APPUTNI SVILUPPO E CICLO VITALE DI SOFTWARE DI I.A.

**1. Introduzione al corso**

* **Obiettivi generali:**
  1. Analizzare un problema e individuare un modello ML adatto
  2. Deploy non banale del modello
  3. Strategie di upgrade e monitoraggio continuo

**2. Prerequisiti**

* Interazione da linea di comando (Linux Bash)
* Conoscenza base di Python (if/else, loop, funzioni, scripting)
* Fondamenti di ML/DL (utile ma non indispensabile)

**3. Version Control Systems (Git)**

**3.1 Perché usare un VCS**

* **Ripristino** di versioni precedenti
* **Collaborazione** senza conflitti manuali
* **Tracciamento** di chi/come/quando ha modificato

**3.2 Concetti chiave**

* **Repository**: cartella inizializzata con git init (crea .git/)
* **Staging area**: dove “prepari” le modifiche da includere nel commit
* **Commit**: snapshot del progetto con autore, data, messaggio e hash
* **Catena di commit**: la storia lineare (o ramificata) del tuo lavoro
* **.gitignore**: file per escludere da Git dati troppo grandi (checkpoint, dataset, ecc.)

**3.3 Flusso di lavoro base**

bash

CopiaModifica

git init

git add <file o cartella>

git commit -m "Descrizione del cambiamento"

**4. Integrazione Git in VS Code**

* **Source Control Panel**: viste di file modificati, staged e diff
* **GitLens** (estensione consigliata):
  + Blame line-by-line
  + Visualizzazione grafica dei branch e dei commit
* **Operazioni comuni** (stage, commit, push/pull) direttamente dall’IDE
* **Branching & Merging**:
  + Branch isolate (feature, bugfix, refactor…)
  + Merge automatico vs conflitti manuali

**5. DevOps vs MLOps**

| **Aspetto** | **DevOps** | **MLOps** |
| --- | --- | --- |
| **Artefatto** | Applicazione eseguibile | Modello ML serializzato |
| **Validazione** | Test unitari/integration | Valutazione su dataset di test (performance metric) |
| **Versioning** | Codice + artefatti | Codice + dati + iperparametri + modello + metriche |
| **CI/CD** | Build → test → deploy | Training → validazione → deploy → monitoraggio |
| **Feedback loop** | Deploy → user feedback → iterazioni | Monitoring performance → retraining → redeploy |

**6. Processi chiave MLOps**

1. **Continuous Integration (CI):**
   * Automazione di lint, test (unit/integration), verifica dipendenze
2. **Continuous Delivery/Deployment (CD):**
   * Rilascio automatico in ambienti di staging/produzione
3. **Monitoring & Logging:**
   * Tracciamento di metriche (accuracy, latenza, drift dei dati)
   * Audit trail degli esperimenti

**7. Esercizio consigliato**

1. **Crea un nuovo repo Git** (anche su GitHub) e clonalo in VS Code.
2. Implementa un piccolo script Python (es. “Hello, MLOps!”) e fai almeno due commit distinti.
3. Crea un branch “feature” e prova a fonderlo nel branch principale, gestendo eventuali conflitti.
4. Configura un semplice workflow CI con GitHub Actions (ad esempio un job che esegue pytest).

Source Control 🡪 vuol dire andare a sviluppare un progetto in modo che tutta la struttra (progetti e file) siano organizzati. In modo da non dover fare copia e incolla del tutto, per risolvere questo problema utilizzeremo dei VSC (Version Control System), questi fanno si che possiamo tenere traccia delle nostre modifiche:  
-restore 🡪 ci permette di tornare ad una versione precedente del nostro codice

-check out 🡪 Prendo la specifica versione di quel file che stavo modificando per vedere com’era

-recover 🡪 Git è capace di riconoscere i file cancellati, questo fa si che possiamo recuperare vecchi file

-collaborate 🡪 ci permette di collaborare con altre persone (contemporaneamente o no) riuscendo a lavorare insieme in modo logico ed intelligente

Git è uno solo di questi VSC ma ce ne sono altri, è però il più famoso ed è generalmente lo standard aziendale

Dobbiamo decidere cos’è una repository, la possiamo vedere come cartella di progetto, dobbiamo fare git init per inizializzarla. Se cancelliamo questa cartella .git perdiamo sia la storia che i file .

I commit sono i cambiamenti del nostro codice, è un singolo cambiamento generamente (può essere anche piccolissimo), il commit lo possiamo vedere come un check point (non deve essere per forza funzionante ma è buona prassi che lo sia). Per essere valido il commit deve avere: autore, data (automatica), un messaggio commento che descrive il commit, quando facciamo un commit viene generato un hash (è un hash univoco per quella repository, quindi se cerchiamo l’hash di un determinato commit ti riporta sempre li).

Git fa la copia di un commit, mentre altri sistemi generano il delta (guardare schema dei commit sul pdf).

Il commit crea una catena (uno dopo l’altro non sparsi), per fare effettivamente questo commit dobbiamo, in una stage area, indicare un container virtuale in cui buttiamo dentro cose finchè non abbiamo finito. Prima di fare un commit dobbiamo dire ciò che vogliamo fare.

Git ignore 🡪 file speciale che contiene all’interno tutti i nomi dei file con estensioni, asterischi … ovvero qualcosa che contiene cose che vanno ignorate e cancellate durante il progetto.

Cose da scaricare:

-Python 🡪 interprete, quindi si porterà le libreria all’interno.

-Miniconda 🡪 package manager per python che fa anche la gestione di ambienti virtuali. Grazie a questo potremo avere differenti interpreti e differenti versioni.

VSCode 🡪 IDE

VSC 🡪 Git:  
Nell’explorer vedremo il file con una lettera maiuscola di fianco:

-A 🡪 se abbiamo aggiunto un nuovo file alla repository

-M 🡪 se è modificato

-D 🡪 se l’abbiamo cancellato

-U 🡪 se non è tracciato

-C 🡪 se va in confiltto

-R 🡪 se lo rinominiamo

-S 🡪 se c’è un submodule (nella repository esiste un’altra sottorepository)

-T 🡪 se cambiamo il tipo

Nel source control abbiamo vari simboli e opzioni:

+ 🡪 mette in stage (contenitore virtuale che una volta che decido che quel cambiamento lo voglio lo mette dentro)

Freccia indietro 🡪 permette di tornare inditero (se il file non è tracciato viene cancellato)

Le credenziali non servono a niente (a meno che non si lavori in cloud) servono solo per una stringa di git

+ (changes) 🡪 serve per tutti i file sottostanti

Branches, sono set isolati di commit che ci permettono di lavorare in maniera separata con la versione del codice, idealmente il branch è una feature, se sbagliamo un commit o qualcosa cancelliamo il branch e ritorna come prima. Serve molto se si lavora insieme.

Una volta fatto il Branch si farà il Merging, mettere apposto il nostro tree (automatico se vengono fatte modifiche in momenti diversi o in punti diversi del file), altrimenti c’è da fare quello manuale, generalmente si fa così: si modifica il file con conflitto, si risolve il conflitto, si rimette in stage, si fa il commit scrivendo che abbiamo risolto il conflitto e poi si rimette in merge.

DevOps:

Sviluppo e rilascio del codice, nello sviluppo si fa pianificazione del codice, si affrontano i problemi e faccio i test. Una volta sviluppato dobbiamo rilasciarlo, vedere se funziona… Verranno fatte in automatico da sistemi di controllo che andremo a programmare.

MLOps:

Simile al DevOps (concettualmente) ma ci sono differenze nei concetti chiave. Nel DevOps c’è il codice che poi viene wrappato e ne viene creato un eseguibile, validato su un dei test.

Data science and EDA (exploratory data analyst)

ML parte dai dati, quindi devo essere sicuro di cosa sto trattando, se decido di addestrare un sistema basato su dati non esistenti il modello non funzionerà. Parte sempre tutto dai dati ! I modelli tramite i dati sparano a caso finchè non beccano la risposta corretta, questo perché sono modelli STATISTICI. I dati hanno una forma e una rappresentazione, una buona norma per rappresentarli è la gaussiana. I modelli imparano dai dati una sorta di rappresentazione statistica, se ho dei dati messi male il tutto avrà una ripercussione sul modello che stiamo andando a rappresentare.

Data science:

-possiamo avere del rumore (dati che creano conflitti)

-possiamo avere pochi dati

-Biases, ne abbiamo diversi, quello più classico è porsi la domanda e sapere da dove arriva una certa risposta

-dati sbilanciati 🡪 posso avere delle classi più rare rispetto alle altre

Insomma una cattiva qualità del dato si riperquote sul nostro modello.

Data science workflow:

* + 1. Si parte dai dati
    2. Bisogna pulire i dati e verificare che non ci siano errori
    3. Dobbiamo capire come rappresentare questi dati
    4. Facciamo un preprocessing (scala i tuoi dati che sono in un dominio numerico e lo devo portare in qualcosa di interpretabile dal sistema, si normalizza il dato)
    5. Model training 🡪 allenamento dell’allenamento del modello. Ci sono dei dati di testing che devono essere esterni al modello.
    6. Misurazione delle performance 🡪 tendenzialmente si fa in modo statistico.

I dati  
Ogni dato è diverso, ci sono quelli

Quantitativi 🡪 continui (altezza, …) praticamente valori infiniti, discreti (numero di qualche oggetto) numeri finiti. Essenzialmente dati che posso misurare con un numero

Qualitativi 🡪 dati strutturati (tabelle, ...) sono dati strutturati appunto, dati non strutturati (dati di partenza, sorgenti testuali, codice delle applicazione)

Misure 🡪 nominali, ordinali, intervalli o di rapporto

Explorartory data analysis

Andare a capire i dati che stiamo utlizzando, utilizziamo quindi degli strumenti di visulaizzazione qualitativi. Possiamo utilizzare anche strumenti quantitativi per i nostri dati. Abbiamo anche una parte di apply processing

Graphical tools

BOX PLOT:  
Ci mostra il nostro grafico del dato, per poter analizzare il nostro dataset

HISTOGRAM:

Rappresentazione visuale della distribuzione quantitativa dei dati.

RUN CHART:

se abbiamo delle serie temporali dobbiamo riuscire a fare un analisi dell’evoluzione del dato lungo una variabile.

SCATTER PLOT:

Serve per poter mettere in relazione i dati tra di loro.

RIDUZIONE DI DIMENSIONALITA’:  
Ci possiamo trovare in una situazione in cui un dato è difficile da rappresentare

MODELLI DI RETI NEURALI ED ADDESTRAMENTO:

Guarda cosa sono i gradienti.

AutoMl tecnologia che ci permettono di risolvere in maniera automatica i problemi

Python Packaging:

Essendo un lingauggio interpretato ha vari problemi, non ha una vera e prorpia definitzione di inizio e fine, Attenzione agli import relativi e assoulti.