Report Document

EduProfiler

Riferimento	edu-profiler_report_battaglia_pennarella
Versione	5
Data	11/01/2025
Destinatario	Studenti di Fondamenti di Intelligenza Artificiale 2024/25
Presentato da	Battaglia Daniel, Pennarella Fabio
Approvato da	Palomba Fabio
GitHub	https://github.com/DanielBattaglia02/EduProfiler

Team Members

Nome	Ruolo	Acronimo	Informazioni di contatto	Matricola
Daniel Battaglia	Team member	DB	d.battaglia8@studenti.unisa.it	05121 16559
Fabio Pennarella	Team member	FP	f.pennarella@studenti.unisa.it	05121 17301

Revision History

Data	Versione	Descrizione	Autori
02/01/2025	v1	Stesura iniziale	DB
03/01/2025	v2	Specifiche PEAS + soluzione del problema	FP
07/01/2025	v3	Completamento soluzione del problema + considerazioni + Ottimizzazioni + revisione parziale	DB
08/01/2025	v4	Prima revisione completa	FP
11/01/2025	v5	KFoldCrossValidation	DB
12/01/2025	v6	Revisione finale	DB, FP

Sommario

1. Introduzione

L'**intelligenza artificiale (IA)** è diventata una componente fondamentale non solo nel campo informatico, ma anche del nostro quotidiano. È importante conoscerne le potenzialità, così da poterla utilizzare efficientemente. Il nostro progetto include l'uso dell'IA per soddisfare il problema che successivamente descriveremo.

2. Definizione del problema

2.1. Obiettivi

Lo scopo del nostro progetto è creare **un'applicazione desktop** che permetta all'utente di inserire i dati di un nuovo studente, e in base ad essi predirne l'indice accademico. Questo indice rappresenta un criterio di valutazione dello studente e che approfondiremo tra poco.

2.2. Specifiche PEAS

Le specifiche **PEAS** (*Performance measure, Environment, Actuators, Sensors*) sono un modello utilizzato in ambito **IA** per descrivere in modo chiaro ed efficiente un problema. In altre parole, le PEAS sono un insieme di specifiche che definiscono il comportamento e le caratteristiche di un agente intelligente. Ogni elemento del modello PEAS serve a specificare un aspetto fondamentale di come l'agente percepisce l'ambiente e come interagisce con esso.

2.2.1. Performance Measure (Misura di prestazione)

Come misura di prestazione dell'applicativo **EduProfiler** si tiene in conto della sua abilità di dare un risultato corretto rispetto al risultato atteso, e che il tutto venga svolto dall'applicativo in un tempo ragionevole.

2.2.2. Environment (Ambiente)

L'ambiente di esecuzione è un ambiente:

- **Completamente osservabile**, in quanto l'agente ha accesso a tutte le informazioni rilevanti sull'ambiente (quali: media voti di ogni studente, media ore di studio di ogni studente, numero attività extra di ogni studente).
- **Deterministico**, in quanto l'esito di una previsione è completamente prevedibile e non dipende da fattori casuali.
- **Statico**, in quanto l'ambiente non cambia mentre l'agente prende decisioni.
- **Episodico**, in quanto le azioni dell'agente sono indipendenti, e ogni decisione riguarda un singolo episodio senza influenzare i successivi.
- **Agente singolo**, in quanto c'è un solo agente che interagisce con l'ambiente.

2.2.3. Actuators (Attuatori)

Gli attuatori rappresentano il meccanismo attraverso cui l'agente restituisce una risposta o influenza l'ambiente. Gli attuatori del sistema sono:

- Output del risultato:
  - Il sistema apprende l'indice accademico del nuovo studente e lo mostra su linea di comando.

2.2.4. Sensors (Sensori)

I sensori rappresentano il meccanismo attraverso cui l'agente acquisisce informazioni sull'ambiente. I sensori del sistema sono:

- **Dataset**:
  - Al sistema viene reso disponibile un dataset di studenti etichettati (media voti, media ore di studio settimanali, numero attività extra, indice accademico) per poter così predire l'indice accademico di un nuovo studente.
- **Interfaccia utente**:
  - I dati di un nuovo studente di cui dovrà essere predetto l'indice accademico vengono inseriti dall'utente tramite un'interfaccia.

3. Analisi del problema

Il progetto è caratterizzato da un unico agente, il cui scopo è quello di apprendere l'indice accademico di un nuovo studente, in base ai valori inseriti dall'utente, ed apprendendo dal dataset attraverso un **algoritmo di machine learning**, ovvero la componente **IA**.

Ogni studente possiede tre caratteristiche:

- **media voti**;
- **media ore di studio settimanali**;
- **numero di attività extra-curriculari che svolge**.

All'effettivo valore di ogni caratteristica è associata una sotto-categoria da tre possibili parametri: "**bassa**", "**media**" e "**alta**".

Per ogni sotto-categoria ci sono dei **range**:

- Media voti:
  - bassa: >=18 && <=23;
  - media: >=24 && <=26;
  - alta: >=27 && <=30;
- Media ore di studio:
  - bassa: >=1 &&<=10;
  - media: >=11 && <=20;
  - alta: >=21;

- Numero attività extra-curriculari:
  - bassa: >=0 && <=3;
  - media: >=4 && <=6;
  - alta: >=7.

Ogni studente, oltre ai suoi 3 attributi principali, avrà quindi una categoria, caratterizzata da una combinazione di tre elementi (i tre attributi principali), ognuno da tre possibili valori {“bassa”, “media”, “alta”}. Il **dataset** sarà composto da **27 studenti**, numero sufficiente per gestire tutte le combinazioni (3^3).

3.1. Indice accademico

L’**indice accademico** ha lo scopo di indicare l’efficienza accademica di un singolo studente in base ai valori dei suoi attributi (media voti, media ore di studio settimanali e numero attività extra-curriculari).

Per ogni studente nel dataset, l’indice accademico verrà calcolato attraverso i seguenti criteri:

- Ogni attributo ha un punteggio:
  - bassa: 1;
  - media: 2;
  - alta: 3;
- La somma dei punteggi di ogni attributo indicherà il valore dell’indice accademico (ex: 3+1+1):
  - **basso**: <=3;
  - **medio**: <=6;
  - **alto**: <=7.

Per i nuovi studenti, l’indice accademico non verrà calcolato, ma predetto attraverso l’**IA**.

3.2. Machine learning

Quello che stiamo trattando è un **problema di classificazione**, la cui definizione è la seguente:
***“Task in cui l’obiettivo è predire il valore di una variabile categorica, chiamata variabile dipendente, target, o classe, tramite l’utilizzo di un training set, ovvero un insieme di osservazioni per cui la variabile target è nota”*.**

I problemi di classificazione sono istanze di problemi di apprendimento supervisionato, e quindi di ***machine learning***, dove l’apprendimento avviene attraverso un **training set** (un dataset di dati etichettati). L’etichetta determina la variabile dipendente, ovvero quello che l’agente dovrà apprendere. Nel nostro contesto, **l’etichetta è l’indice accademico**, e l’agente ne deve apprendere il valore.

L'algoritmo usato è l'**albero decisionale**, il quale mira a creare un albero i cui nodi rappresentano gli attributi/caratteristiche, e i cui archi ne rappresentano i valori (decisioni). Ecco il suo svolgimento:

- La miglior caratteristica del training set viene posizionata alla radice;
- Il training set viene diviso in sotto-insiemi, ognuno composto da valori simili per una certa caratteristica;
- I primi due step vengono ripetuti fino a quando non viene raggiunto un nodo foglia in ogni sotto-albero.

3.3. Information gain ed entropia

Per scegliere quale attributo debba dividere il dataset è stato utilizzato l'**Information Gain**, ovvero "***la misura che indica il grado di purezza di un attributo, ovvero quanto un certo attributo sarà in grado di dividere adeguatamente il dataset***".

Alla base dell'information Gain è presente l'**entropia**, che nella teoria dell'informazione indica in che misura un messaggio è ambiguo e difficile da capire. Maggiore è l'entropia, maggiore è l'ambiguità.

4. Soluzione del problema

In base a quanto detto precedentemente, per creare il nostro agente in grado apprendere, occorre:

- generare il training set;
- implementare la logica dell'albero decisionale;
- includere Information Gain ed entropia nella seguente logica.

Abbiamo considerato, secondo il problema da risolvere, più adatto un algoritmo di apprendimento invece che di ricerca. Ecco alcuni vantaggi:

- **Generalizzazione**: è in grado di generalizzare i risultati, ovvero di trovare soluzioni di dati non ancora esistenti, attraverso appunto l'apprendimento del dataset;
- **Scalabilità**: Gli algoritmi di machine learning sono progettati per funzionare bene con dataset di qualsiasi dimensione. Se in futuro il dataset cresce o vengono introdotti nuovi attributi, l'algoritmo di apprendimento può essere riaddestrato per integrare i nuovi dati senza dover riscrivere l'intera logica;
- **Efficienza computazionale**: Una volta addestrato, il modello è molto rapido nel fare predizioni. Non ha bisogno di calcolare soluzioni da zero ogni volta che viene presentato un nuovo input.

4.1. Tecnologie utilizzate

Linguaggio di programmazione	Java
IDE	IntellIJ
jdk	21
Librerie GUI	Java Swing e Java AWT
Framework Testing	Junit Inclusione di 4 librerie nella cartella ***lib***: - junit-jupiter-api-5.10.0.jar; - junit-jupiter-engine-5.10.0.jar; - junit-platform-commons-1.10.0.jar - junit-platform-engine-1.10.0.jar Download da: Maven Central Repository
Tool generazione documentazione	javadoc

4.2. Implementazione Information Gain

L'information gain è una misura utilizzata negli alberi decisionali per determinare quale attributo utilizzare come nodo in un punto specifico dell'albero. Nel caso del nostro applicativo, che calcola la previsione dell'indice accademico di uno studente tramite l'utilizzo di un albero decisionale, si ha bisogno di scegliere quali degli attributi di uno studente (scelto tra: media voti, media ore di studio settimanali, numero attività extra) dovrà essere utilizzato come nodo radice dell'albero o di un sottoalbero. Dato però che i vari valori contrastanti degli attributi all'interno del dataset appaiono con la stessa frequenza, l'information Gain risulterà uguale per tutti.

$$Gain(D, A) = H(D) - \sum_{v \in values(A)} \frac{|D_v|}{|D|} \bullet H(D_v)$$

dove
- D è l'entropia del dataset;
- D_v è il sottoinsieme di D per cui l'attributo A ha valore v;
- $|D_v|$ è il numero di elementi di D_v;
- $|D|$ è il numero di elementi del dataset.

```
public static double calcolaInformationGain(List<Studente> studenti, String
attributo) {
  double entropiaIniziale=Entropia.calcolaEntropia(studenti);

  Map<String, List<Studente>> suddivisioni = studenti.stream()
          .collect(Collectors.groupingBy(
                  studente -> getValoreAttributo(studente, attributo)
          ));

   double entropiaCondizionale = 0.0;
   int totale = studenti.size();

   for (List<Studente> sottoinsieme : suddivisioni.values()) {
       double peso = (double) sottoinsieme.size() / totale;
       entropiaCondizionale += peso * Entropia.calcolaEntropia(sottoinsieme);
   }

   return entropiaIniziale - entropiaCondizionale;
}
```

4.3. Implementazione Entropia

L'entropia serve per misurare il livello di "disordine" o incertezza in un insieme di dati. In una classificazione, l'entropia quantifica quanto i dati sono misti in termini delle loro etichette. Nel nostro caso si vuole calcolare l'entropia del dataset iniziale dato in input alla nostra applicazione, ossia un insieme di dati su 27 studenti, etichettati tramite l'indice accademico.

$$H(D) = -\sum_{c} p(c) \bullet log_2 p(c)$$ dove p(c) è la proporzione della classe c nel dataset D.

```java
public static double calcolaEntropia(List<Studente> studenti) {

    if (studenti.isEmpty()) {
        return 0.0;
    }

    Map<String, Long> frequenze = studenti.stream()
            .collect(Collectors.groupingBy(
                    studente -> studente.getCategoria().getIndiceAccademico(),
                    Collectors.counting()
            ));

    int totale = studenti.size();
    double entropia = 0.0;

    for (long frequenza : frequenze.values()) {
        double probabilita = (double) frequenza / totale;
        entropia -= probabilita * log2(probabilita);
    }

    return entropia;
}

private static double log2(double x) {
    return Math.log(x) / Math.log(2);
}
```

4.4. Generazione dataset

Il nostro dataset è generato da una classe apposita, ed è caratterizzato da 27 studenti, risultato del numero di combinazioni dei tre possibili valori dell'indice accademico, ovvero 3^3. Come già detto in precedenza, per ognuno di loro è specificata l'etichetta, ovvero l'indice accademico, cosa che l'algoritmo dovrà apprendere.

```
public static List<Studente> generaStudenti() {
   List<Studente> studenti = new ArrayList<>();

   studenti.add(new Studente(18, 3, 2, 1));
   studenti.add(new Studente(19, 4, 3, 1));
   studenti.add(new Studente(18, 5, 7, 1));
   studenti.add(new Studente(21, 15, 2, 1));
   studenti.add(new Studente(21, 18, 4, 1));
   studenti.add(new Studente(19, 20, 6, 1));
   studenti.add(new Studente(20, 21, 1, 1));
   studenti.add(new Studente(20, 21, 5, 1));
   studenti.add(new Studente(22, 21, 9, 1));
   studenti.add(new Studente(25, 4, 2, 1));
   studenti.add(new Studente(25, 5, 4, 1));
   studenti.add(new Studente(25, 5, 8, 1));
   studenti.add(new Studente(25, 19, 2, 1));
   studenti.add(new Studente(25, 20, 5, 1));
   studenti.add(new Studente(25, 18, 6, 1));
   studenti.add(new Studente(25, 23, 2, 1));
   studenti.add(new Studente(25, 24, 5, 1));
   studenti.add(new Studente(25, 21, 9, 1));
   studenti.add(new Studente(30, 2, 2, 1));
   studenti.add(new Studente(29, 3, 4, 1));
   studenti.add(new Studente(30, 6, 10, 1));
   studenti.add(new Studente(27, 17, 2, 1));
   studenti.add(new Studente(28, 20, 3, 1));
   studenti.add(new Studente(29, 18, 7, 1));
   studenti.add(new Studente(30, 21, 2, 1));
   studenti.add(new Studente(30, 25, 5, 1));
   studenti.add(new Studente(30, 28, 12, 1));
   return studenti;
}
```

Il costruttore di uno studente prevede 4 parametri, il cui ultimo (type) indica se calcolare o meno l'indice accademico. Per creare il dataset, l'indice accademico viene calcolato per ogni studente, mentre ciò non avviene per i nuovi studenti, poichè l'obiettivo è appunto apprendere l'etichetta. Quando type=1, l'indice viene calcolato, altrimenti no.

```
public Studente(double mediaVoti, double mediaOreDiStudio, int mediaAttivitaExtra,
int type) {
   this.mediaVoti = mediaVoti;
   this.mediaOreDiStudio = mediaOreDiStudio;
   this.mediaAttivitaExtra = mediaAttivitaExtra;
   this.categoria = new Categoria(this.mediaVoti, this.mediaOreDiStudio,
this.mediaAttivitaExtra, type);
}
```

4.5. Algoritmo di apprendimento supervisionato

4.5.1. Albero decisionale

La costruzione dell'albero decisionale parte dalla generazione del nodo radice, costruendo in maniera ricorsiva i sottoalberi di ogni nodo, la cui fine è stabilita dai nodi foglia, ovvero i valori dell'indice accademico da apprendere. L'attributo di ogni nodo viene scelto attraverso il calcolo dell'information gain.

Il seguente metodo inizia la generazione dell'albero, restituendo il nodo radice.

```
public static Nodo costruisciAlbero(List<Studente> studenti) {
   List<String> attributi = new ArrayList<>(Arrays.asList("mediaVoti",
"mediaOreDiStudio", "mediaAttivita"));

   String migliorAttributo = scegliMigliorAttributo(studenti, attributi);

   return costruisciNodo(studenti, migliorAttributo, attributi);
}
```

Ogni nodo viene costruito ricorsivamente. Le liste sinistra, destra e centrale suddividono gli studenti in base al valore dell'attributo scelto (bassa, media, alta). Una volta analizzati i tre attributi di uno studente, e quindi aver costruito i tre nodi verticali, viene generato il nodo foglia con il valore dell'indice accademico corrispettivo.

```
private static Nodo costruisciNodo(List<Studente> studenti, String attributoCorrente,
List<String> attributi) {
    if (studenti.isEmpty()) {
        return null;
    }

    Nodo nodo = new Nodo(attributoCorrente);

    List<Studente> sinistra = new ArrayList<>();
    List<Studente> destra = new ArrayList<>();
    List<Studente> centrale = new ArrayList<>();

    for (Studente studente : studenti) {
        String valoreAttributo = getValoreAttributo(studente, attributoCorrente);
        if ("Bassa".equals(valoreAttributo)) {
            sinistra.add(studente);
        } else if ("Alta".equals(valoreAttributo)) {
            destra.add(studente);
        } else {
            centrale.add(studente);  // Categoria "media"
        }
    }

    List<String> attributiRestanti = new ArrayList<>(attributi);
    attributiRestanti.remove(attributoCorrente);

    String migliorAttributo = null;
    if (attributiRestanti.isEmpty()) {
        migliorAttributo = "indiceAccademico";
        attributiRestanti.add("indiceAccademico");
        attributiRestanti.add("indiceAccademico_end");
    }
    else if (attributiRestanti.contains("indiceAccademico_end")){
        return new Nodo(getIndiceAccademico(studenti), true);
    }
    else
    {
        migliorAttributo = scegliMigliorAttributo(studenti, attributiRestanti);
    }

    nodo.setSinistro(costruisciNodo(sinistra, migliorAttributo, attributiRestanti));
    nodo.setDestro(costruisciNodo(destra, migliorAttributo, attributiRestanti));
    nodo.setCentrale(costruisciNodo(centrale, migliorAttributo, attributiRestanti));

    return nodo;}
```

Questo metodo permette di navigare nell'albero decisionale, fino ad arrivare ad apprendere l'indice accademico del nuovo studente.

```
public static String prediciIndice(Nodo nodo, Studente studente) {
    if (nodo == null) {
        return null; }

    if (nodo.isFoglia()) {
        return nodo.getValorePredizione(); }

    String valoreAttributo = getValoreAttributo(studente, nodo.getAttributo());
    Nodo prossimoNodo = null;

    if ("Bassa".equals(valoreAttributo)) {
        prossimoNodo = nodo.getSinistro();
        if (prossimoNodo == null) {
            prossimoNodo = nodo.getCentrale();
            if (prossimoNodo == null) {
                prossimoNodo = nodo.getDestro();
            }
        }
    } else if ("Alta".equals(valoreAttributo)) {
        prossimoNodo = nodo.getDestro();
        if (prossimoNodo == null) {
            prossimoNodo = nodo.getCentrale();
            if (prossimoNodo == null) {
                prossimoNodo = nodo.getSinistro();
            }
        }
    } else {
        prossimoNodo = nodo.getCentrale();
        if (prossimoNodo == null) {
            prossimoNodo = nodo.getSinistro();
            if (prossimoNodo == null) {
                prossimoNodo = nodo.getDestro();
            }
        }
    }

    return prediciIndice(prossimoNodo, studente);    }
```

Il seguente metodo restituisce il valore del nodo foglia, ovvero quello dell'indice accademico corrispettivo

```
private static String getIndiceAccademico(List<Studente> studenti) {
   String indiceAccademico = studenti.get(0).getCategoria().getIndiceAccademico();
   for (Studente studente : studenti) {
       String indice = studente.getCategoria().getIndiceAccademico();
       if (indice == null) {
           continue;
       }
       if (!indice.equals(indiceAccademico)) {
           return indiceAccademico;
       }
   }
   return indiceAccademico;   }
```

Il seguente metodo sceglie il miglior attributo per ogni nodo attraverso il calcolo dell'InformationGain.

```
private static String scegliMigliorAttributo(List<Studente> studenti, List<String> attributi)
{
   String migliorAttributo = null;
   double massimoGuadagno = Double.NEGATIVE_INFINITY;

   for (String attributo : attributi) {
       double guadagno = InformationGain.calcolaInformationGain(studenti, attributo);
       if (guadagno > massimoGuadagno) {
           massimoGuadagno = guadagno;
           migliorAttributo = attributo;
       }
   }

   return migliorAttributo;  }
```

Quest'ultimo metodo serve semplicemente a fornire il valore di un determinato attributo di uno studente

```
private static String getValoreAttributo(Studente studente, String attributo) {
   switch (attributo) {
       case "mediaVoti":
           return studente.getCategoria().getMediaVoti();
       case "mediaOreDiStudio":
           return studente.getCategoria().getMediaOreDiStudio();
       case "mediaAttivita":
           return studente.getCategoria().getMediaAttivitaExtra();
       case "indiceAccademico":
           return studente.getCategoria().getIndiceAccademico();
       default:
           return ""; }
}
```

5. Miglioramenti/Ottimizzazioni

Per completare la risoluzione del nostro problema e migliorare le performance del nostro modello, una delle ottimizzazioni più utili è l'implementazione della validazione incrociata (*cross-validation*). Questo processo è cruciale per valutare la capacità di generalizzazione del nostro modello, riducendo il rischio di overfitting e ottenendo stime più robuste delle sue prestazioni. In particolare, in questo contesto, utilizzeremo la **K-Fold Cross Validation**, che è una delle tecniche più comunemente adottate nel machine learning per validare i modelli predittivi.

La **K-Fold Cross Validation** consiste nel suddividere il nostro dataset in K sottoinsiemi, chiamati **fold**. A ogni iterazione, uno di questi fold viene utilizzato come **test set**, mentre i restanti K-1 fold vengono utilizzati per addestrare il modello. Questo processo viene ripetuto K volte, ogni volta con un fold diverso utilizzato come test set. In questo modo, ogni parte del nostro dataset viene utilizzata sia per addestrare che per testare il modello, riducendo il rischio di una valutazione errata dovuta alla suddivisione casuale del dataset. Alla fine del processo, la performance del modello viene calcolata come la media delle accuratezze ottenute in ciascun fold, fornendo una stima complessiva delle prestazioni del modello su dati non visti durante l'addestramento.

L'**accuratezza** (*accuracy*) è la metrica utilizzata per valutare la qualità del modello. Essa misura la percentuale di previsioni corrette fatte dal modello rispetto al totale delle previsioni. Durante ogni iterazione della K-Fold Cross Validation, l'accuratezza viene calcolata come il numero di previsioni corrette nel test set diviso il numero totale di esempi in quel set. Dopo aver completato tutte le iterazioni, l'accuratezza media di tutte le iterazioni ci fornirà una misura robusta e affidabile delle prestazioni generali del modello. Questo approccio aiuta a mitigare gli effetti dell'overfitting, poiché il modello non viene mai testato solo su una porzione limitata del dataset, ma su diverse sotto-sezioni, permettendo di ottenere una stima più accurata della sua capacità di generalizzare su dati mai visti prima.

```
public static double eseguiKFold(List<Studente> studenti, int numeroFold) {
    Collections.shuffle(studenti);

    int foldSize = studenti.size() / numeroFold;
    double accuratezzaTotale = 0;

    for (int i = 0; i < numeroFold; i++) {
        List<Studente> testSet = new ArrayList<>(studenti.subList(i * foldSize, (i + 1) *
foldSize));
        List<Studente> trainingSet = new ArrayList<>(studenti);
        trainingSet.removeAll(testSet);

        Nodo albero = AlberoDecisionale.costruisciAlbero(trainingSet);
        System.out.println(trainingSet.size() + "--" + testSet.size());

        int correttezza = 0;
        for (Studente studente : testSet) {
            String previsione = AlberoDecisionale.prediciIndice(albero, studente);
            if (previsione==null)
            {
                return -20;}

            if (previsione.equals(studente.getCategoria().getIndiceAccademico())) {
                correttezza++;}    }

        double accuratezzaFold = (double) correttezza / testSet.size();
        accuratezzaTotale += accuratezzaFold;  }
return accuratezzaTotale / numeroFold; }
```

6. Considerazioni

L'adozione dell'albero decisionale come algoritmo di apprendimento per il nostro problema si rivela la scelta più appropriata rispetto ad altre tecniche come il KNN o il Naive Bayes. Questo approccio si dimostra particolarmente adatto grazie alla natura delle variabili categoriali, che non sono numeriche ma altamente correlate tra loro, e al dataset composto da 27 studenti. Questi elementi consentono una gestione efficiente e una buona interpretabilità da parte dell'albero decisionale.

Nonostante i numerosi vantaggi, è importante sottolineare che l'uso dell'albero decisionale implica un rischio potenziale di **overfitting**. Tuttavia, nel contesto specifico del nostro dataset, che è completo e rappresentativo di tutte le possibili informazioni, questo rischio risulta essere minimo. Il modello è in grado di generalizzare adeguatamente grazie alla presenza di un dataset che include tutte le variabili e informazioni necessarie. In presenza di nuovi dati, non inclusi nel set di addestramento, potrebbe manifestarsi un fenomeno di overfitting, qualora il modello si adattasse in maniera troppo specifica alle caratteristiche del training set, perdendo la capacità di generalizzare correttamente.

Al contrario, il rischio di **underfitting** è contenuto, poiché gli attributi presenti nel dataset sono sufficientemente esplicativi per estrarre regole valide, catturare le differenze tra le diverse classi e ottenere modelli in grado di interpretare adeguatamente i dati.

7. Glossario

Termine	Descrizione
Machine Learning	Il **machine learning** (apprendimento automatico) è una branca dell'intelligenza artificiale che si concentra sulla costruzione di algoritmi e modelli che permettono ai computer di apprendere dai dati, senza essere esplicitamente programmati per compiere compiti specifici
GUI (*Grapichal User interface)*	L'**Interfaccia utente** che consente agli utenti di interagire con un sistema, un'applicazione o un dispositivo elettronico attraverso elementi grafici visivi, come finestre, icone, bottoni e menù, anziché tramite comandi testuali o input di basso livello.
Dataset	Un **dataset** è una raccolta strutturata di dati utilizzata per analisi, addestramento di modelli, o altre operazioni computazionali. I dataset sono fondamentali in molte aree, tra cui il machine learning.
IDE (Integrated Development Environment)	Rappresenta un software che fornisce una serie di strumenti e funzionalità per facilitare la scrittura, il test e il debug del codice durante lo sviluppo di applicazioni. L'obiettivo principale di un **ambiente di sviluppo** è semplificare il processo di programmazione, migliorando l'efficienza del programmatore.
Overfitting	L'**overfitting** si verifica quando un modello si adatta troppo strettamente ai dati di addestramento, catturando non solo la tendenza generale dei dati, ma anche il rumore o le fluttuazioni casuali che potrebbero non essere rilevanti per i dati futuri. In altre parole, il modello "memorizza" i dati di addestramento invece di imparare le loro caratteristiche generali.
Underfitting	L'**underfitting** si verifica quando un modello non è sufficientemente complesso per catturare le strutture sottostanti nei dati. In altre parole, il modello è troppo semplice per rappresentare adeguatamente i dati, e non riesce a "imparare" correttamente da essi. L'underfitting può verificarsi anche quando un modello non è stato addestrato abbastanza a lungo o ha troppi vincoli.

8. Riferimenti bibliografici

- Materiale E-learning;
- Russell SJ, Norvig P. Artificial Intelligence A Modern Approach. Third edition ed. Pearson Education; 2020.