Modulo di EDUCAZIONE CIVICA

Bias algoritmico, discriminazioni e responsabilità nella programmazione

1. La responsabilità del programmatore

Quando un software prende una decisione sbagliata, la colpa non è mai "della macchina". Il software non agisce nel vuoto: è progettato da persone che scelgono regole e dati.

Il compito del programmatore non è solo scrivere codice che "gira", ma prevedere le *conseguenze delle proprie scelte tecniche*.

Scelte superficiali che possono generare errori (danni/discriminazioni):

- **Dataset distorti:** Usare dati storici che contengono già pregiudizi.
- **Test insufficienti:** Non verificare come il software si comporta con diverse categorie di utenti.
- **Codice "rigido":** Scrivere logiche difficili da controllare o modificare.

Concetto chiave: Etica by Design. *La protezione dei diritti e l'equità devono essere parte del progetto fin dalla prima riga di codice, non aggiunte in seguito come correzioni*.

➢ **Profili di responsabilità**

- **Responsabilità Civile e Professionale:** Se il codice causa un danno a terzi (es. un bug che espone dati sensibili di migliaia di utenti), l'azienda risponde inizialmente, ma può rivalersi sul programmatore se dimostra la sua **colpa grave** (ovvero l'aver ignorato standard minimi di diligenza).
  Se il programmatore è freelance, risponde direttamente del mancato rispetto dei requisiti tecnici o delle scadenze.
  Sta diventando prassi per i programmatori stipulare polizze di Responsabilità Civile Professionale (RC Professionale) per coprire errori od omissioni.
- **Responsabilità legale:** Solitamente ricade sull'azienda, ma in caso di **grave negligenza tecnica** (es. ignorare bug noti o norme di sicurezza), può emergere una responsabilità diretta del programmatore.
- **L'Obbligo di Segnalazione:** Legalmente, il tecnico ha un "dovere di avviso"; se ravvisa un pericolo, deve comunicarlo formalmente. Questo sposta la responsabilità decisionale sul management.
- **L'AI Act Europeo:** Le nuove normative UE stanno introducendo regole rigide per le AI ad "alto rischio" (salute, giustizia, lavoro), imponendo trasparenza e controllo umano. La legge richiede:
  - **Data Governance:** Devi dimostrare che i dati di addestramento non sono distorti (*bias*).
  - **Documentazione Tecnica:** Ogni scelta architetturale deve essere tracciata.
  - **Human-in-the-loop:** Il software deve permettere un intervento umano che possa sovrascrivere o fermare l'IA.
- **Sanzioni:** Le multe possono arrivare fino al **7% del fatturato annuo globale** o 35 milioni di euro. Sebbene queste colpiscano l'azienda, le indagini interne per individuare i responsabili tecnici della "mancata conformità" saranno inevitabili.

- **Problema delle IA autonome (black box)**

Con alcuni sistemi di **intelligenza artificiale molto complessi**, diventa difficile capire esattamente **come il sistema abbia preso una decisione**.

Questo fenomeno è chiamato **"black box"**.

In questi casi la tendenza delle normative è quella di attribuire la responsabilità:

- allo **sviluppatore**, se il sistema è progettato in modo difettoso
- all'**utilizzatore professionale**, se non vengono rispettate le procedure di sicurezza.

2. Il Bias Informatico (Pregiudizio Algoritmico)

Il **bias** è un errore sistematico che produce risultati discriminatori. Raramente è intenzionale: spesso nasce da errori di modellazione della realtà. Questi errori possono favorire o penalizzare specifici gruppi demografici, influenzando aree critiche come selezione del personale, giustizia e riconoscimento facciale.

Esempi di bias nella progettazione

- **Bias di rappresentazione**

Se in una classe Cittadino si rende obbligatorio l'attributo:

```
string CodiceFiscale
```

si potrebbero escludere gli stranieri che non lo possiedono ancora.

- **Stereotipizzazione nelle proprietà**

Se in un sistema HR definiamo:

```
Enum Genere { Maschio, Femmina }
```

stiamo riducendo una realtà complessa a uno schema binario rigido che potrebbe non rappresentare tutti gli utenti

- **Proxy Variables (bias nascosto)**

A volte il bias non è esplicito ma nascosto in variabili che **indirettamente rappresentano altre informazioni**.

Esempio:

Un sistema di reclutamento elimina la variabile `Genere` per essere imparziale, ma mantiene la variabile `Hobby`. L'algoritmo potrebbe imparare che alcuni hobby sono più frequenti in un certo genere, ricreando la discriminazione.

- **Bias nei dati**

Se si istruisce un'IA con dati storici che contengono discriminazioni (es. selezioni del personale passate che favorivano solo uomini), l'IA imparerà a discriminare.

- **Bias nel design:**

Quando chi programma non tiene conto delle diversità (es. sistemi di riconoscimento facciale che funzionano peggio su carnagioni scure perché testati solo su campioni caucasici).

3. Dove si nasconde il bias nel codice (OOP)

I pilastri della programmazione a oggetti (OOP) possono essere usati impropriamente:

- **Gerarchie di classi rigide**

Se una classe Lavoratore contiene un metodo `CalcolaPensione()` che non considera periodi di maternità o part-time o lavori precari, il sistema produrrà risultati discriminatori.

- **Incapsulamento e opacità**

Un algoritmo di selezione come

```
bool IsCandidatoIdeale()
```

che restituisce solo "vero" o "falso", nasconde criteri decisionali (non visibili all'utente) che non saprà mai perché è stato scartato. Questo porta al problema delle **black box**.

4. Programmazione Etica: Interfacce e Dependency Injection

Per evitare che il codice nasconda bias, dobbiamo renderlo **trasparente** e **verificabile**.

- **Problema dell'hard-coding: s**crivere regole direttamente nel codice in modo fisso.

```
public bool IsIdoneo(Candidato c)
{
  if (c.Eta > 40)
      return false;
  if (c.Nazionalita != "Italiana")
      return false;
  return true;
}
```

Quali sono i problemi?

- la regola è **nascosta dentro il codice**
- è difficile da modificare
- potrebbe contenere **bias discriminatori**
- nessuno dall'esterno può verificare facilmente la logica

- **Uso delle interfacce**

Invece di scrivere logiche decisionali fisse (che nascondono bias), usiamo le *interfacce* per rendere i criteri di scelta intercambiabili e verificabili.

Le interfacce permettono di separare **le regole decisionali dal sistema principale**.

```
public interface IValutazioneEtica
{
    bool IsIdoneo(Candidato c);
}
```

Questo significa:

- **la regola di valutazione non è più fissa**
- può essere implementata in modi diversi.

Esempio:

```
public class ValutazioneSoloCompetenze : IValutazioneEtica
{
    public bool IsIdoneo(Candidato c)
    {
        return c.Competenze >= 70;
    }
}
```

Oppure un'altra:

```
public class ValutazioneInclusiva : IValutazioneEtica
{
    public bool IsIdoneo(Candidato c)
    {
        return c.Competenze >= 60 && c.Esperienza > 2;
    }
}
```

- **Dependency Injection**

La regola decisionale viene fornita dall'esterno invece di essere scritta dentro la classe.

```
public class SistemaReclutamento
{
    private  IValutazioneEtica regola;

    public SistemaReclutamento (IValutazioneEtica regolaFornita)
    {
        regola = regolaFornita; // Dependency Injection: la regola viene
                                // "iniettata" dall'esterno
    }
    public void ElaboraCandidatura(Candidato c)
    {
        if (regola.IsIdoneo(c))
                ...
    }
}
```

Vantaggi Etici:

1. **Trasparenza:** La logica non è più sepolta nel codice.
2. **Audit Algoritmico:** Un ente terzo può testare il sistema iniettando una regola di controllo per verificare se l'algoritmo è imparziale.
3. **Flessibilità:** Le regole possono essere aggiornate per rispettare nuove leggi senza fermare il sistema.

5. Il Diritto alla Spiegazione (GDPR)

GDPR (General Data Protection Regulation - Regolamento UE 2016/679):
normativa europea sulla privacy, in vigore dal 25 maggio 2018, che disciplina il trattamento dei dati personali, rafforzando i diritti dei cittadini e le responsabilità di aziende e organizzazioni.
Mira a garantire la protezione dei dati, la trasparenza e la sicurezza, applicandosi a chiunque offra beni/servizi

Il **GDPR** (Regolamento UE 2016/679) stabilisce che se un cittadino è oggetto di una decisione automatizzata, ha il **diritto di ricevere informazioni significative sulla logica utilizzata**.

- **No al "Computer says no":** Un programma non deve limitarsi a un true/false. Deve essere progettato per fornire i motivi del rifiuto. Quindi è opportuno lanciare eccezioni o restituire oggetti che spieghino i motivi della decisione
- **XAI (Explainable AI):** È il settore che studia come rendere i sistemi complessi capaci di spiegare le proprie decisioni agli esseri umani.

6. Caso di Studio: Il Redlining Digitale

Il **redlining digitale** è una forma di discriminazione moderna che utilizza algoritmi, big data e decisioni aziendali per limitare l'accesso a servizi tecnologici, finanziari o educativi a specifiche popolazioni o aree geografiche, spesso a basso reddito o minoranze.
Questa pratica perpetua disuguaglianze storiche, riducendo le opportunità economiche e sociali attraverso infrastrutture internet inadeguate o algoritmi discriminatori.

```
public class Cliente
{
    public double Reddito { get; set; }
    public string CAP { get; set; } // Il bias di residenza
    public int Eta { get; set; }
}
```

```
public class Banca
{
    public bool ApprovaMutuo(Cliente c)
    {
        // Bias Geografico: se il CAP è di una zona "povera",
        //sei considerato a rischio → nega il mutuo.
        if (c.CAP == "12345")
            return false;
        return c.Reddito > 30000;
    }
}
```

Anche se il programmatore non ha inserito "razza" o "religione", l'uso del CAP può produrre lo stesso effetto discriminatorio su base statistica.

7. Conclusione

Come programmatori, siamo i "legislatori" del mondo digitale: scriviamo le leggi digitali che governano la vita delle persone. Un errore o una scelta superficiale nel codice può negare a una persona un lavoro, un prestito o un diritto fondamentale.

La qualità del codice non è solo una sfida tecnica, è una forma di giustizia sociale.