UNIVERSITÀ DEGLI STUDI
DI SALERNO

Università degli Studi di Salerno

Dipartimento di Informatica

Laurea Triennale in Informatica

Corso di Fondamenti di Intelligenza Artificiale

Soft17 – An AI Agent for Blackjack

Abbatiello Simone
0512119659
Nappi Vincenzo
0512119813
Niemiec Francesco
0512118999

Anno accademico 2025/2026

Indice

1 Introduzione

Durante la seconda parte del corso di *Fondamenti di Intelligenza Artificiale*, in particolare quella dedicata agli argomenti di **Machine Learning**, è emerso in modo sempre più evidente il forte legame tra le tecniche di apprendimento automatico e il mondo della statistica.

Abbiamo deciso di cogliere l'occasione offerta dallo sviluppo del progetto finale per approfondire tale correlazione, analizzandola in un contesto applicativo concreto. In seguito a una fase di brainstorming, volta a individuare un dominio che consentisse di unire questi due ambiti, abbiamo preso in considerazione i documenti forniti dal docente, nei quali il tema dei *giochi* emerge come un caso di studio ricorrente.

Guidati dalla curiosità e dalla relativa semplicità delle regole, abbiamo quindi scelto il gioco del **Blackjack**. Da questa scelta nasce l'idea di sviluppare un modello che sia in grado di stimare l'esito atteso di una mano di Blackjack sulla base delle informazioni disponibili e sulla base di tutto ciò compiere decisioni sensate e giocare in maniera autonoma.

2 Nozioni basilari di Blackjack

2.1 Overview

Il Blackjack è un gioco di carte dalla forte semplicità, non a caso è tra i più diffusi nei casinò di tutto il mondo. Nella sua versione comune il gioco prevede l'uso di un numero di mazzi standard di 52 carte francesi che va da 1 a 8, e coinvolge solo due tipologie di figure: il "banco" (anche detto "dealer") ed i giocatori. L'obiettivo del singolo giocatore è ottenere una mano il cui valore totale sia pari o quanto più possibile vicino a 21, senza superare tale soglia, e che risulti superiore a quello del banco. Il banco gioca secondo le stesse regole e di conseguenza perde nel momento in cui totalizza un valore minore del giocatore o quando esso sfora la soglia del ventuno. Le carte numeriche mantengono il loro normale valore, le figure valgono ciascuna 10 punti, mentre l'asso può assumere valore 1 o 11 a seconda della scelta del giocatore. Il gioco prende il nome dalla combinazione delle carte asso e jack nero, ma nella sua terminologia viene chiamata blackjack qualunque combinazione di un asso con una figura. Ogni partecipante compete esclusivamente contro il banco, non contro gli altri giocatori al tavolo, e può compiere una serie di scelte (come chiedere carta, stare, raddoppiare o dividere) che incidono direttamente sull'esito della mano. Ciò nonostante nella sua variante a più giocatori nel momento in cui il dealer sfora, tutti i giocatori che non sforano vincono, avendo ognuno di fatto un'influenza, anche se implicita su ciascuna partita. La situazione iniziale di ogni partita è costituita dal dealer con due carte: una scoperta ed una coperta, ed il giocatore sulla base delle due carte che riceve deve stabilire se pescare ancora (**HIT**) o rimanere con solo

quelle due carte (**STAND**). Vi sono anche ulteriori possibilità, lo **SPLIT** può verificarsi quando vengono pescate come prime due carte dal giocatore due carte dallo stesso valore, in tal caso è possibile considerarle come due "prime carte" e giocare due mani contemporaneamente. L'ultimo caso è il **DOUBLE** nel quale si va a raddoppiare la puntata iniziale.

Un elemento distintivo del blackjack rispetto ad altri giochi da casinò è la possibilità di intervento razionale del giocatore sull'andamento della partita. Sebbene il gioco resti fondamentalmente basato sul caso, esistono strategie matematicamente fondate, in grado di aiutare il giocatore. Proprio quest'ultimo aspetto matematico ha portato moltissima attenzione dalla comunità scientifica sul gioco, aprendo le porte prima ad analisi statistiche e poi a nuovi orizzonti come algoritmi genetici e modelli di Machine Learning.

2.2 Vantaggio del banco

Chiamiamo vantaggio del banco la misura quantitativa dello svantaggio strutturale a cui è sottoposto il giocatore di blackjack nel lungo periodo. Esso esprime la perdita media attesa per unità di puntata ed è un indicatore fondamentale per l'analisi matematica ed economica del gioco, in quanto sintetizza l'effetto congiunto delle regole, delle probabilità e dell'ordine delle decisioni.
A differenza di altri giochi da casinò, il vantaggio del banco nel blackjack non è fisso, ma dipende in modo significativo dalla configurazione regolamentare del tavolo e dal comportamento del giocatore. L'origine del vantaggio del banco è riconducibile a una serie di asimmetrie strutturali. La più rilevante è l'ordine di gioco: il giocatore è obbligato ad agire per primo e perde immediatamente la puntata nel caso in cui superi 21, indipendentemente dal risultato finale del banco. Questa regola, apparentemente neutrale, introduce un vantaggio matematico stabile a favore del casinò. A ciò si aggiungono le regole fisse del banco, che ne limitano la discrezionalità ma risultano, nel complesso, statisticamente ottimali.
Ulteriori contributi al vantaggio del banco derivano dalla struttura dei pagamenti.
Il pagamento del blackjack naturale è un elemento cruciale: una remunerazione pari a 3:2 riduce significativamente il margine del casinò, mentre formule meno favorevoli al giocatore, come il pagamento 6:5, possono incrementare il vantaggio del banco. Anche il numero di mazzi utilizzati e le restrizioni sulle opzioni di gioco (raddoppio, divisione, resa) incidono in modo misurabile sul margine complessivo.
Dal punto di vista teorico, il vantaggio del banco nel blackjack è di particolare interesse perché non è imposto esclusivamente dal caso, ma emerge dall'interazione tra regole istituzionali e decisioni individuali. Esso rappresenta quindi un esempio di come un sistema apparentemente equo possa generare, tramite vincoli asimmetrici, un esito sfavorevole a una delle parti.

Attraverso l'attuazione di alcuni comportamenti è possibile ridurre questo svantaggio, primo di tutti è l'applicazione della strategia di base.

2.3 Strategia di base

Parliamo di strategia di base come dell'insieme di decisioni ottimali che il giocatore dovrebbe adottare in ogni possibile configurazione della propria mano e della carta scoperta del banco, assumendo condizioni di gioco standard e l'assenza di informazioni supplementari sull'ordine delle carte. Essa costituisce il riferimento teorico fondamentale per l'analisi razionale del gioco e per la valutazione del vantaggio statistico del banco. Per ciascuna combinazione di mano del giocatore e carta visibile del banco, viene determinata l'azione che massimizza il valore atteso della giocata. È importante effettuare una distinzione tra **hard-hand**, in cui l'asso ha valore fisso pari a 1, e **soft-hand**, in cui l'asso può assumere valore 11 senza rischio immediato di sballo. L'adozione coerente della strategia di base consente di ridurre il margine di vantaggio del banco a valori minimi, che in condizioni regolamentari favorevoli possono scendere a meno del'uno percento. È tuttavia importante sottolineare che la strategia di base non elimina il rischio né garantisce vincite nel breve periodo: essa opera esclusivamente sul piano statistico, ottimizzando le decisioni nel lungo termine. Nel dettaglio questa strategia è stata formalizzata nel 1956 da Roger R. Baldwin, nell'articolo "The Optimal Strategy in Blackjack". Di seguito riportiamo le tabelle relative alla strategia di base del Blackjack.

BLACKJACK BASIC STRATEGY CHART

Legend: : Hit : Double : Stand : Dealers Upcard

	DEALER UPCARD									
Hard Total	2	3	4	5	6	7	8	9	10	A
8 or Less	H	H	H	H	H	H	H	H	H	H
9	H	D	D	D	D	H	H	H	H	H
10	D	D	D	D	D	D	D	D	H	H
11	D	D	D	D	D	D	D	D	D	H
12	H	H	S	S	S	H	H	H	H	H
13-16	S	S	S	S	S	H	H	H	H	H
17+	S	S	S	S	S	S	S	S	S	S
Soft Total	2	3	4	5	6	7	8	9	10	A
A,2	H	H	H	D	D	H	H	H	H	H
A,3	H	H	H	D	D	H	H	H	H	H
A,4	H	H	D	D	D	H	H	H	H	H
A,5	H	H	D	D	D	H	H	H	H	H
A,6	H	D	D	D	D	H	H	H	H	H
A,7	D	D	D	D	D	S	S	H	H	H
A,8+	S	S	S	S	S	S	S	S	S	S
Pairs	2	3	4	5	6	7	8	9	10	A
2,2 + 3,3	S	S	S	S	S	S	H	H	H	H
4,4	H	H	H	S	S	H	H	H	H	H
5,5	D	D	D	D	D	D	D	D	H	H
6,6	S	S	S	S	S	H	H	H	H	H
7,7	S	S	S	S	S	S	H	H	H	H

3 Tecnologie utilizzate

3.1 Overview

Abbiamo deciso di sfruttare l'ambiente condiviso online **Google Colab** e di conseguenza il linguaggio scelto per raggiungere l'obiettivo prefissato è **python**. In particolare abbiamo utilizzato le seguenti librerie:

- **pandas**: permette l'esplorazione a la manipolazione di dati in modo semplice e flessibile;
- **numpy**: ottimizza la gestione di ampi dati multidimensionali come array e matrici in modo efficiente e fornisce una serie di funzioni matematiche per la loro gestione;
- **seaborn e matplotlib**: permettono la visualizzazione di dati tramite schemi o grafici informativi. Sono state principalmente utilizzata per mostrare i risultati dei training degli algoritmi implementati.
- **scikit-learn**: fornisce tool semplici ed efficienti che permettono la realizzazione di algoritmi di machine learning. Permette inoltre di valutare il modello con il calcolo delle metriche integrato.
- **tkinker**: permette di sviluppare interfacce grafiche per i programmi python implementati.

4 Descrizione dell'agente

4.1 Obiettivi di business

L'obiettivo del progetto è lo sviluppo di un **agente intelligente** in grado di:

- selezionare, per ogni mano osservabile, l'azione più appropriata al fine di massimizzare il *win rate*. Le azioni considerate sono:
  - **Hit**: richiedere una carta aggiuntiva;
  - **Stand**: mantenere la mano attuale;
- prendere decisioni esclusivamente sulla base della propria mano e della carta scoperta del dealer, senza conoscere le carte rimanenti nel mazzo;
- migliorare le proprie prestazioni nel tempo, apprendendo dai dati osservati.

4.2 Specifica dell'ambiente (PEAS)

L'ambiente in cui opera l'agente viene descritto attraverso la specifica **PEAS** (Performance, Environment, Actuators, Sensors).

Performance: la misura di prestazione dell'agente è legata alla sua capacità di massimizzare il risultato economico nel gioco. In particolare vengono considerate nelle prime due pipeline l'accuratezza e robustezza nella previsione dell'esito della mano (vittoria o non sconfitta) a partire dallo stato informativo disponibile al momento della decisione, valutate tramite metriche di classificazione supervisionata (F1-score come metrica principale, affiancata da precision e recall). Per quanto riguarda il reinforcement learning invece andiamo a considerare la percentuale di vittorie dell'agente.

Environment: l'ambiente è rappresentato da un tavolo di Blackjack comprendente:

- uno o più mazzi di carte con distribuzione stocastica;
- un dealer che segue una policy fissa;
- la mano del giocatore;
- l'insieme delle regole del gioco.

Le proprietà dell'ambiente sono:

- **Parzialmente osservabile**: l'agente conosce solo la propria mano e la carta scoperta del dealer;
- **Stocastico**: l'ordine delle carte nel mazzo è pseudocasuale;
- **Sequenziale**: ogni azione influenza l'esito finale della mano;
- **Statico**: l'ambiente non cambia mentre l'agente sta deliberando;
- **Discreto**: il numero di stati e azioni è finito;
- **A singolo agente**: il dealer segue una strategia prefissata e non è considerato un agente decisionale.

Actuators: gli attuatori dell'agente corrispondono alle azioni disponibili: Hit, Stand.

Sensors: l'agente percepisce:

- le carte della propria mano;
- la carta scoperta del dealer;
- le azioni consentite in un determinato stato (ad esempio la possibilità di effettuare double o split);
- l'esito finale della mano (vittoria, sconfitta o pareggio).

4.3 Scelta della prima strategia

Un primo approccio al problema avrebbe potuto basarsi sulla teoria dei giochi e sull'utilizzo di algoritmi genetici. Tuttavia, tale soluzione presenta diverse criticità:

Variabilità dei cromosomi: la rappresentazione delle soluzioni risulterebbe altamente variabile, rendendo complessa la codifica delle strategie.

Complessità dello spazio di ricerca: l'elevato numero di stati e azioni comporta uno spazio di ricerca estremamente ampio, con conseguenti tempi di elaborazione proibitivi.

Inaccuratezza della fitness: la natura stocastica dell'ambiente rende la funzione di fitness fortemente influenzata dalla casualità.

Alla luce di queste considerazioni, abbiamo deciso di affrontare il problema tramite tecniche di **apprendimento supervisionato**. In questo contesto, l'agente apprende una funzione che associa a ogni stato osservabile del gioco un'etichetta nota, rappresentata dall'esito della mano, utilizzando un dataset di mani simulate.

5 Metodologia e Raccolta dati

5.1 Metodologia di sviluppo: CRISP-DM

Lo sviluppo del progetto è stato guidato dal modello CRISP-DM (Cross-Industry Standard Process for Data Mining), un processo standard ampiamente adottato per la realizzazione di sistemi di data mining e machine learning. Tale modello fornisce una visione strutturata e iterativa del ciclo di vita di un progetto di apprendimento automatico, suddividendolo in fasi ben definite.

In particolare, il CRISP-DM prevede le seguenti fasi principali:

- **Business Understanding**: definizione degli obiettivi del sistema e dei criteri di successo;
- **Data Understanding**: raccolta dei dati, analisi esplorativa e valutazione della loro qualità;
- **Data Preparation**: pulizia dei dati e costruzione delle feature rilevanti;
- **Modeling**: selezione e addestramento dei modelli di apprendimento automatico;
- **Evaluation**: valutazione delle prestazioni del modello rispetto agli obiettivi prefissati;

- **Deployment**: integrazione del modello all'interno di un sistema utilizzabile.

Nel contesto del presente lavoro, le fasi di *Business Understanding* e *Data Understanding* sono state affrontate attraverso la definizione degli obiettivi dell'agente intelligente e l'analisi preliminare dei dati generati dal simulatore di Blackjack. Le successive operazioni di data cleaning e feature engineering rientrano nella fase di *Data Preparation*, il cui obiettivo è rendere i dati idonei all'addestramento di modelli di apprendimento supervisionato. In conclusione, il modello CRISP-DM è stato utilizzato come **riferimento metodologico** per strutturare il progetto, senza l'obiettivo di coprirne esplicitamente tutte le fasi, ma adottandone i principi per guidare le scelte progettuali effettuate.

5.2 Scelta del dataset

Per la scelta del dataset sono state considerate due possibili strategie:

1. Generare un dataset sintetico tramite un simulatore di Blackjack, in grado di produrre un numero arbitrario di mani realistiche.

2. Utilizzare dataset già disponibili online [1][2].

La prima opzione come pro avrebbe garantito un grande controllo sui dati, ma un errore minimo nel simulatore o una nostra scelta sbagliata e/o inesperta nella categorizzazione dei dati avrebbe avuto un impatto troppo alto sul modello. La seconda opzione, seppur all'apparenza più facile, avrebbe potuto celare delle insidie, come una difficile comprensione dei dataset oppure "fastidi" dovuti alla loro enorme dimensione. Avendo valutato per bene ambo le alternativi ci siamo trovati inizialmente di fronte ad un bivio, optando poi per una soluzione "ibrida". Il più corposo tra i principali due dataset trovati contava cinquanta milioni di mani di blackjack simulate, un numero talmente elevato che il training del modello con le risorse a nostra disposizione sarebbe risultato infinito. Esso però riportava anche una repository **GitHub** con il programma usato per simulare queste mani. Abbiamo quindi preso questo programma e generato un dataset con circa un milione di mani giocate, una quantità tale da poter effettuare un buon training in tempi tutto sommato contenuti. È importante notare come la scelta di mantenere inalterato l'algoritmo nasce da due ragioni: innanzitutto per mantenere inalterati gli ottimi risultati relativi all'usabilità del dataset, la seconda (nonchè forse la più importante) è quella di voler andare ad operare direttamente sui dati anzichè sulla loro generazione.

[1] (`https://www.kaggle.com/datasets/mojocolors/900000-hands-of-blackjack-results`)
[2] (`https://www.kaggle.com/datasets/dennisho/blackjack-hands/data`)

5.3 Regole seguite

Per generare il dataset si è quindi scelto di utilizzare le stesse regole indicate dal dataset (`https://www.kaggle.com/datasets/dennisho/blackjack-hands/data`). Le regole in questione sono:

- **Shoe da 8 mazzi** (penetrazione di 6,5 mazzi)
- La prima carta dello shoe viene scartata
- Il Blackjack paga 3:2
- È possibile raddoppiare su qualsiasi combinazione iniziale di due carte
- Il raddoppio dopo lo split è consentito
- È possibile splittare qualsiasi coppia iniziale fino a un massimo di 4 mani
- Non è consentito il re-split degli Assi
- Dopo lo split degli Assi viene distribuita una sola carta per mano e non è possibile ottenere Blackjack
- Non è consentita la resa dopo uno split
- Il banco pesca su soft 17 (H17)

Bisogna notare anche quanto segue:

- Tutte le mani sono giocate **heads-up** (un solo giocatore contro il banco)
- La puntata iniziale per ogni mano è sempre pari a 1
- Le carte 10, J, Q e K sono considerate equivalenti e registrate come 10
- Gli Assi sono sempre registrati come 11, indipendentemente dal loro valore effettivo nella mano
- I semi non vengono considerati e non sono registrati
- Il **Running Count** e il **True Count** (troncati all'intero) sono calcolati con il sistema **Hi-Lo** e registrati all'inizio di ogni mano, prima della distribuzione delle carte
- Il numero di carte rimanenti nello shoe è registrato all'inizio di ogni mano e include anche le carte che non verranno utilizzate a causa di una penetrazione inferiore al cento percento

6 Prima pipeline

6.1 Esplorazione

Il dataset generato necessità di varie operazioni di pulizia dei dati ed in generale di data engineering per renderlo consono ai nostri scopi. C'è necessità di **Feature scaling**, **Feature construction**, **Feature extraction**. Iniziamo però da una fase iniziale di esplorazione dei dati, la quale serve per comprendere su quali aree agire. Data la natura condivisa di questa fase abbiamo deciso di usare *Colab* (quindi i comandi che andremo a specificare successivamente faranno riferimento alla sua sintassi). Come prima cosa quindi abbiamo importato il file del dataset (in formato .csv) per visionare le caratteristiche. In particolare il dataset iniziale era composto da:

- **shoe_id**: Identificativo del mazzo utilizzato per la mano.
- **cards_remaining**: Numero di carte rimanenti nel mazzo all'inizio della mano.
- **dealer_up**: Carta scoperta del dealer.
- **initial_hand**: Mano iniziale del giocatore.
- **dealer_final**: Mano finale del dealer.
- **dealer_final_value**: Valore finale della mano del banco.
- **player_final**: Mano finale del giocatore dopo tutte le azioni.
- **player_final_value**: Valore finale della mano del giocatore.
- **actions_taken**: Sequenza di azioni effettuate dal giocatore.
- **run_count**: Numero di round giocati nella simulazione.
- **true_count**: Conteggio all'inizio della mano.
- **win**: Risultato della mano per il giocatore.

Successivamente abbiamo individuato *WIN* come colonna utile per la nostra classificazione. Di fatto abbiamo analizzato quali fossero i valori che potesse assumere e li abbiamo raggruppati in tre categorie, ovvero minori di zero, uguali a zero e maggiori di zero. Fatto ciò abbiamo visualizzato il numero di istanze per ciascuna categoria:

- 421940 istanze con win **maggiore di zero.**
- 493992 istanze con win **minore di zero.**
- 84068 istanze con win **uguale a zero.**

A primo impatto il dataset presentava uno sbilanciamento verso le "sconfitte", tuttavia abbiamo scelto di non reputarlo così influente alla luce di due considerazioni: la prima è che è insito alla natura del gioco (favorevole al banco) che siano presenti molte più sconfitte che vittorie, e la seconda è che andando a considerare semplicemente i casi "favorevoli" ovvero quelli dove la win è maggiore o uguale a zero, il dataset sarebbe apparso come bilanciato. Successivamente abbiamo verificato quanti valori nulli fossero presenti nelle colonne, i risultati hanno mostrato che nessuna colonna presentava valori nulli. Ciò non era assolutamente inaspettato data la natura "sintetica" dei dati.

6.2 Data Cleaning

Siccome tutte le colonne del dataset non presentano dati mancanti, questa fase non comprende all'atto pratico nessun cambiamento.

6.3 Feature Scaling

Avendo precedentemente specificato che non vogliamo concentrarci sull'aspetto economico delle singole puntate, ci viene naturale il voler normalizzare i valori di win per identificare semplicemente i tre casi possibili:

- Vittoria (attualmente $0 < x <= 7$, vorremmo solamente 1)
- Pareggio (attualmente $== 0$)
- Sconfitta (attualmente da < 0 a -7, vorremmo solamente -1)

Per fare ciò abbiamo valutato diverse opzioni:

Normalizzazione min-max: il problema di questo tipo di normalizzazione era che a noi interessava mantenere anche lo zero a fini statistici ma applicandola ciò non sarebbe possibile (se invece avessimo considerato lo 0 come parte della vittoria allora sarebbe stata perfetta)

Normalizzazione Z-score: qui non avremmo avuto pieno controllo ma la trasformazione sarebbe stata attuata sulla densità media.

Abbiamo quindi optato per una strategia di discretizzazione con soglie che ci permette di fare esattamente ciò che ci serve.

6.4 Feature Engineering

Nonostante il lavoro effettuato nella fase precedente, il dataset risultava fortemente inadeguato ai nostri scopi, quindi in questa fase tramite tecniche di Feature Selection e Feature Construction abbiamo cercato di plasmarlo al meglio per i nostri scopi.

6.4.1 Feature Selection

Iniziamo da ciò che siamo andati a rimuovere. Primo step è stato quello di eliminare shoe_id in quanto è un semplice indicatore del mazzo di carte e non ha alcuna correlazione con la decisione nel nostro contesto. Fatto ciò passiamo all'eliminazione di colonne che potrebbero causare **Data Leakage**, in particolare facciamo riferimento a:

- **dealer_final**: rappresenta la mano finale del dealer e, di conseguenza, non è un'informazione disponibile al momento della decisione del giocatore.
- **dealer_final_value**: valore finale della mano del dealer; per le stesse motivazioni espresse sopra, questa informazione è osservabile solo a fine mano.
- **player_final**: descrive la mano finale del giocatore, includendo carte ottenute attraverso azioni future non ancora osservabili nel momento della decisione.
- **player_final_value**: valore finale della mano del giocatore; analogamente alla variabile precedente, non è disponibile prima del completamento della sequenza di azioni.
- **actions_taken**: rappresenta un vero e proprio caso di *data leakage*, in quanto codifica direttamente la sequenza di azioni che il modello dovrebbe invece apprendere a prevedere.

Detto ciò abbiamo eliminato anche run_count, true_count e cards_remaining perchè relativi al conteggio delle carte.

6.4.2 Feature Construction

Una volta conclusa la fase di *Feature Selection*, è stato necessario modificare alcune colonne del dataset al fine di renderle utilizzabili dal modello. La prima trasformazione riguarda la variabile **initial_hand**, che è rappresentata come una lista di carte. Questa rappresentazione risulta poco adatta all'utilizzo diretto da parte del modello; per questo motivo si è deciso di scomporla in tre differenti variabili:

- **player_sum**: la somma dei valori di tutte le carte presenti nella mano del giocatore;
- **player_is_soft**: variabile binaria che indica se la mano è *soft*, ovvero se contiene un asso valutabile come 11;
- **player_pair**: variabile binaria che indica se la mano è composta da due carte dello stesso valore e quindi se è possibile effettuare lo *split*.

6.5 Colonne finali del Dataset

Al termine della fase di elaborazione, il dataset finale risulta composto dalle seguenti variabili:

- **player_sum**
- **player_is_soft**
- **player_pair**
- **dealer_up**

La variabile target è **win**, che indica l'esito della decisione presa dal giocatore, assumendo i valori di vittoria e sconfitta.

6.6 Modeling

Prima di parlare degli algoritmi utilizzati occorre fare delle premesse. Per scegliere i migliori parametri adatti agli algoritmi e il problema che stiamo affrontando abbiamo deciso di utilizzare una **griglia di ricerca** per comprendere i valori adatti per i vari iperparametri. In particolare **GridSearchCV**, una classe contenuta nella libreria scikit-learn che ci permette di testare più combinazioni di parametri automaticamente evitando di rieseguire manualmente i training (e sfrutta cross-validation). La prima operazione effettuata è lo splitting del dataset per distinguere i dati di test e i dati per il training. Nello specifico l'80% dei dati sarà utilizzato per il training e il 20% per il testing. Avviene successivamente un ulteriore split dei dati di training distinguendo tra training vero e proprio e validazione, nello specifico il 78.5% per il training e il restante per la validazione. È stato inoltre fissato un seme randomico per rendere i test ripetibili. Infine *stratify* permette di suddividere il dataset in modo che ogni sottoinsieme mantenga la stessa identica proporzione di classi del dataset originale. Per la prima versione di questa pipeline (Pipeline 1A) l'algoritmo utilizzato è un albero decisionale di classificazione (DecisionTreeClassifier) specificando i seguenti parametri:

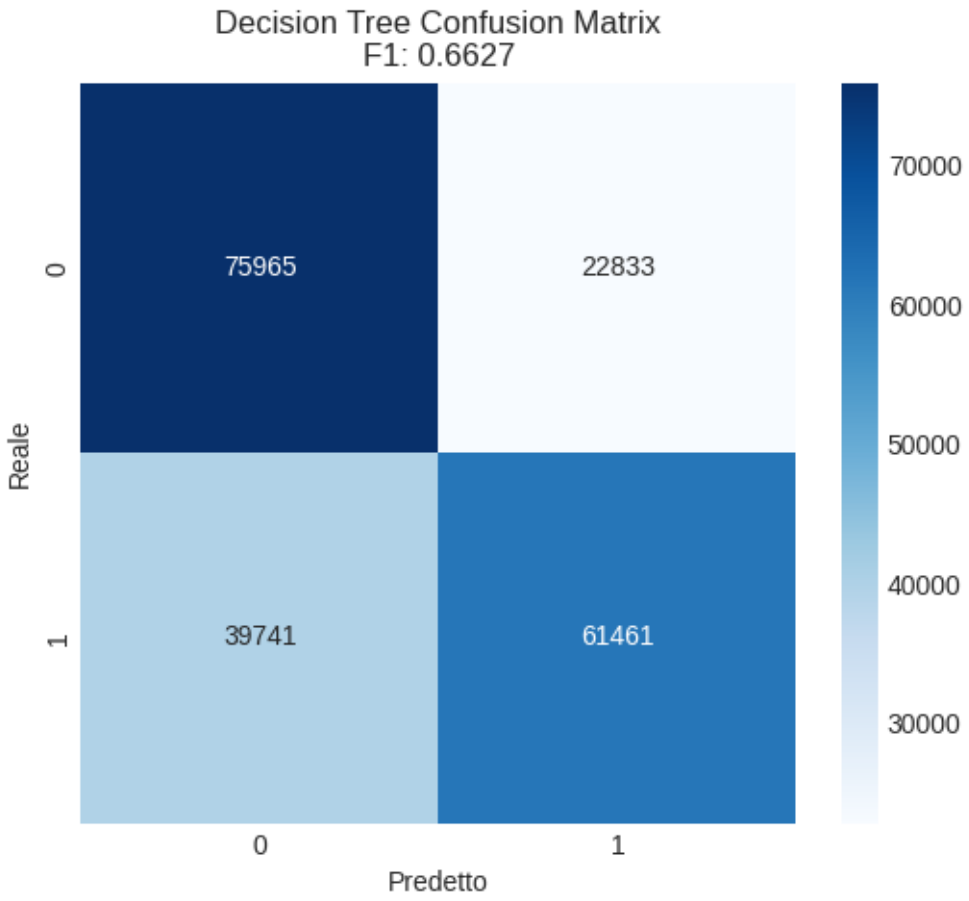
- **max_depth:** che specifica la profondità massima che l'albero può raggiungere con i seguenti valori: [10,20,30]
- **min_samples_split:** che determina numero minimo di campioni prima di effettuare un'ulteriore ramificazione. I valori sono: [2,5]
- **min_samples_leaf:** che determina il numero minimo di campioni che devono essere presenti nelle foglie. I valori sono: [1,2]

6.7 Valutazioni finali

I migliori parametri scelti dalla griglia sono: max_depth = 10, min_samples_leaf= 1, min_samples_split= 2.
Le metriche invece ci hanno fornito i seguenti risultati:

- F1-Score Test: 0.6627
- Accuracy: 0.6871
- Precision: 0.7291
- Recall: 0.6073

L'algoritmo ha portato a dei risultati ragionevoli che bilanciano le prestazioni ma forniscono anche predizioni discretamente accurate sul finale della partita, nonostante la sua natura stocastica.



7 Seconda pipeline

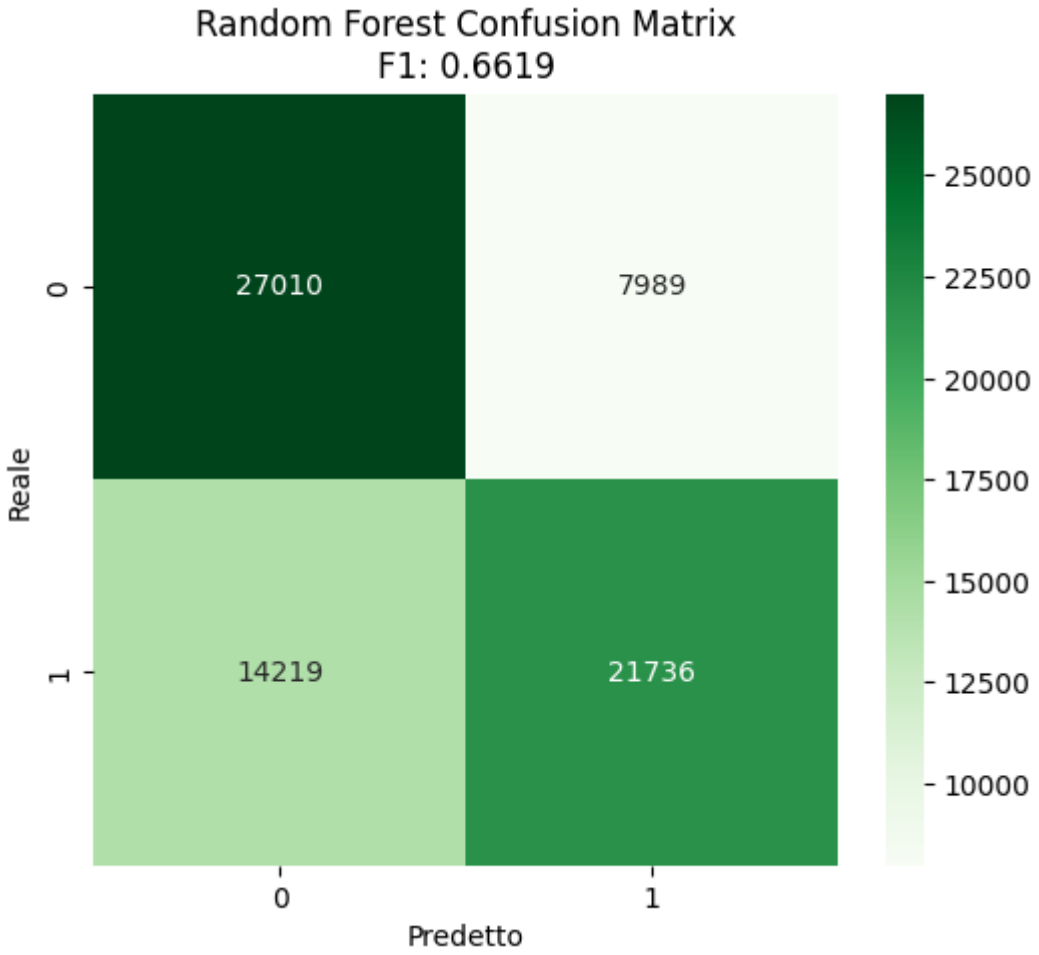
Per la seconda versione della pipeline (Pipeline 1B) è stato adottato un approccio basato su **ensemble di alberi decisionali, ovvero Random Forest** (*RandomForestClassifier*), mantenendo invariate tutte le fasi di preprocessing, feature engineering e suddivisione del dataset rispetto alla pipeline 1A, al fine di garantire un confronto equo tra i modelli.
Analogamente a quanto fatto per la pipeline 1A, anche in questo caso è stato utilizzata una **griglia di ricerca** per la selezione degli iperparametri, operando sulla medesima formulazione del problema come task di classificazione binaria. In particolare:

- vengono utilizzate le stesse feature;
- non vengono introdotte ulteriori trasformazioni dei dati;
- il processo di selezione degli iperparametri è identico a quello adottato per la pipeline 1A.

I principali iperparametri considerati per la Random Forest sono:

- **n_estimators:** numero di alberi presenti. I valori testati sono: [100, 200];
- **max_depth:** profondità massima consentita per ciascun albero. I valori considerati sono: [15, 20];
- **min_samples_split:** numero minimo di campioni richiesto per effettuare una suddivisione interna. I valori testati sono: [2, 5].

L'introduzione di questa pipeline consente di valutare l'impatto di un modello più complesso rispetto a un singolo albero decisionale sul problema considerato. Dai risultati sperimentali emerge che la pipeline 1B non fornisce miglioramenti sostanziali rispetto alla pipeline 1A in termini di prestazioni predittive. Le metriche di valutazione risultano infatti comparabili, indicando che l'aumento della complessità del modello non si traduce in un beneficio significativo nel contesto analizzato. Di conseguenza, la pipeline 1B viene considerata principalmente come termine di confronto, mentre la pipeline 1A rappresenta una soluzione più efficiente e interpretabile.


Random Forest Confusion Matrix
F1: 0.6619
27010
7989
14219
21736
Reale
0
1
0
1
Predetto
25000
22500
20000
17500
15000
12500
10000

8 Reinforcement Learning

8.1 Motivazioni dietro al cambio di approccio

Sebbene i precedenti approcci si siano dimostrati efficienti nell'affrontare il problema in esame, hanno anche evidenziato delle problematiche:

- il Blackjack è un **processo decisionale sequenziale**, in cui ogni azione influenza gli stati futuri della partita, tuttavia ciò non viene riflesso con questi approcci;
- l'assenza di feedback intermedio;
- l'azione ritenuta migliore viene dedotta indirettamente a partire dalle predizioni del finale delle partite del modello: ciò risulta essere meno adatto a problemi in cui l'obiettivo principale è l'ottimizzazione di una sequenza di decisioni.

Inoltre i risultati ottenuti evidenziano che il modello riesce a individuare circa il 60% dei veri casi positivi. Questo evidenzia una percentuale di falsi negativi fin troppo alta. Di fatto quindi l'informazione di "Perdita o non Perdita" ed un insieme di feature forse troppo ridotto, portano il nostro modello a non svolgere propriamente il compito prestabilito. Rendendoci conto che per la costruzione di un modello che fosse in grado di giocare a Blackjack in maniera efficiente non bastava affrontare il problema come un problema di apprendimento supervisionato, abbiamo deciso di cambiare il nostro approccio.

8.2 Introduzione al ruolo di Markov

Parallelamente al corso di *Fondamenti di Intelligenza Artificilae*, uno degli esami che abbiamo seguito era quello di *Musimatica*. All'interno di tale corso è stato introdotto il concetto di "Catena di Markov"[1], come un modello matematico per descrivere sistemi che hanno un'evoluzione casuale guidata da distribuzioni di probabilità. Tale modello possiede due componenti: uno insieme di stati e le probabilità di transizione da uno stato all'altro. Abbiamo quindi identificato un facile parallelismo col blackjack, ma con nozioni veramente basilari non sarebbe stato possibile andare avanti. Abbiamo quindi condotto delle ricerche basilari, necessarie per capire se stessimo ancora sbagliando strada. Per comprenderne i risultati è quindi necessario dare una serie di definizioni.

8.2.1 Processo di Markov

Definiamo un **Processo di Markov** come un processo che non ha memoria degli stati precedenti, e di conseguenza lo stato futuro è dettato esclusivamente dallo stato attuale. Di questi processi, quelli che hanno un numero finito (o contabile) di stati sono detti **Catene di Markov** [2].

8.2.2 Stato di Markov

Il motivo dietro al quale non si tiene conto degli stati precedenti è che ciascuno stato cattura già le informazioni più rilevanti degli stati precedenti. Possiamo dare una definizione più formale di stato markoviano tramite questa formulazione: *Uno stato S_t è detto stato di Markov, se e solo se*

$$P(S_{t+1} \mid S_1, S_2, \ldots, S_t) = P(S_{t+1} \mid S_t).[3]$$

8.2.3 Markov Reward Process

Un **Processo di Ricompensa di Markov** è una catena di Markov con valori di ricompensa (reward).
Un Processo di Ricompensa di Markov è definito come la quadrupla:

$$(\mathcal{S}, \mathbf{P}, \mathcal{R}, \gamma),$$

dove:

- $\mathcal{S}$ è un insieme finito di stati;
- $\mathbf{P}$ è la matrice di transizione degli stati;
- $\mathcal{R}$ è la funzione di ricompensa, tale che

$$\mathcal{R}_s = E[R_{t+1} \mid S_t = s];$$

- γ è il fattore di sconto, con $\gamma \in [0, 1]$.

In particolare notiamo questo γ che indica quanto valore attribuiamo alle ricompense future rispetto a quelle immediate. Quindi se è uguale a zero ci interessano solo le ricompense immediate, al contrario se è uguale a uno consideriamo le ricompense future quasi allo stesso modo di quelle attuali.

8.2.4 Markov Decision Process

Un **Markov Decision Process** (MDP) è un processo di ricompensa di Markov con azioni. È un ambiente in cui tutti gli stati sono Markov, è definito dalla quintupla:

$$(\mathcal{S}, \mathcal{A}, \mathbf{P}, \mathcal{R}, \gamma),$$

dove $\mathcal{A}$ è un set finito di azioni a. Questo particolare tipo di processo è spesso associato ad una policy π, ovvero una distribuzione delle azioni a dato lo stato s:

$$\pi(a \mid s) = P(A_t = a \mid S_t = s)$$

L'obiettivo è quello di trovare una policy ottimale, ovvero una strategia che massimizza la ricompensa cumulativa a lungo termine, e che quindi rispetto alle altre policy produrrà un risultato almeno uguale. Definiamo in particolare un **MDP Parzialmente Osservabile** (POMDP) un processo decisionale di Markov con stati nascosti, ovvero composto dalla 7-tupla:

$$(\mathcal{S}, \mathcal{A}, \mathcal{O}, \mathbf{P}, \mathcal{R}, \mathcal{Z}, \gamma),$$

dove $\mathcal{O}$ un insieme finito di osservazioni e $\mathcal{Z}$ la funzione di osservazione.

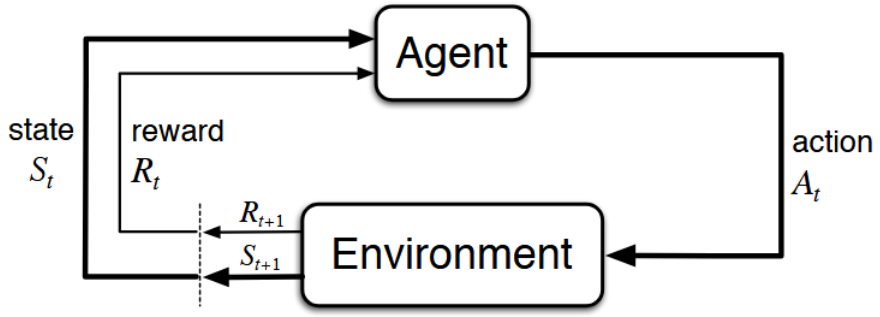
8.3 Markov ed il Blackjack

Il problema del Blackjack è stato modellato come un **MDP**.
Lo stato dell'ambiente è definito esclusivamente dalle informazioni osservabili dal giocatore al momento della decisione, ovvero: la somma della propria mano, la presenza di una mano soft, e la carta scoperta del dealer. Sebbene il gioco del Blackjack, nella sua formulazione completa, sia più propriamente descrivibile come un Processo Decisionale Parzialmente Osservabile (POMDP) a causa della dipendenza dall'ordine delle carte nel mazzo, abbiamo assunto una formulazione *puramente* markoviana rispetto allo stato informativo del giocatore, ignorando volontariamente informazioni non osservabili. Tale approssimazioni consente quindi di affrontare il problema come un MDP episodico con stati e azioni discrete, con ricompensa associata all'esito finale della mano. Abbiamo trovato riscontro con tale affermazione[4], ciò ci ha moralmente dato un via libera per il procedere con questo approccio.

8.4 Scelta del Reinforcement Learning

Una volta classificato il problema, abbiamo deciso di adottare un approccio di **Reinforcement Learning**, che viene naturale siccome trattiamo il tutto come un MDP. Difatto definiamo il Reinforcement Learning come un paradigma di apprendimento automatico in cui un agente impara a prendere decisioni sequenziali interagendo con un ambiente, al fine di massimizzare una misura cumulativa di ricompensa nel lungo periodo. Più precisamente, l'agente e l'ambiente interagiscono attraverso una sequenza di istanti temporali discreti $t = 0, 1, 2, 3, \ldots$. Per ogni istante t, l'agente riceve una rappresentazione dello stato dell'ambiente, $S_t \in \mathcal{S}$, dove $\mathcal{S}$ è l'insieme di tutti gli stati possibili. Sulla base di tale informazione, l'agente seleziona un'azione $A_t \in \mathcal{A}(S_t)$, dove $\mathcal{A}(S_t)$ rappresenta l'insieme delle azioni disponibili nello stato S_t.

All'istante successivo, come conseguenza dell'azione intrapresa, l'agente riceve una ricompensa numerica $R_{t+1} \in \mathcal{R} \subset \mathcal{R}$ e si ritrova in un nuovo stato S_{t+1}.



[5] Il problema adesso è quindi quello di far apprendere una policy che massimizzi il ritorno atteso nel lungo periodo.

8.5 On-policy vs Off-policy

Per comprendere le nostre scelte è necessario prima comprendere una differenza sostanziale tra due tipi di Reinforcement Learning: on-policy ed off-policy. Nel Reinforcement Learning, gli algoritmi possono essere classificati in base al rapporto tra la **policy utilizzata per interagire con l'ambiente**(*comportamentale*) e la **policy che viene appresa**(*target*). Un algoritmo è detto **off-policy** quando la policy comportamentale è diversa dalla policy target. Questo consente di apprendere una policy ottimale indipendentemente dalla strategia di esplorazione adottata. In questo caso quindi l'agente apprende il valore delle azioni assumendo che in futuro venga seguita una policy ottimale, indipendentemente dal comportamento esplorativo corrente. Questo consente di convergere più facilmente verso una strategia teoricamente ottimale, ma che potrebbe risultare più *aggressiva* e potrebbe non tener conto del rischio associato all'esplorazione durante la fase di apprendimento. Invece l'algoritmo è detto **on-policy** quando la policy utilizzata per selezionare le azioni durante l'interazione con l'ambiente coincide con la policy che viene aggiornata e migliorata nel processo di apprendimento. In questo caso, l'agente apprende il valore delle azioni effettivamente intraprese, includendo esplicitamente il comportamento esplorativo. Nel nostro caso la policy appresa riflette anche il rischio introdotto dalle azioni esplorative. Questo porta l'agente a sviluppare una strategia generalmente più *conservativa*, penalizzando azioni che, pur essendo ottimali in teoria, possono risultare rischiose in fase di esplorazione. Vale la pena osservare ambo gli approcci, per comprenderne al meglio i trade-off che si ottengono applicando uno rispetto all'altro nel caso del Blackjack.

Abbiamo quindi selezionato due algoritmi:

- **Q-learning:** che è un algoritmo di tipo off-policy.

- **Sarsa:** che è un algoritmo di tipo on-policy.

Siamo partii dal primo di questi.

9 Q-Learning

Il **Q-learning** è un algoritmo di Reinforcement Learning *off-policy*, introdotto da Christopher J. C. Watkins nel 1989 [6], con l'obiettivo di apprendere una policy ottimale per un Processo Decisionale di Markov senza richiedere la conoscenza delle dinamiche dell'ambiente.

9.1 Definizione formale

Il Q-learning si basa sull'apprendimento di una funzione di valore stato-azione $Q(s, a)$, che rappresenta il ritorno atteso cumulativo ottenibile eseguendo l'azione a nello stato s e seguendo successivamente la policy ottimale. Formalmente:

$$Q^*(s, a) = \max_{\pi} E\left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \,\middle|\, S_0 = s, A_0 = a, \pi\right].$$

Origine storica L'algoritmo è stato proposto da Watkins nel suo lavoro di dottorato come estensione dei metodi di programmazione dinamica a contesti in cui il modello dell'ambiente non è noto. La sua formulazione consente di aggiornare le stime di $Q(s, a)$ ad ogni iterazione utilizzando esclusivamente esperienza diretta, rendendolo particolarmente adatto a problemi stocastici ed episodici.

9.2 Funzionamento

Durante l'interazione con l'ambiente, l'agente osserva lo stato corrente s_t, seleziona un'azione a_t secondo una politica esplorativa , riceve una ricompensa r_{t+1} e avanza allo stato successivo s_{t+1}. La stima della funzione Q viene aggiornata secondo la regola:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)\right],$$

dove $\alpha \in (0, 1]$ è il tasso di apprendimento e $\gamma \in [0, 1]$ è il fattore di sconto.

Essendo un algoritmo **off-policy**, il Q-learning aggiorna le proprie stime assumendo che, a partire dallo stato successivo, venga seguita una policy ottimale, indipendentemente dall'azione effettivamente scelta durante l'esplorazione. Questa caratteristica consente all'algoritmo di convergere verso una policy ottimale anche in presenza di comportamenti esplorativi rischiosi.

10 SARSA

L'algoritmo **SARSA** (State-Action-Reward-State-Action) è un algoritmo di Reinforcement Learning *on-policy*, introdotto per apprendere una policy direttamente a partire dal comportamento adottato dall'agente durante l'interazione con l'ambiente.

10.1 Origine e contesto

SARSA fa la sua prima apparizione nel paper "On-Line Q-Learning Using Connectionist System"[7], sotto il nome di "Modified Connectionist Q-Learning". L'algoritmo ha due idee cardine dietro al suo funzionamento:

Temporal Difference Learning: l'agente impara da una differenza temporale tra due stime successive, non dal risultato finale completo, permettendo di aggiornare la stima del valore corrente durante il training.

Exploration-Exploitation tradeoff: trovare un buon compromesso tra *esplorazione* (quando l'agente compie azioni che lo portano in nuovi stati e situazioni mai sperimentate prima) ed *exploitation* (quando l'agente sfrutta le esperienze passate per ottenere prestazioni migliori nell'ambiente in cui si trova attualmente).

[8]

10.2 Definizione e differenza col Q-Learning

L'algoritmo SARSA apprende una funzione di valore azione-stato $Q(s, a)$, che rappresenta il ritorno atteso cumulativo ottenibile eseguendo l'azione a nello stato s e seguendo la stessa policy utilizzata per l'esplorazione. Il nome stesso dell'algoritmo deriva dalla sequenza di variabili utilizzate nell'aggiornamento. A differenza del Q-learning, non assume che l'agente segua una policy ottimale nello stato successivo, rendendolo più adatto a contesti in cui il rischio associato all'esplorazione deve essere tenuto in considerazione.

10.3 Funzionamento dell'algoritmo

Durante l'interazione con l'ambiente, l'agente osserva lo stato s_t, sceglie un'azione a_t, riceve una ricompensa r_{t+1} e transita nello stato successivo s_{t+1}, selezionando quindi una nuova azione a_{t+1} secondo la stessa policy. L'aggiornamento della funzione Q è dato da:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right].$$

Essendo un algoritmo **on-policy**, SARSA apprende una policy che riflette direttamente il comportamento esplorativo dell'agente, risultando in strategie più conservative rispetto a quelle apprese tramite Q-learning, ed è per questo che abbiamo deciso di usarlo.

11 Applicazione del Reinforcement Learning

Occorre infine definire come un'ultima distinzione, fondamentale per comprendere le nostre implementazioni, la differenza tra *online* ed *offline* reinforcement learning. **L'online reinforcement learning** prevede di simulare interazioni con l'ambiente per apprendere delle policy ottimali. Utilizza di volta in volta l'esperienza immediatamente appresa nelle successive interazioni con l'ambiente ed ha il vantaggio di potersi adattare ai cambiamenti date le continue interazioni con esso. Il grande problema di questo approccio è che a causa di tutte le simulazioni che genera con l'ambiente è computazionalmente costoso. Di fatto quindi questo tipo di approccio non necessità di un dataset per l'addestramento. A differenza del precedente **l'offline reinforcement learning** mira a creare delle policy efficaci su dati collezionati precedentemente, senza nessuna interazione con l'ambiente. Questo approccio porta con se anche delle sfide importanti: occorre migliorare le policy che si raccolgono dai dati così da stimare valori per azioni non presenti nel dataset.

11.1 Implementazione con Q-Learning

Occorre approfondire gli elementi che caratterizzano il Q-Learning per poterlo comprendere appieno ed implementarlo. Tra gli elementi più importanti c'è proprio la **Q-table**. Quest'ultima è una tabella che contiene i reward per ogni azione di ogni stato ed è possibile farne il lookup. In particolare ha una riga per ogni stato e una colonna per ogni azione. Inizialmente ogni valore di reward della tabella è inizializzato a zero ma ad ogni esplorazione il valore in questione cambia basandosi sull'equazione già precedentemente discussa.

Altri elementi fondamentali nel Q-learning sono i seguenti iperparametri caratterizzanti:

- α: rate di apprendimento che varia da 0 a 1. Occorre ottimizzare questo parametro per fare in modo che i valori di reward di ogni iterazione della Q-table siano conoscenza utile al progredire dell'agente.
- γ: termine che regola come comportarsi nel corso della ricerca della soluzione. Fa in modo che più vicino si arriva alla soluzione, maggiore sarà la preferenza verso dei reward a breve termine. Definisce quanto è lungimirante un agente ed anche questo varia tra 0 e 1.
- ϵ: termine che regola l'esplorazione causandone una diminuzione nel tempo di modo che la policy appresa migliori sfuttando le conoscenze acquisite nel tempo.

11.2 Primo approccio - Q-Learning Online

La prima implementazione adotta un approccio con Q-learning online, nel quale l'agente apprende interagendo direttamente con un ambiente simulato di Blackjack.

11.2.1 Architettura e Implementazione

L'architettura si compone di due componenti fondamentali: un ambiente di gioco implementato nella classe `BlackjackEnv` e un agente Q-learning descritto dalla classe `QLearningAgent`.

L'ambiente simula le dinamiche del gioco utilizzando sei mazzi di carte francesi standard, mescolati casualmente a ogni reset episodico. La gestione del mazzo include un meccanismo di ricreazione automatica quando le carte disponibili si esauriscono durante una mano, garantendo che l'agente possa sempre completare la propria sequenza decisionale senza interruzioni dovute a vincoli di disponibilità. Tale scelta implementativa, sebbene semplifichi la logica di esecuzione, introduce una sottile differenza rispetto alle regole convenzionali del Blackjack nei casinò reali, dove il rimescolamento avviene secondo una penetrazione predefinita del mazzo.

Lo stato dell'ambiente è rappresentato dalla tripla $(s_{\text{player}}, s_{\text{dealer}}, a_{\text{usable}})$, dove s_{player} indica la somma delle carte del giocatore, s_{dealer} rappresenta il valore della carta scoperta del dealer, e a_{usable} è un indicatore booleano della presenza di un asso utilizzabile come 11 senza causare sballamento. Questa rappresentazione dello stato è conforme alla formulazione classica del problema del Blackjack come Markov Decision Process episodico, nel quale lo stato cattura tutta l'informazione rilevante per la decisione ottimale del giocatore.

L'agente implementa l'algoritmo Q-learning con i seguenti iperparametri: $\varepsilon = 0.1$ per la politica ε-greedy, $\alpha = 0.1$ come tasso di apprendimento, e $\gamma = 0.9$ quale fattore di sconto. La funzione di valore azione-stato $Q(s, a)$ è rappresentata mediante una tabella implementata come dizionario Python con inizializzazione ottimistica a zero, che associa a ogni coppia stato-azione un vettore di due valori corrispondenti alle azioni disponibili: *stand* (codificata come 0) e *hit* (codificata come 1). Tale rappresentazione tabulare è appropriata per il Blackjack, dato il numero limitato di stati osservabili.

11.2.2 Processo di Apprendimento

Il processo di training si articola su un totale di un milione di episodi. Ad ogni episodio, l'ambiente viene reinizializzato mediante chiamata al metodo `reset()`, che ricrea il mazzo di carte e distribuisce la mano iniziale. L'agente seleziona le azioni secondo una strategia ε-greedy durante la fase di training: con probabilità ε sceglie un'azione casuale per favorire l'esplorazione dello spazio degli stati, mentre con probabilità $1 - \varepsilon$ seleziona l'azione con valore Q massimo per lo stato corrente, sfruttando la conoscenza acquisita.

L'aggiornamento della funzione Q segue la regola standard del Q-learning off-policy:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right] \qquad (1)$$

dove s_t è lo stato al tempo t, a_t è l'azione intrapresa, r_{t+1} è la ricompensa ricevuta, e s_{t+1} rappresenta lo stato successivo. L'operatore max calcola il valore massimo tra le azioni disponibili nello stato futuro, rendendo l'algoritmo off-policy: l'agente aggiorna le stime assumendo che in futuro seguirà la politica ottimale, indipendentemente dall'azione effettivamente scelta durante l'esplorazione. Questa caratteristica permette all'agente di apprendere una politica più aggressiva e potenzialmente ottimale, anche quando il comportamento esplorativo introduce azioni subottimali.

La struttura delle ricompense è definita come segue: $+1$ per vittoria, 0 per pareggio, -1 per sconfitta. Questo schema di ricompense sparse, concentrate esclusivamente alla fine dell'episodio, rende il problema particolarmente adatto al Q-learning episodico, nel quale l'agente impara a valutare le conseguenze a lungo termine delle proprie azioni iniziali attraverso il meccanismo di propagazione del valore tramite il fattore di sconto γ.

11.2.3 Risultati Sperimentali

Al termine del training su un milione di episodi, la Q-table risultante contiene valori appresi per 280 coppie stato-azione distinte incontrate durante l'interazione con l'ambiente. I risultati finali evidenziano un *win rate* pari a 0.4029 (40.29%), un *loss rate* di 0.5104 (51.04%) e un *draw rate* di 0.0867 (8.67%).

L'analisi della curva di apprendimento, ottenuta mediante media mobile su finestre di 10.000 episodi, mostra una convergenza progressiva della ricompensa media verso valori asintoticamente stabili. Il *win rate* cumulativo si stabilizza intorno a valori che si avvicinano alla soglia di riferimento della strategia di base ottimale del Blackjack, documentata nella letteratura [9]essere approssimativamente 0.42 nelle condizioni regolamentari standard. Questo comportamento conferma che l'agente è in grado di migliorare significativamente le proprie prestazioni rispetto a una strategia puramente casuale, apprendendo una politica che riflette principi decisionali fondamentali del Blackjack, quali la necessità di richiedere carta quando la somma è bassa e di mantenersi quando la somma è prossima a 21.

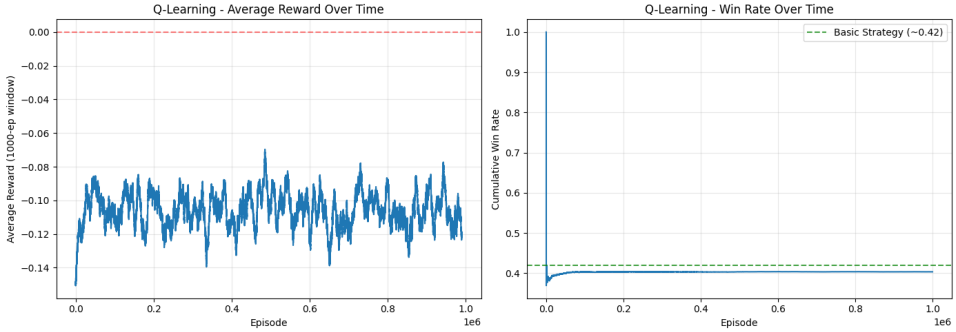


Figure 1: Risultati Implementazione– Online Q-Learning

11.2.4 Vantaggi dell'Approccio Online

L'approccio di Q-learning online presenta diversi vantaggi intrinseci nel contesto del Blackjack. In primo luogo, l'agente apprende in modo completamente autonomo senza necessità di dati pre-raccolti, adattandosi dinamicamente alle caratteristiche dell'ambiente attraverso l'interazione diretta. Questa modalità di apprendimento risulta particolarmente efficace in contesti dove la raccolta di dati offline sarebbe costosa o impraticabile, permettendo all'agente di esplorare liberamente lo spazio delle politiche possibili. In secondo luogo, la natura episodica del Q-learning online consente all'agente di **bilanciare autonomamente esplorazione e sfruttamento** attraverso il parametro ε, garantendo che vengano visitate anche regioni dello spazio degli stati meno frequenti ma potenzialmente rilevanti per l'apprendimento di una politica ottimale. Il meccanismo di ricreazione del mazzo quando le carte si esauriscono, sebbene costituisca una semplificazione rispetto alle regole reali, assicura che l'agente non incontri mai situazioni di stallo e possa completare ogni episodio con una transizione terminale ben definita.

Inoltre, l'approccio online permette un **apprendimento incrementale e continuo**: l'agente migliora progressivamente la propria politica ad ogni episodio, con aggiornamenti immediati della Q-table basati sull'esperienza più recente. Questa caratteristica risulta vantaggiosa in scenari dove l'ambiente può presentare variazioni nel tempo o dove è necessario adattarsi rapidamente a nuove condizioni di gioco.

11.2.5 Limitazioni dell'Approccio Online

Nonostante i vantaggi evidenziati, l'approccio online presenta alcune limitazioni significative. La principale criticità riguarda l'efficienza campionaria: per raggiungere una convergenza soddisfacente, l'agente necessita di un numero elevato di interazioni con l'ambiente, come dimostrato dai un milione di episodi richiesti per stabilizzare il *win rate*. Questo comporta un costo computazionale non trascurabile, sebbene nel caso specifico del Blackjack simulato tale costo risulti ancora gestibile grazie alla semplicità delle operazioni di simulazione.
Un'altra limitazione riguarda la stocasticità intrinseca del processo di apprendimento: esecuzioni diverse dello stesso algoritmo possono produrre politiche con prestazioni leggermente differenti a causa della casualità nell'inizializzazione, nell'esplorazione e nell'ordine di visita degli stati. Questa variabilità, sebbene contenuta grazie all'elevato numero di episodi, introduce un elemento di incertezza nella valutazione delle prestazioni finali.
Infine, la dipendenza dalla qualità dell'esplorazione rappresenta un aspetto critico: una strategia ε-greedy con un valore di ε troppo basso potrebbe impedire all'agente di scoprire regioni promettenti dello spazio degli stati, mentre un valore troppo alto rallenterebbe la convergenza. Nel contesto implementato, il valore $\varepsilon = 0.1$ rappresenta un compromesso ragionevole, ma la scelta ottimale di questo iperparametro richiederebbe un processo di tuning più approfondito.

11.3 Secondo approccio – Q-Learning Offline

Il secondo approccio adotta Q-learning offline, nel quale l'agente apprende esclusivamente da un dataset di esperienze pre-raccolte, **senza interazione diretta** con l'ambiente durante la fase di training.

11.3.1 Architettura e Motivazione

Questa metodologia si inserisce nel paradigma più ampio dell'offline reinforcement learning, noto anche come batch reinforcement learning, che ha ricevuto crescente attenzione nella comunità scientifica per la sua applicabilità a domini in cui l'interazione online con l'ambiente risulta costosa, rischiosa o impraticabile [10].
Nel contesto del presente progetto, l'approccio offline è stato introdotto con

l'obiettivo di investigare se un agente possa apprendere una politica competitiva a partire da un dataset statico di transizioni, riducendo potenzialmente i costi computazionali associati alla simulazione ripetuta di un milione di episodi. Inoltre, l'apprendimento offline permette di studiare l'impatto della qualità e della distribuzione dei dati sulla convergenza dell'algoritmo e sulle prestazioni finali della politica appresa.
Recenti lavori in letteratura hanno evidenziato come l'ottimizzazione diretta di obiettivi di reinforcement learning, anche in contesti offline, possa mitigare parzialmente i limiti degli approcci puramente supervisionati, rafforzando l'importanza dell'allineamento tra funzione obiettivo e massimizzazione della ricompensa [11].

11.3.2 Generazione e Preprocessing del Dataset

Il dataset utilizzato per l'addestramento offline è stato generato mediante una versione modificata del simulatore di Blackjack precedentemente usato. Tale simulatore implementa le regole standard del Blackjack con otto mazzi di carte, penetrazione di 6.5 mazzi, e seguendo la strategia di base ottimale per la selezione delle azioni del giocatore.
La modifica fondamentale apportata al codice di generazione riguarda la strategia decisionale adottata dal giocatore simulato. Mentre il codice originale implementava la strategia di base completa del Blackjack, includendo azioni complesse quali *double*, *split* e *surrender*, la versione modificata è stata semplificata per generare esclusivamente transizioni relative alle azioni *hit* e *stand*. Questa scelta è motivata da due considerazioni principali.
In primo luogo, la limitazione dello spazio delle azioni a due sole opzioni rende il problema di apprendimento più trattabile e facilita il confronto diretto con l'approccio online, che considera anch'essa solo le azioni *hit* e *stand*.
La strategia implementata nel generatore modificato è una politica semplificata: il giocatore richiede carta (*hit*) quando la somma della propria mano è strettamente inferiore a 17, e si mantiene (*stand*) altrimenti. Questa regola, sebbene subottimale rispetto alla strategia di base, produce un dataset di transizioni sufficientemente variegate per l'apprendimento offline.
In secondo luogo, la semplificazione della strategia di generazione dei dati riflette una situazione realistica nell'offline reinforcement learning, dove il dataset disponibile è spesso raccolto mediante una politica comportamentale che non coincide necessariamente con la politica ottimale. L'algoritmo di Q-learning offline deve quindi essere in grado di estrarre una politica migliorata a partire da dati generati da una strategia imperfetta, dimostrando capacità di generalizzazione e robustezza rispetto alla distribuzione dei dati.
Il dataset finale contiene un milione di transizioni, ciascuna rappresentata dalla tupla (s_t, a_t, r_t, s_{t+1}), dove $s_t = (s_{\text{player}}, s_{\text{dealer}}, a_{\text{usable}})$ è lo stato al tempo t, $a_t \in \{0, 1\}$ è l'azione intrapresa, $r_t \in \{-1, 0, 1\}$ è la ricompensa osservata, e s_{t+1} è lo stato successivo.

Le ricompense sono state discretizzate in modo tale che le vittorie corrispondano a $r = 1$, le sconfitte a $r = -1$ e i pareggi a $r = 0$. Questa struttura rende il dataset direttamente utilizzabile per l'addestramento mediante Q-learning offline.
È importante sottolineare che il codice modificato per la generazione del dataset non è incluso nel repository GitHub del progetto, in quanto costituisce uno strumento ausiliario utilizzato esclusivamente per la fase di raccolta dati. La documentazione tecnica fornisce tuttavia una descrizione completa delle modifiche apportate, permettendo la riproducibilità del processo di generazione del dataset.

11.3.3 Algoritmo di Apprendimento Offline

L'algoritmo di Q-learning offline implementato nel secondo approccio differisce dall'approccio online principalmente per l'assenza di interazione con l'ambiente durante il training. L'agente, rappresentato dalla classe `Offline QLearningAgent` , è inizializzato con una Q-table vuota e iperparametri identici a quelli utilizzati nell'approccio online: $\alpha = 0.1$ e $\gamma = 0.9$.
Il processo di addestramento consiste in un singolo passaggio sequenziale attraverso il dataset di transizioni. Per ogni tupla (s_t, a_t, r_t, s_{t+1}), l'agente aggiorna la propria stima della funzione Q secondo la regola:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right] \tag{2}$$

Questa regola di aggiornamento è formalmente identica a quella utilizzata nel Q-learning online, ma opera su dati statici piuttosto che su esperienze raccolte dinamicamente. L'assenza di esplorazione attiva implica che l'agente apprende esclusivamente dagli stati e dalle azioni presenti nel dataset, senza possibilità di visitare regioni dello spazio degli stati non rappresentate nei dati.

11.3.4 Valutazione e Risultati Sperimentali

Per valutare le prestazioni della politica appresa offline, è stato implementato un ambiente di test identico a quello utilizzato nell'approccio online. L'agente viene valutato su un milione di episodi, seguendo una politica greedy derivata dalla Q-table appresa, ovvero selezionando in ogni stato l'azione con valore Q massimo.
I risultati sperimentali mostrano un *win rate* pari a 0.3814 (38.14%), un *loss rate* di 0.5666 (56.66%) e un *draw rate* di 0.0520 (5.20%). Rispetto alla versione online, si osserva una riduzione del *win rate* di circa 2.15 punti percentuali, accompagnata da un aumento del *loss rate*. Questa differenza, sebbene contenuta, indica che l'apprendimento offline produce una politica leggermente meno performante rispetto all'approccio online.

L'analisi delle cause di tale discrepanza evidenzia due fattori principali. In primo luogo, il dataset offline è stato generato mediante una politica comportamentale subottimale, introducendo un bias nella distribuzione delle transizioni rispetto a quelle che sarebbero state osservate seguendo una politica ottimale. Questo fenomeno, noto come *distributional shift* [12], rappresenta una delle principali sfide nell'offline reinforcement learning.
In secondo luogo, l'assenza di esplorazione attiva durante l'addestramento offline impedisce all'agente di correggere errori di stima della funzione Q in regioni dello spazio degli stati scarsamente rappresentate nel dataset. Nel Q-learning online, la strategia ε-greedy consente invece di rivisitare periodicamente anche stati rari, migliorando progressivamente la qualità delle stime.

11.3.5 Vantaggi dell'Approccio Offline

Nonostante le prestazioni leggermente inferiori, l'approccio offline presenta vantaggi significativi in termini di efficienza computazionale, riproducibilità e riutilizzo dei dati.
Tali osservazioni sono coerenti con risultati recenti che mostrano come l'ottimizzazione diretta di obiettivi di reinforcement learning in contesti offline possa comunque produrre politiche competitive, a condizione di disporre di dataset sufficientemente informativi [11].

11.3.6 Limitazioni dell'Approccio Offline

Le limitazioni principali dell'approccio offline sono strettamente connesse alla qualità e alla copertura del dataset. Il bias introdotto dalla politica comportamentale e l'assenza di esplorazione rendono l'algoritmo vulnerabile a errori di estrapolazione, fenomeno noto come *off-policy evaluation problem*.
La letteratura recente propone estensioni del Q-learning offline, come *Conservative Q-Learning* e *Implicit Q-Learning*, che introducono meccanismi di penalizzazione per azioni non sufficientemente supportate dal dataset, riducendo l'over-ottimismo nelle stime della funzione Q [11]. L'integrazione di tali tecniche è lasciata come possibile estensione futura del presente lavoro.

11.4 Confronto tra le implementazioni e Scelta Finale

11.4.1 Analisi Comparativa delle Prestazioni

Il confronto sistematico tra le due implementazioni rivela differenze significative in termini di prestazioni, efficienza computazionale e robustezza. Dal punto di vista delle prestazioni pure, l'implementazione online si dimostra superiore con un *win rate* di 40.29% contro il 38.14% dell'approccio offline, corrispondente a un vantaggio di 2.15 punti percentuali.

Sebbene questa differenza possa apparire modesta in termini assoluti, nel contesto del Blackjack rappresenta un divario non trascurabile.
L'analisi della distribuzione degli esiti rivela ulteriori differenze qualitative. L'implementazione online presenta un *draw rate* di 8.67%, significativamente superiore al 5.20% dell'approccio offline. Questa discrepanza suggerisce che la politica appresa online adotta un comportamento più conservativo in situazioni ambigue, optando per lo *stand* quando la probabilità di vittoria è comparabile a quella di pareggio. Al contrario, la politica offline, influenzata dalla strategia del dataset che richiede carta fino a 17, tende ad assumere maggiori rischi, risultando in un numero inferiore di pareggi ma anche in un *loss rate* più elevato (56.66% contro 51.04%).
Dal punto di vista della stabilità e convergenza, l'implementazione online beneficia del meccanismo di esplorazione ε-greedy, che garantisce la visita continua di diverse regioni dello spazio degli stati anche in fasi avanzate , garantendo una convergenza più robusta e uniforme della Q-table, riducendo il rischio di sovrastime o sottostime locali. L'approccio offline, essendo vincolato alla distribuzione statistica del dataset non può correggere errori di stima in regioni scarsamente rappresentate, potendo quindi risultare instabile nel predire stati visitati raramente durante la generazione dei dati.

11.4.2 Efficienza Computazionale e Scalabilità

L'efficienza computazionale rappresenta uno dei principali vantaggi dell'approccio offline. In assenza di interazione diretta con l'ambiente durante la fase di addestramento, l'algoritmo opera esclusivamente su un dataset statico di transizioni, evitando il costo computazionale associato alla simulazione dell'ambiente, alla gestione della dinamica di gioco e al calcolo iterativo delle ricompense. Di conseguenza, il training offline risulta generalmente più rapido rispetto all'approccio online, consentendo un utilizzo più efficiente delle risorse di calcolo. Nel contesto specifico del presente progetto, il beneficio computazionale complessivo risulta tuttavia limitato. Il dataset offline è stato infatti generato tramite simulazione, e il costo associato alla fase di raccolta dei dati è comparabile a quello richiesto dall'addestramento online. Pertanto, se si considera l'intero ciclo che comprende sia la generazione delle esperienze sia la fase di apprendimento, il vantaggio dell'approccio offline risulta marginale nel caso in esame.
La situazione cambia in modo sostanziale quando il dataset è già disponibile o può essere riutilizzato per più cicli di sviluppo e sperimentazione. In questi scenari, il costo di raccolta dei dati viene ammortizzato su molteplici sessioni di addestramento, rendendo l'approccio offline particolarmente vantaggioso. Questo aspetto assume un ruolo cruciale in domini in cui la simulazione o l'interazione con l'ambiente è costosa, limitata o rischiosa, e dove la possibilità di riutilizzare dati storici consente di esplorare diverse configurazioni algoritmiche in modo efficiente e scalabile.

11.4.3 Trade-off Fondamentali

I risultati sperimentali evidenziano un trade-off fondamentale tra prestazioni e efficienza. L'implementazione online raggiunge prestazioni superiori al costo di tempi di training più lunghi e di una maggiore variabilità stocastica tra esecuzioni diverse. L'approccio offline offre training rapidi e risultati deterministici, ma soffre di prestazioni ridotte a causa del bias introdotto dalla politica di generazione del dataset.
Un secondo trade-off riguarda la flessibilità rispetto alla dipendenza dai dati. L'approccio online è completamente autonomo e non richiede dati preraccolti, adattandosi dinamicamente all'ambiente attraverso l'esplorazione. Tuttavia, questa flessibilità ha un costo in termini di sample complexity: l'agente deve interagire ripetutamente con l'ambiente per accumulare esperienza sufficiente. L'approccio offline, al contrario, dipende criticamente dalla qualità del dataset disponibile, ma può estrarre politiche competitive da un numero relativamente limitato di transizioni se i dati sono ben distribuiti e rappresentativi.
Un terzo trade-off concerne la robustezza rispetto a errori di stima.
L'approccio online beneficia del meccanismo di correzione continua fornito dall'esplorazione, che permette all'agente di rivedere le proprie stime anche in stati visitati raramente. L'implementazione offline, invece, può accumulare errori di estrapolazione in regioni dello spazio degli stati scarsamente rappresentate nel dataset, senza possibilità di correzione attraverso nuove esperienze.

11.4.4 Scelta Finale e Motivazioni

Alla luce dell'analisi comparativa condotta, la scelta finale ricade sul Q-learning online. Questa decisione è motivata da tre considerazioni principali che riflettono i requisiti specifici del progetto e gli obiettivi di apprendimento prefissati.
In primo luogo, le prestazioni superiori della l'approccio online rappresentano il criterio decisionale primario. Nel contesto del Blackjack, dove l'obiettivo è massimizzare il *win rate* a lungo termine, il vantaggio di 2.15 punti percentuali dell'approccio online si traduce in un beneficio economico concreto nel caso di applicazione del sistema in scenari realistici. Sebbene la differenza possa sembrare modesta, nel lungo periodo di migliaia o milioni di mani giocate, questo divario si amplifica significativamente.
In secondo luogo, l'autonomia e la flessibilità dell'approccio online si allineano meglio con gli obiettivi didattici del progetto nel contesto del corso di Fondamenti di Intelligenza Artificiale. L'implementazione del Q-learning online permette di comprendere in modo più completo i meccanismi di esplorazione, convergenza e bilanciamento tra exploitation ed exploration che costituiscono i pilastri fondamentali del reinforcement learning.

L'approccio offline, sebbene interessante da un punto di vista di ricerca avanzata, introduce complessità aggiuntive legate alla qualità dei dati e al distributional shift che esulano dagli obiettivi formativi basilari del corso.
In terzo luogo, la maggiore robustezza e affidabilità della versione online la rendono più adatta per un sistema che potrebbe essere esteso o modificato in futuri sviluppi. La capacità dell'agente online di adattarsi dinamicamente a variazioni delle regole di gioco o a diversi livelli di difficoltà rappresenta un vantaggio strategico rispetto all'approccio offline, che richiederebbe la rigenerazione del dataset ad ogni modifica dell'ambiente.
È tuttavia importante sottolineare che l'approccio offline ha dimostrato potenzialità significative e rappresenta una direzione promettente per futuri sviluppi del progetto. In particolare, l'implementazione di tecniche avanzate di offline reinforcement learning quali Conservative Q-Learning o Behavior Cloning potrebbe permettere di superare le limitazioni osservate e raggiungere prestazioni comparabili o superiori a quelle dell'approccio online. Inoltre, l'utilizzo di dataset generati mediante strategie di esplorazione più sofisticate, quali la strategia di base ottimale del Blackjack, potrebbe ridurre significativamente il gap prestazionale osservato.
In conclusione, la scelta dell'approccio online riflette una valutazione ponderata dei trade-off tra prestazioni, efficienza, robustezza e allineamento con gli obiettivi formativi del progetto. Questa decisione non sminuisce il valore dell'approccio offline, che ha fornito insights importanti sulla dipendenza delle prestazioni dalla qualità dei dati e ha permesso di esplorare una metodologia complementare di apprendimento per rinforzo. L'esperienza acquisita attraverso l'implementazione di entrambe le implementazioni costituisce un contributo significativo alla comprensione dei principi fondamentali del reinforcement learning e delle sue applicazioni a problemi decisionali sequenziali come il Blackjack.

11.5 Terzo approccio - SARSA

Dopo aver ottenuto risultati tutto sommato significativi con un approccio off-policy, è arrivato il momento di provare SARSA, algoritmo simbolo della metodologia on-policy. Dall'esperienza maturata con il Q-Learning, abbiamo deciso di implementare solo l'approccio *online.*

11.5.1 Ambiente di simulazione

L'ambiente di simulazione implementa una variante del gioco del Blackjack modellata come un processo decisionale di Markov (come precedentemente giustificato). Lo stato dell'ambiente è definito dalla mano del giocatore, dalla mano del dealer e dalla carta scoperta del dealer. Il gioco utilizza un mazzo composto da otto deck standard. Il mazzo viene rimescolato automaticamente quando il numero di carte residue scende sotto una soglia prefissata,

al fine di garantire stabilità statistica e ridurre possibili bias legati al conteggio delle carte. Il valore delle mani viene calcolato gestendo dinamicamente la presenza degli assi, distinguendo tra mani *soft* e *hard*. Il dealer segue le regole standard del Blackjack, pescando fino a raggiungere almeno 17 e colpendo sul *soft 17*. Analogamente ai precedenti approcci si è scelto di permettere all'agente di scegliere solo tra due azioni, ovvero *Hit* e *Stand*. La funzione di ricompensa è stata calibrata empiricamente, partendo da una scelta iniziale di far valere maggiormente vittorie (+5) e non andare a punire altrettanto duramente le sconfitte (-3), si è passato poi ad un approccio tipicamente neutro (+1 vittoria, -1 sconfitta, 0 pareggio) poichè i risultati ottenuti erano migliori.

11.5.2 Rappresentazione dello stato

Per l'apprendimento, lo stato viene rappresentato in forma compatta come:

$$s = (\text{player\_sum}, \text{is\_soft}, \text{dealer\_showing})$$

dove player_sum indica la somma della mano del giocatore, is_soft segnala la presenza di un asso utilizzabile come valore 11, e dealer_showing rappresenta la carta scoperta del dealer. Questa rappresentazione consente di mantenere lo spazio degli stati contenuto e discreto.

11.5.3 Agente SARSA

L'agente viene inizializzato con i seguenti iperparametri:

- α (*learning rate*), che controlla la velocità di aggiornamento dei valori Q, $\alpha = 0.01$;
- γ (*discount factor*), che determina l'importanza delle ricompense future, il valore scelto è 0.95;
- ϵ, parametro iniziale della strategia di esplorazione ϵ-greedy, il valore scelto è 1.0;
- ϵ_{decay}, fattore di decadimento esponenziale di ϵ, valore scelto 0.9999;
- ϵ_{min}, valore minimo consentito per l'esplorazione, valore scelto 0.01.

La funzione di valore d'azione $Q(s, a)$ è memorizzata in una Q-table implementata come dizionario annidato,inizializzata implicitamente a zero per tutte le coppie stato–azione non ancora visitate. Vengono mantenuti inoltre dati sugli esiti degli episodi, includendo ricompense, vittorie, sconfitte e pareggi. La politica di esplorazione è basata su una strategia epsilon-greedy: con probabilità ϵ viene scelta un'azione casuale, mentre con probabilità $1 - \epsilon$ viene selezionata l'azione con valore Q massimo.

Il parametro ϵ decresce progressivamente seguendo l'epsilon_decay nel corso dell'addestramento, favorendo inizialmente l'esplorazione e successivamente lo sfruttamento delle conoscenze acquisite. Nei casi in cui uno stato non sia presente nella Q-table o non possieda azioni precedentemente valutate, l'agente seleziona un'azione casuale, garantendo robustezza anche nelle fasi iniziali dell'apprendimento. È inoltre introdotta una regola deterministica che forza l'azione *STAND* quando la somma della mano del giocatore è maggiore o uguale a 21, evitando transizioni non valide e riducendo il rumore nell'aggiornamento dei valori Q.

11.5.4 Aggiornamento SARSA

Per ogni transizione osservata $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$, il valore viene aggiornato secondo la regola:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right]$$

Questa formulazione differisce dal Q-learning classico in quanto utilizza l'azione effettivamente selezionata dalla politica corrente nello stato successivo, anziché l'azione greedy. Ciò rende l'algoritmo sensibile alla strategia di esplorazione adottata e garantisce coerenza tra policy di comportamento e policy di valutazione. Nel caso di stati terminali, l'aggiornamento viene effettuato utilizzando esclusivamente la ricompensa osservata:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t - Q(s_t, a_t) \right]$$

Al termine di ogni episodio, il parametro ϵ viene aggiornato secondo una legge di decadimento esponenziale:

$$\epsilon \leftarrow \max(\epsilon_{\min}, \epsilon \cdot \epsilon_{\text{decay}})$$

Questo meccanismo consente all'agente di esplorare ampiamente lo spazio stato–azione nelle fasi iniziali, per poi convergere progressivamente verso una politica quasi deterministica.

11.5.5 Procedura di Addestramento

L'addestramento avviene su un numero prefissato di episodi. Tale numero è stato posto a 500000, poichè tramite varie prove empiriche abbiamo verificato che non ci fosse nessun sostanziale aumento delle prestazioni andando oltre tale soglia.

Per ciascun episodio, l'agente:

1. inizializza l'ambiente e seleziona un'azione iniziale;
2. interagisce con l'ambiente fino al raggiungimento di uno stato terminale;
3. aggiorna la Q-table ad ogni transizione secondo la regola SARSA;
4. registra la ricompensa totale dell'episodio;
5. aggiorna il parametro di esplorazione ϵ.

Il numero massimo di passi per episodio è limitato, prevenendo cicli infiniti e garantendo stabilità computazionale.

11.5.6 Valutazione della Policy

La fase di valutazione viene condotta disabilitando l'esplorazione ($\epsilon = 0$) su:

$$N_{\text{eval}} = 500{,}000$$

episodi indipendenti. L'agente segue una politica puramente greedy, consentendo di stimare le prestazioni effettive della policy appresa. Le metriche di valutazione includono:

- win rate;
- loss rate;
- draw rate;
- ricompensa media per episodio.

Questa separazione netta tra addestramento ed evaluation consente di misurare in modo affidabile la qualità della policy risultante.

11.6 Analisi della policy

La policy finale viene visualizzata tramite heatmap binarie separate per mani *hard* e *soft*. Per ogni combinazione di somma del giocatore e carta del dealer, viene indicata l'azione ottimale appresa. Le mappe risultanti mostrano una forte corrispondenza con la *basic strategy* del Blackjack, con addirittura momenti in cui la percentuale di vittorie risulta leggermente maggiore.

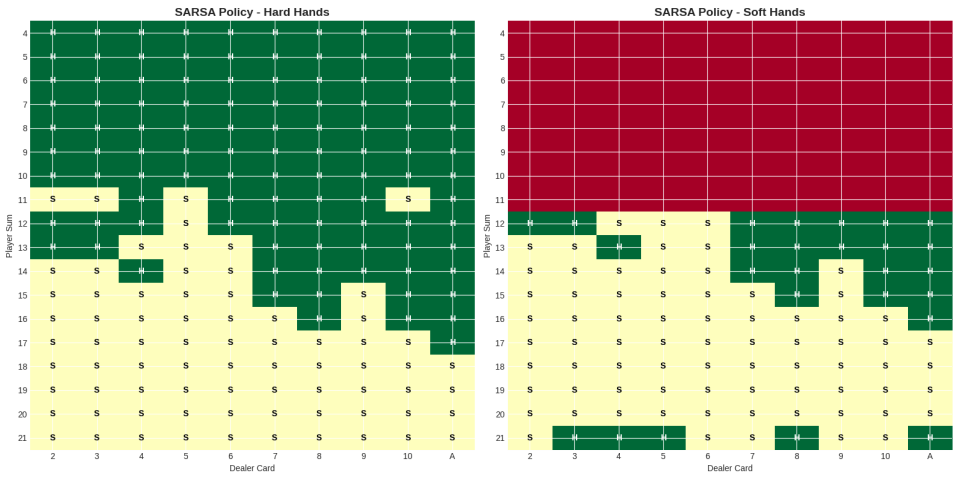


Figure 2: SARSA Policy – Online SARSA

11.7 Risultati sperimentali e confronto con Q-Learning online

L'implementazione con **SARSA online** si è dimostrata ancora più efficace dell'implementazione con il Q-Learning online. Al termine del training su cinquecentomila episodi, la Q-table risultante contiene valori appresi per 280 azioni coppie-stato. I risultati finali hanno un *win rate* del 42.67%, contro il 40.29% ottenuto con il Q-Learning online ed un miglioramento di più di 4 punti percentuale rispetto al Q-Learning offline. Si può osservare un leggero aumento del *draw rate* che raggiunge il 9.43% rispetto all '8.67% dell'approccio con Q-Learning online. Infine si ha una significativa riduzione delle sconfitte che raggiungono il 47.90%, il risultato più basso ottenuto tra tutti gli approcci analizzati. Nonostante il grande traguardo del limite di vittorie teorico raggiunto, anche questa implementazione ha gli stessi compromessi analizzati durante l'implementazione dell'online Q-Learning. Nelle seguenti figure è possibile osservare il progredire del modello fino a raggiungere risultati vicini alla *basic strategy* inizialmente illustrata.

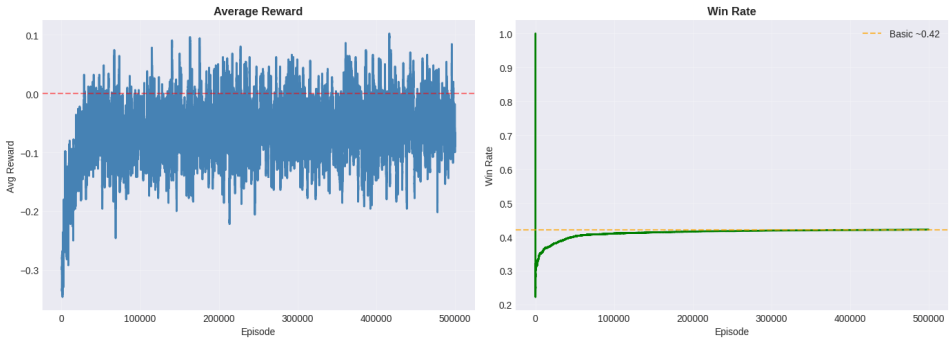


Figure 3: Risultati Implementazione – Online SARSA

11.8 Confronto finale tra le implementazioni di Reinforcement Learning

Qui è possibile osservare una tabella di riepilogo che contiene tutti i risultati ottenuti con gli approcci che prevedevano l'utilizzo del Reinforcement Learning.

Table 1: Confronto delle prestazioni degli agenti RL su Blackjack

Algoritmo	**Win Rate (%)**	**Loss Rate (%)**	**Draw Rate (%)**
Q-Learning (Online)	40.29%	51.04%	8.67%
Q-Learning (Offline)	38.14%	55.66%	5.20%
SARSA (Online)	42.67%	47.90%	9.43%

12 Demo implementate

Dopo tutte le analisi e le considerazioni effettuate, la decisione di implementare un prototipo del nostro agente non poteva che ricadere sui due modelli più performanti che sono riusciti a raggiungere (o quasi) la base strategy per massimizzare il **win rate**. Abbiamo quindi implementato una demo con il Q-Learning online e il SARSA online. Le demo sono di fatto uguali con l'unica differenza è che quella del Q-Learning gioca automaticamente mani di blackjack mentre l'implementazione con SARSA prevede una demo interattiva. La demo è stata implementata con la libreria python `tkinker` che permette di realizzare interfacce grafiche. La demo permette quindi di visualizzare graficamente il banco e le carte in gioco sulla parte sinistra, mentre permette di visualizzare informazioni utili e le predizioni del modello nella parte a destra. La demo interattiva con SARSA include tre tasti: il primo permette di giocare una partita, il secondo di fare hit ed il terzo permette di fare stand.

A destra sarà possibile visualizzare la scelta del modello consigliata per prendere le decisioni secondo la policy appresa dal modello.
Con riferimento alla demo che gioca automaticamente con Q-Learning le uniche differenze sono i tasti disattivati, non è quindi possibile interagire con il gioco in esecuzione.

13 Considerazioni finali

Alla luce di tutto il lavoro fatto, tocca fare delle considerazioni. Abbiamo raggiunto ottimi risultati, senza dare direttamente in pasto all'agente la strategia di base, riuscendo addirittura ad equipararla. Tenendo questo a mente, e considerando gli obiettivi che ci eravamo preposti inizialmente non possiamo che essere soddisfatti. Questo progetto ci ha permesso di approfondire le nostre conoscenze sull'apprendimento supervisionato ed acquisire nuove competeneze sul Reinforcement Learning (e basi teoriche sugli studi di Markov). Nonostante gli ottimi risultati ottenuti, siamo fiduciosi che tramite un ulteriore apprendimento delle tecniche e degli algoritmi applicabili, sia possibile andare ad ottenere risultati ancora migliori.

References

[1] Roberto De Prisco. Appunti per i corsi di musimatica e algorithmic music and sound computing. *UNISA*, page 86, 2024. `http://intranet.di.unisa.it/~robdep/Musimatica/dispensa/dispensa2024.pdf`.

[2] J. R. Norris. Markov chains. *Cambridge Series on Statistical and Probabilistic Mathematics*, page 15, 1997. `https://cape.fcfm.buap.mx/jdzf/cursos/procesos/libros/norris.pdf`.

[3] Davide Bacciu. Markov decision processes. *Università di Pisa*, page 6. `https://elearning.di.unipi.it/pluginfile.php/31332/course/section/3378/lect3_MDP.pdf?time=1588178153474`.

[4] Lucas Bordeu and Javier Castro. Optimal blackjack betting strategies through dynamic programming and expected utility theory, 2025. `https://arxiv.org/abs/2505.00724`.

[5] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An introduction*. The MIT Press, 2015. `https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf`.

[6] Christopher J.C.H. Watkins. Learning from delayed rewards. 1989. `https://link.springer.com/article/10.1007/BF00992698#Bib1`.

[7] G. Rummery and Mahesan Niranjan. On-line q-learning using connectionist systems. *Technical Report CUED/F-INFENG/TR 166*, 11 1994. https://www.researchgate.net/profile/Mahesan-Niranjan/publication/2500611_On-Line_Q-Learning_Using_Connectionist_Systems/links/5438d5db0cf204cab1d6db0f/On-Line-Q-Learning-Using-Connectionist-Systems.pdf.

[8] Khayyon Parker. Sarsa - an algorithm not named by its inventor. *Medium*, 2022. https://medium.com/codex/sarsa-an-algorithm-not-named-by-its-inventor-be768b43d771.

[9] Roger R. Baldwin. The optimum strategy in blackjack. *Journal of the American Statistical Association*, 1956. https://web.williams.edu/Mathematics/sjmiller/public_html/341Fa09/handouts/Baldwin_OptimalStrategyBlackjack.pdf.

[10] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2110.06169*, 2021. https://arxiv.org/abs/2110.06169%7D%7D.

[11] Denis Tarasov, Alexander Nikulin, Ilya Zisman, Albina Klepach, Andrei Polubarov, Nikita Lyubaykin, Alexander Derevyagin, Igor Kiselev, and Vladislav Kurenkov. Yes, q-learning helps offline in-context reinforcement learning. *arXiv preprint arXiv:2502.17666*, 2025. https://arxiv.org/abs/2502.17666%7D%7D.

[12] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. In *Advances in Neural Information Processing Systems*, 2020. https://proceedings.neurips.cc/paper/2020/hash/0d2b2061826a5df3221116a5085a6052-Abstract.html%7D%7D.