori sulla diagonale principale, che riflettono il numero di previsioni corrette.

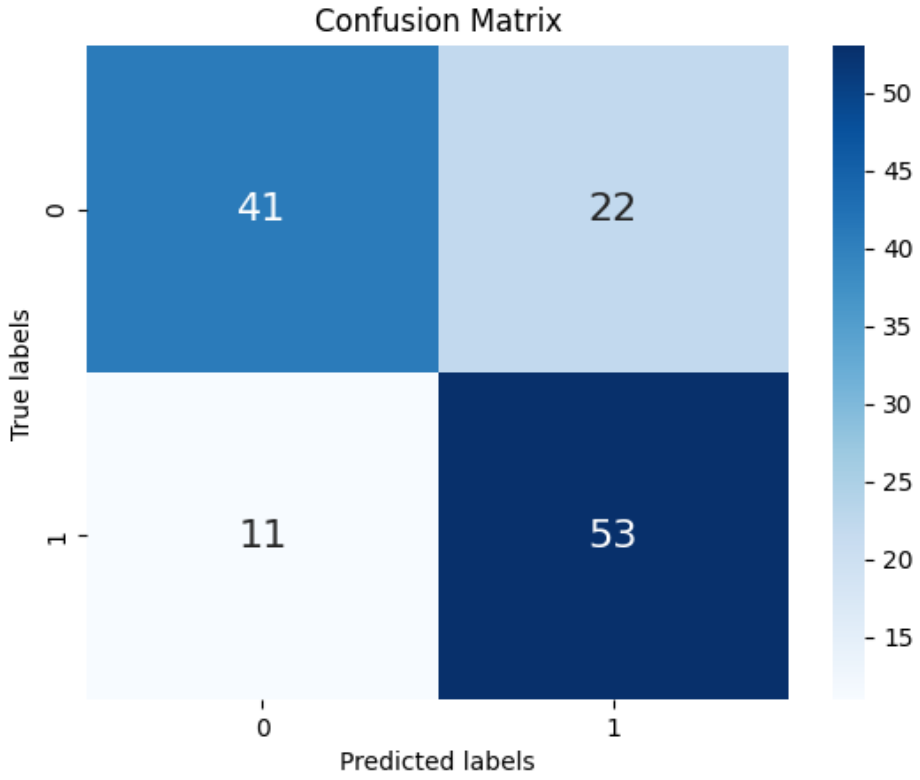

Figure 3: Matrice di confusione per la rete neurale

Un dato importante da considerare è che l'accuratezza complessiva del nostro modello è risultata essere del 74%. Inoltre, attraverso una serie di test eseguiti in varie condizioni, abbiamo osservato che l'accuratezza media si attesta intorno al 70%. Questo ci suggerisce che il modello sia generalmente affidabile nel compiere previsioni, anche se può esserci una variazione nelle prestazioni in base al particolare insieme di dati di test utilizzato.

In definitiva, i risultati ottenuti dimostrano che il nostro modello di rete neurale ha una buona capacità predittiva nel nostro contesto specifico.

4.1.3 Considerazione

Prima di continuare sull'analisi dei risultati ottenuti grazie a questo modello, è importante fare alcune considerazioni sulla rete neurale. Durante l'esecuzione della grid search o training della rete, è emerso un aspetto cruciale da considerare: la variabilità dei risultati ottenuti ad ogni iterazione dell'addestramento. Abbiamo notato come l'accuratezza della rete, a parità di iperparametri e training set, cambi a causa di un diverso addestramento che avviene all'interno del modello. Questa variabilità è principalmente dovuta alla casualità nell'inizializzazione dei pesi della rete neurale e alla dimensione limitata del dataset utilizzato per l'addestramento. Poiché i pesi iniziali della rete vengono scelti in modo casuale, ogni avvio dell'addestramento porta a una configurazione diversa dei pesi, influenzando direttamente il processo di apprendimento della rete. Inoltre, quando si lavora con un dataset di piccole dimensioni, anche piccole variazioni nei dati di addestramento possono avere un impatto significativo sulle prestazioni del modello, portando a risultati diversi a ogni iterazione dell'addestramento. È fondamentale comprendere questa variabilità nei risultati durante la fase di grid search e considerare attentamente la media dei risultati ottenuti per trarre conclusioni affidabili riguardo al modello ottimale da selezionare.

Considerato ciò, abbiamo comunque notato dei valori ricorrenti, durante le varie esecuzioni delle grid search e training dei modelli. In quasi tutti i casi di addestramento, è emerso che i valori di batch pari a 15 e di epochs pari a 20, sono risultati particolarmente ricorrenti. Inoltre, è stata selezionata un'architettura con due livelli nascosti, come migliore. Questa architettura specifica, ha dimostrato una buona capacità di generalizzazione e adattabilità ai dati di input. In particolare, i due livelli nascosti sono stati configurati con 44 e 22 neuroni rispettivamente.

4.2 Alberi di decisione

Gli alberi decisionali sono un algoritmo di machine learning che, che utilizza un modello di decisione a più stadi, per classificare o predire i dati. Sono comunemente utilizzati per la loro facilità di interpretazione e semplicità nell'implementazione. Nell'immagine seguente, viene riportato un esempio di un albero decisionale con una struttura molto elementare, il che può essere vantaggioso per prevenire problema dell'overfitting.

Nell'ambito del progetto è stata impiegata la libreria "scikit-learn" di Python, per implementare l'albero di decisione. Tale libreria include diversi classificatori, tra cui il "DecisionTreeClassifier", il quale è stato utilizzato per l'addestramento del modello desiderato.

4.2.1 Tuning parametri

Al fine di ottenere la migliore classificazione possibile, è stata implementata una procedura di ottimizzazione dei parametri (tuning), simile a quanto già fatto per le reti neurali, per individuare la combinazione ottimale in grado di massimizzare la precisione. Il DecisionTreeClassifier include diversi parametri, quindi, per la fase di tuning sono stati fatti due step operativi:

1. La prima fase di tuning ci ha consentito di esplorare un ampio numero da testare. A differenza di un controllo esaustivo, la "RandomizedSearchCV", esplora un sottoin-

sieme dei paramenti che si stanno considerando e permette di ottenendo un risultato non preciso ma rapido ovvero con un basso costo computazionale.

2. La seconda fase ha enfatizzato una ricerca più precisa, riducendo il numero di parametri da ottimizzare attraverso l'utilizzo della RandomizerSearchCV. In particolare, sono stati considerati degli intervalli ristretti dei parametri che ricorrevano più frequentemente (Ad esempio, se per un certo parametro il numero 2 era molto presente, si testava un intervallo di tipo $[0, 5]$). Successivamente, su questo sottoinsieme di parametri, abbiamo eseguito una GridSearchCV, che, a differenza della RandomizerSearchCV, effettua un controllo esaustivo di tutte le combinazioni possibili, a discapito della velocità.

Questo approccio, suddiviso in due fasi, ci ha consentito di abbreviare i tempi di esecuzione della GridSearchCV, grazie alla nostra conoscenza preliminare dei potenziali migliori iperparametri, senza compromettere la precisione dei risultati. Qui di seguito, vengono riportati i parametri su cui è stato eseguito il tuning:

```
tree_param_grid = {
    'criterion': ['gini', 'entropy'],
    'random_state': [None, 42],
    'splitter': ['random', 'best'],
    'max_features': [None, 'sqrt', 'log2'] +
        np.arange(1, 10).tolist(),
    'min_samples_split': np.arange(2, 10).tolist(),
    'min_samples_leaf': np.arange(1, 10).tolist(),
    'min_weight_fraction_leaf': float_range(0.0, 0.5),
    'min_impurity_decrease': float_range(0.0, 0.5),
}
```

Per ottenere il miglior modello di classificazione dei vincitori delle partite di tennis, in cui la scelta dei parametri, per la grid search, è stata guidata dai risultati ottenuti eseguendo varie volte la random search. Facendo ciò, sono emersi alcuni parametri sul quale ci siamo concentrati per eseguire un tuning più meticoloso.

Utilizzando la "tree_param_grid", riportata sopra, abbiamo ottenuto i seguenti iperparametri per il modello di albero di decisione, riportato di seguito:

```
decision_tree_classifier = DecisionTreeClassifier(
    criterion='gini',
    splitter='best',
    max_features='sqrt',
    min_samples_split=9,
    min_weight_fraction_leaf=0.1,
    random_state=None
)
```

Si può notare come nel contesto della predizione del vincitore delle partite di tennis, il processo di tuning abbai portato all'utilizzo degli iperparametri riportati sopra. Tra questi, possiamo notare l'adozione del metodo "Gini" per la suddivisione dell'albero. Questo parametro

misura la purezza/impurezza dei nodi, in base al valore dell'entropia, favorendo una separazione tra le vincitori e perdenti, senza valutare l'entropia complessiva dei dati.

Per quanto riguarda il metodo di suddivisione, è stato utilizzato il metodo 'best'. Questo ci ha permesso di trovare il punto di suddivisione ottimale, il che è importante per garantire che le divisioni siano fatte in modo accurato e ben bilanciato.

Inoltre, limitare il numero massimo di feature considerate durante la ricerca del miglior punto di suddivisione a 'sqrt' delle feature totali ci aiuta a mantenere il modello meno complesso. In un contesto con molte variabili, questa scelta aiuta a evitare l'overfitting, garantendo che l'albero si concentri solo sulle feature rilevanti. Oltre a ciò, è stato importato il numero minimo di campioni richiesti per eseguire una suddivisione a 9. Questo assicura che le suddivisioni siano fatte solo quando c'è un numero significativo di campioni disponibili. Infine, anche il valore di "min_weight_fraction_leaf", assimene a "max_features" e "min_samples_split", precedentemente analizzati, ci permette di mandatene la struttura dell'albero più semplice, evitando overfitting e avendo un modello più stabile.

Come ultimo iperparametro, è stato usato il valore "None" per il seed. Ciò ci permette di garantire una maggiore casualità nei risultati e una migliore generalizzazione del modello.

4.2.2 Valutazione prestazioni

Come si può notare nella figura 4, l'albero rappresentato è il risultato ottenuto dopo la fase di addestramento con tuning degli iperparametri.

La struttura è composta da pochi livelli di nodi decisionali. Questa caratteristica dell'albero di decisione è importante da evidenziare dato che, con una struttura più semplice, si riesce ad evitare il problema dell'overfitting, che si verifica quando il modello è troppo complesso e si adatta troppo bene ai dati di addestramento, ma non riesce a generalizzare bene sui dati di test.

Un albero di decisione corto può anche avere una minore accuratezza predittiva rispetto a un albero di decisione più profondo. Pertanto, è importante trovare il giusto equilibrio tra la complessità del modello e la sua accuratezza predittiva. In questo caso, l'albero di decisione corto si è dimostrato essere una scelta appropriata, poiché è in grado di prevedere bene i risultati delle partite di tennis senza essere troppo complesso e generare errori a causa all'overfitting.

Grazie a questa semplice struttura, il modello ad albero di decisione ci ha permetto di arrivare a livelli di accuratezza attorno al 75%.

Per valutare l'efficacia del nostro modello di classificazione, abbiamo utilizzato anche in questo caso la matrice di confusione, uno strumento per comprendere rapidamente le performance dell'albero. Q

La matrice di confusione generata riflette i risultati promettenti ottenuti dal nostro modello. Con un'accuratezza del 75%, il modello ha dimostrato di essere in grado di fare previsioni attendibili sui dati di test. Questo livello di accuratezza suggerisce che il modello è in grado di distinguere con successo tra i vincitori e i perdenti delle partite di tennis nella maggior parte dei casi.

Esaminando i valori della matrice di confusione della figura 5, possiamo osservare come il nostro modello ha suddiviso correttamente la maggior parte delle istanze in base alle loro

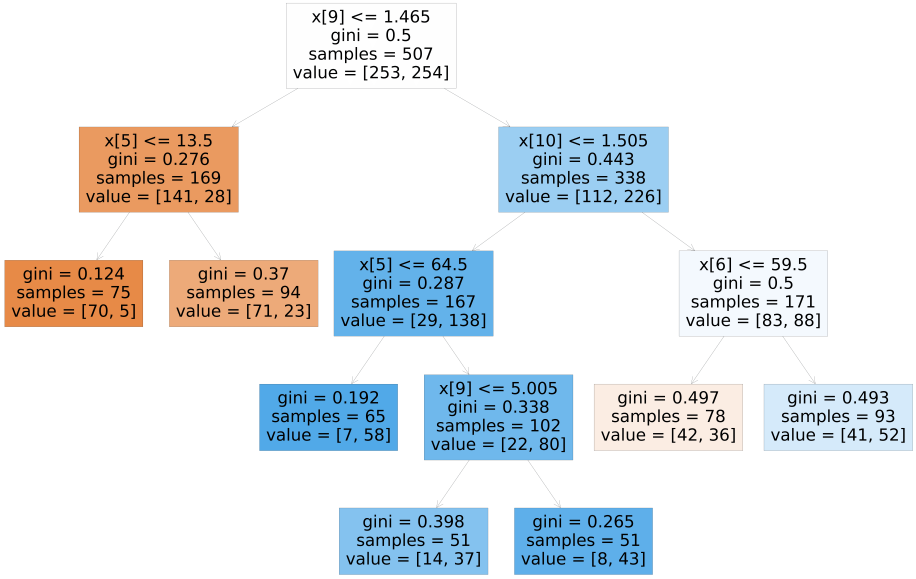

Figure 4: Struttura albero di decisione

classi effettive. Una percentuale elevata di valori sulla diagonale principale indica una buona capacità del modello di distinguere tra le classi.

Basandoci sui risultati osservati nella matrice di confusione, possiamo affermare con fiducia che il nostro modello di classificazione ha dimostrato buone performance nel prevedere i vincitori delle partite di tennis.

4.2.3 Considerazioni

In conclusione, le variazioni nell'accuratezza del modello, simili a ciò che abbiamo notato esserci per le reti neurali eseguendo varie prove di training, ottenute attraverso esecuzioni ripetute degli alberi decisionali possono essere attribuite a diversi fattori. La sensibilità dell'algoritmo ai dati di addestramento e alle inizializzazioni casuali dei parametri, insieme alla variazione nei dati di test e agli iperparametri utilizzati, può portare a differenze nelle prestazioni del modello. È importante considerare questa variabilità e utilizzare tecniche come la media delle prestazioni su più esecuzioni per ottenere una stima più affidabile delle prestazioni del modello. Inoltre, la dimensione ridotta del dataset di addestramento, comune nel contesto delle previsioni delle partite di tennis, può aumentare la sensibilità del modello alle variazioni casuali nei dati e contribuire alla variabilità nelle prestazioni osservate.

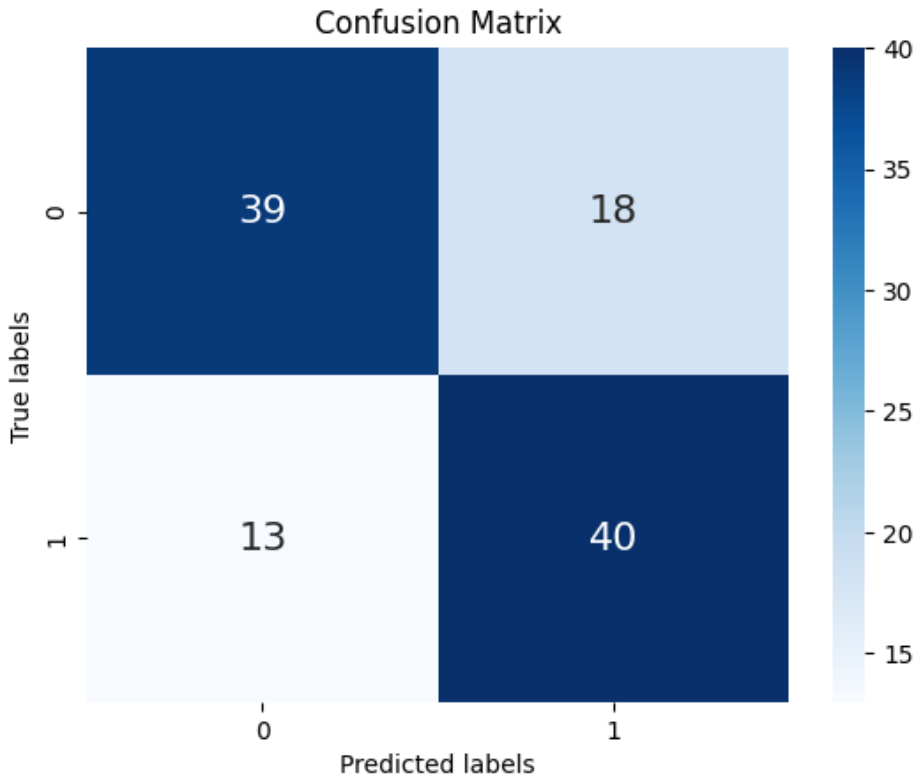

Figure 5: Matrice di confusione per l'albero di decisione

4.3 Naive Bayes

Il classificatore Naive Bayes è un modello di machine learning ampiamente utilizzato per la classificazione di eventi probabilistici, il che lo rende adatto allo scopo di questo progetto. Il suo nome deriva dal teorema di Bayes, che fornisce un modo per calcolare la probabilità condizionata di un evento dato l'occorrenza di un altro evento ovvero le partite che sono già state giocate, per le quali, si conosce il risultato. Il classificatore Naive Bayes assume indipendenza condizionale tra le variabili di input.

Nel contesto della previsione delle partite di tennis, il classificatore Naive Bayes può essere utilizzato per stimare la probabilità che una partita venga vinta da un certo giocatore. Proprio perché si tratta di assunzioni sul futuro, questo modello risulta particolarmente adatto, osservando gli eventi più probabili del passato. Fornendo dati che possono essere disponibili prima della partita, alleniamo il modello a prevedere quale giocatore vincerà.

Il classificatore Naive Bayes utilizza il teorema di Bayes per calcolare la probabilità condizionata inversa, ovvero la probabilità che una partita di tennis possa essere vinta dato il valore delle variabili di input. Questo viene fatto considerando diversi dati provenienti dal campo, come eventuali partite passate tra i due giocatori, i loro rank ATP e le quote di alcuni siti di scommesse.

All'interno della libreria "scikit-learn" esistono diverse categorie di naive Bayes. Tra queste abbiamo implementato la "CategoricalNB", ottimizzata per i dataset che contengono attributi categorici. Per questo motivo, essa risulta molto valida per il dataset utilizzato,

dato l'elevato numero di valori di tipo categorico presenti in esso.

Una volta addestrato il modello utilizzando un set di dati di training contenente le variabili di input e le corrispondenti classi di output , è possibile utilizzarlo per fare previsioni su nuovi dati.

L'efficacia del classificatore Naive Bayes dipende dalla validità dell'assunzione di indipendenza condizionale tra le variabili di input e dalla rappresentatività dei dati di training utilizzati per addestrare il modello. Tuttavia, nonostante la sua semplicità e le assunzioni semplificate, Naive Bayes può spesso fornire risultati accurati, soprattutto in presenza di un grande numero di variabili di input categoriche come nel caso della previsione delle partite di tennis.

4.3.1 Tuning iperparametri

Nella fase di tuning degli iperparametri, abbiamo valutato diverse combinazioni al fine di individuare la configurazione ottimale per massimizzare le performance. Il set di iperparametri considerati includeva l'alpha, il force_alpha e il fit_prior. L'alpha rappresenta il parametro di smoothing Laplace, parametro che viene utilizzato per evitare problemi di probabilità zero quando si calcolano le probabilità condizionate. Il force_alpha, invece, controlla se forzare l'alpha specificato anche se non viene visto durante l'addestramento (utile se si desidera garantire una regolarizzazione uniforme su tutte le feature), mentre il fit_prior determina se stimare le probabilità a priori delle classi dai dati o utilizzare delle probabilità uniformi.

```
naive_param_grid = {
    'alpha': [0.01, 0.1, 0.5, 1.0, 2.0, 3.0, 5.0],
    'force_alpha': [True, False],
    'fit_prior': [True, False],
}
```

Durante la ricerca degli iperparametri ottimali, abbiamo valutato tutte le combinazioni dei parametri riportati sopra. Dopo diversi tentativi, il modello ottimale identificato è stato con un valore di alpha pari a 0.01. Questo valore di alpha più basso, ha prodotto risultati migliori in termini di accuratezza e generalizzazione del modello.

Una volta conclusa la fase di ottimizzazione degli iperparametri, abbiamo istanziato il modello con solamente il valore di alpha, poiché alcuni parametri come il force_alpha e il fit_prior non sono risultati migliori rispetto ai loro valori di default. Questa scelta è stata motivata dalla valutazione ottenuta dalla grid search, che ha suggerito che i valori predefiniti fossero gli ottimi per il nostro problema specifico.

In definitiva, il processo di tuning degli iperparametri ci ha consentito di identificare una configurazione ottimale per il classificatore Naive Bayes. La scelta del valore ottimale di alpha riflette un equilibrio tra la regolarizzazione del modello e la capacità di adattamento ai dati osservati, contribuendo così a garantire una previsione accurata e robusta.

4.3.2 Valutazione prestazioni

Rispetto agli altri modelli, questo classificatore porta con sé i risultati migliori (e più stabili). Questo è probabilmente dovuto alla sua natura probabilistica, molto calzante in

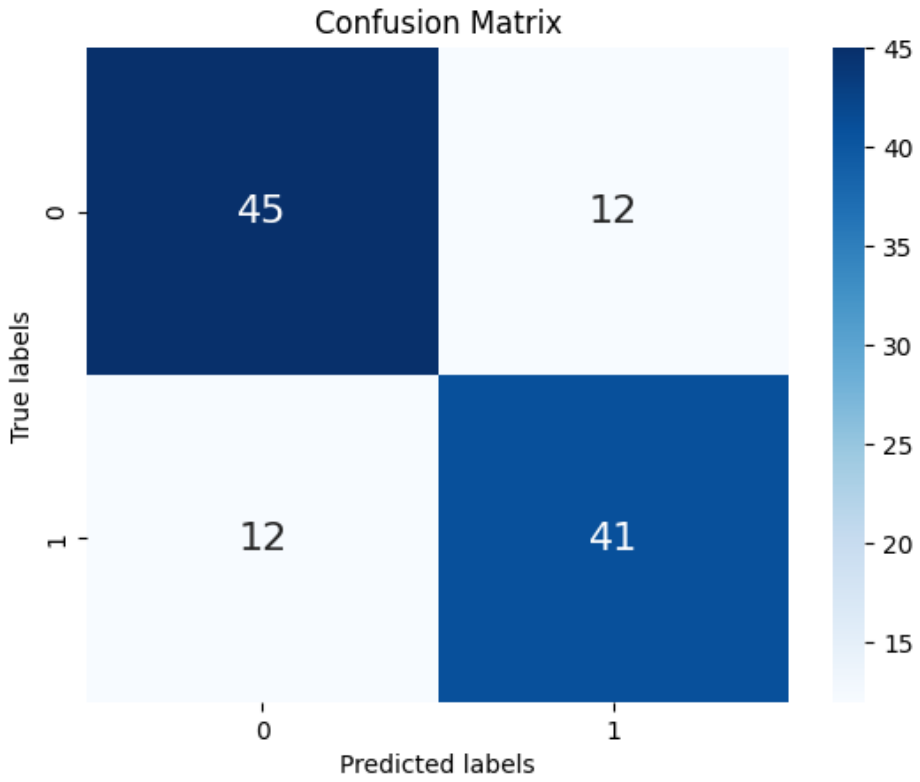

Figure 6: Matrice di confusione per il classificatore bayesiano

questa situazione. Rispetto alle predizioni ha una accuratezza del 78%.

L'algoritmo Naive Bayes, pur essendo un modello di classificazione ampiamente utilizzato e spesso efficace, si basa su un'assunzione che ne limita la portata: l'indipendenza tra le caratteristiche dell'istanza da classificare. Tale assunzione, definita "ingenua", implica che l'algoritmo ignora le relazioni e le interdipendenze che possono esistere tra le diverse variabili che descrivono un caso. Questa assunzione, che a primo avviso potrebbe sembrare negativa, si rivela vantaggiosa in alcuni casi reali (come questo), permettendoci di ottenere un risultato migliore rispetto ad altri modelli.

La Figura 6 mostra la matrice di confusione del classificatore. Notiamo che tende a classificare il giocatore 2 come vincitore molto spesso, sebbene poche partite siano state classificate erroneamente.

La sua fase di tuning è stata meno complicata rispetto all'albero di decisione in quanto questo modello contiene meno parametri da provare.

4.4 Considerazioni

Il modello Naive Bayes ha dimostrato una notevole robustezza durante i test, mantenendo performance coerenti anche eseguendo più training del modello. Questa caratteristica riflette la sua capacità di generalizzazione, ovvero la sua capacità di adattarsi efficacemente a nuovi dati non osservati durante l'addestramento. Il modello Naive Bayes ha continuato a dimostrare una buona capacità di apprendimento e una bassa tendenza all'overfitting. Questo

suggerisce che il classificatore Naive Bayes è in grado di estrarre efficacemente i pattern nei dati e di generalizzare tali pattern per fare previsioni accurate.

5 Valutazione risultati tra i modelli

Nell'analisi dei risultati ottenuti tra i diversi modelli di machine learning, abbiamo considerato la curva ROC (Receiver Operating Characteristic) come indicatore di performance, per confrontare direttamente i tre modelli utilizzati. Questo grafico fornisce una visualizzazione del trade-off tra le predizioni vere corrette e le predizione vere non corrette, di ciascun modello, aiutandoci a valutare la capacità dei modelli nella classificazione dei risultati delle partite di tennis.

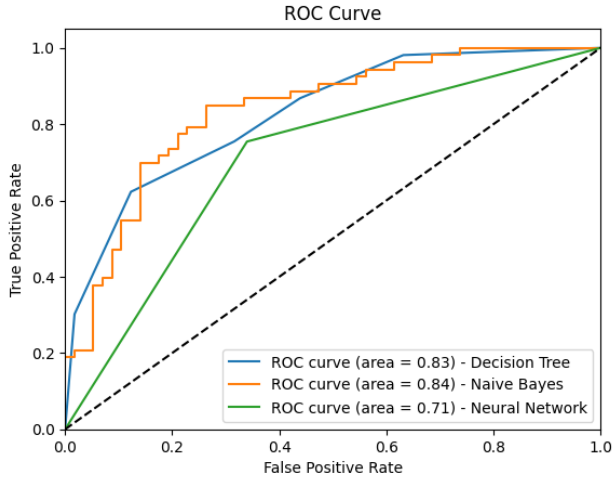

Figure 7: Curve ROC per i modelli implementati

Come si può osserva nella figura 7, Naive Bayes si è distinto come il modello che ha ottenuto la migliore performance complessiva. Questo risultato può essere giustificato dalla natura semplice e intuitiva del classificatore Naive Bayes, che si basa sul principio del teorema di Bayes e sull'assunzione di indipendenza tra le variabili predittive. Inoltre, Naive Bayes è noto per la sua robustezza e stabilità, mantenendo spesso buone performance anche in presenza di dati rumorosi o mancanti.

Inoltre, è importante considerare il contesto specifico della nostra analisi dei risultati delle partite di tennis. La natura relativamente semplice e ben strutturata dei dati del tennis potrebbe favorire l'efficacia di approcci più semplici come Naive Bayes, consentendo al modello di cogliere le relazioni nei dati e di fornire previsioni accurate. Inoltre, l'assunzione di indipendenza delle variabili nel classificatore Naive Bayes ha mostrato essere ben applicabile al contesto dell partite di tennis, consentendo al modello di fornire previsioni coerenti e affidabili.

Complessivamente, questi fattori contribuiscono a giustificare il successo di Naive Bayes rispetto ad altri modelli come reti neurali e alberi di decisione nel contesto specifico della nostra analisi dei risultati delle partite di tennis.

6 Conclusioni

In questa relazione, abbiamo esplorato diverse metodologie di analisi dei dati nel contesto della previsione dei risultati delle partite di tennis. Attraverso l'implementazione e la valutazione di vari modelli di machine learning, abbiamo ottenuto risultati significativi che evidenziano le peculiarità e le sfide di questo ambito.

Le reti neurali si sono dimostrate essere un buon modello per la previsione delle partite, dimostrando la loro capacità di catturare relazioni complesse nei dati e fornire previsioni accurate. La loro flessibilità e la loro capacità di adattarsi a diverse strutture di dati le rendono uno strumento prezioso per la previsione dei risultati nel tennis e in altri contesti. Nonostante ciò, è importante sottolineare che la loro stabilità è compromessa da un dataset di dimensioni ridotte e dalla natura casuale della creazione dei pesi dei neuroni.

D'altra parte, il classificatore Naive Bayes si è dimostrato essere il modello più stabile e affidabile su diversi test, mantenendo un livello costante di accuratezza. La sua semplicità e la sua efficienza lo rendono un'opzione attraente per la classificazione dei risultati delle partite di tennis, soprattutto considerando la sua capacità di gestire dataset di grandi dimensioni.

Gli alberi di decisione hanno dimostrato di possedere ottime capacità di interpretazione dei dati grazie alla loro struttura semplice. Questo permette di ottenere una comprensione intuitiva dei fattori che influenzano i risultati delle partite di tennis, facilitando l'identificazione di pattern e relazioni significative. Tuttavia, è importante notare che gli alberi di decisione possono presentare limitazioni nella capacità di generalizzazione e nell'affrontare la complessità dei dati, soprattutto in confronto ad approcci più avanzati come le reti neurali. Nonostante ciò, l'utilizzo di alberi decisionali può essere ancora vantaggioso per l'esplorazione preliminare dei dati e per ottenere una visione generale dei fattori che influenzano i risultati nel tennis.

In conclusione, questa relazione ha fornito una panoramica esaustiva delle metodologie di analisi dei dati nel contesto del tennis, evidenziando le relative forze e debolezze di ciascun modello. Sulla base di questi risultati, si consiglia l'utilizzo di naive bayes per la previsione dei risultati delle partite di tennis, mentre reti neurali e alberi di decisione possono rappresentare una solida alternativa in termini di stabilità e affidabilità.

Infine, oltre agli approcci analitici esaminati in questa relazione, ci sono molteplici direzioni che potrebbero essere esplorate per migliorare ulteriormente la previsione dei risultati delle partite di tennis. Ad esempio, potrebbe essere interessante esplorare l'utilizzo di tecniche più avanzate come deep learning. Inoltre, l'incorporazione di dati aggiuntivi, come le condizioni meteorologiche o lo stato di forma degli atleti, potrebbe arricchire ulteriormente il modello e migliorare le sue capacità predittive.