

UNIVERSITA' DEGLI STUDI DI
NAPOLI FEDERICO II



Scuola Politecnica e delle Scienze di Base
Corso di Laurea in Ingegneria Informatica

Elaborato finale in **Ingegneria del Software**

***Machine Learning Operations:
principi, caratteristiche ed applicazione***

Anno Accademico 2020-2021

Candidato:
Antonio Romano
matr. N46004321

Tesi svolta con *dedizione e pazienza*
ai tempi del *Coronavirus*

L'autore

Indice

Introduzione	5
Capitolo 1: Metodologie di sviluppo software.....	7
1.1 Le metodologie pesanti	7
1.1.1 Il modello a cascata	7
1.2 Le metodologie iterative	8
1.2.1 Il modello a spirale.....	8
1.3 Le metodologie Agili	9
1.3.2 Il perfezionamento della metodologia Agile: DevOps	11
1.4 DevOps	11
1.4.1 Differenze tra Scrum e DevOps	14
1.4.2 Applicazione dei principi DevOps nei sistemi ML	16
Capitolo 2: Il Machine Learning	17
2.1 I diversi apprendimenti di una macchina.....	18
2.1.1 Apprendimento supervisionato (<i>supervised learning</i>)	18
2.1.2 Apprendimento non supervisionato (<i>unsupervised learning</i>).....	18
2.1.3 Apprendimento per rinforzo (<i>reinforcement learning</i>)	19
2.2 La modellazione predittiva e le sue tecniche	19
2.3 Quando può essere utile il Machine Learning?	20
2.4 Esempi di applicazioni di apprendimento automatico	21
2.5 Evoluzione futura del Machine Learning	22
Capitolo 3: Il Machine Learning Operations (MLOps).....	23
3.1 Cos'è MLOps	24
3.2 I problemi che risolve MLOps	26
3.3 I principi di MLOps.....	26
3.3.1 Il Versioning.....	27

3.3.2 La Riproducibilità	27
3.3.3 Il Test	28
3.3.4 L' Automazione	29
3.3.5 Il Feature Store	31
3.3.6 Il Continuous Integration	32
3.3.7 Il Continuous Distribution	32
3.3.8 Il Continuous Monitoring	32
3.4 Implementazioni MLOps	36
3.5 Applicazioni MLOps	37
Capitolo 4: Esempio di Applicazione MLOps	39
4.1 Obiettivi del modello Machine Learning	40
4.2 Il dataset scelto	40
4.3 Preparazione dei dati	42
4.3.1 Pulizia dei dati	44
4.3.2 Visualizzazione dei dati	45
4.3.3 Training dei dati, suddivisione dei dati e predizione	47
4.4 Addestramento e sperimentazione del modello	50
4.4.1 Analisi dell'accuratezza degli algoritmi di addestramento	50
4.4.2 Data Versioning Control (DVC) dei dati	52
4.4.3 Test del modello ML	54
4.4.4 Track del modello ML con DVC e GitHub Actions	57
4.4.5 Test di predizione del modello ML	63
4.5 Test automatizzati per modelli di Machine Learning	66
4.6 Distribuzione del modello ML e dell'applicazione	72
4.6.1 Release su GitHub	80
4.7 Continuous Integration per il modello ML	82
Conclusioni	88
Appendice: COVID Probability Infection Checker	90
A.1 Index.html e Show.html	91
A.2 Casi d'uso	93
A.3 Evoluzione futura	95
Bibliografia	96
Ringraziamenti	99

Introduzione

Nell'ultimo decennio, l'Intelligenza Artificiale si è dimostrata disciplina di notevole importanza soprattutto in ambito medico, finanziario e dell'automotive. È proprio per questa evoluzione dell'Intelligenza Artificiale ha fatto sì che nel settore dell'Ingegneria del Software ci sia stata la necessità di trovare una metodologia o un insieme di pratiche utili per la realizzazione di prodotti software che implementano all'interno di esso il Machine Learning, una disciplina fondamentale dell'Intelligenza Artificiale. Questa è una delle tante motivazioni per cui l'autore della tesi si è spinto ad approfondire il **Machine Learning Operations (MLOps)** che nasce con l'obiettivo di colmare le carenze legate all'interdipendenza del software tradizionale con l'integrazione e al mantenimento di un modello ML basandosi sui principi DevOps.

L'obiettivo della tesi di Laurea è quello di fornire una panoramica approfondita dell'MLOps, sottolineando inizialmente l'evoluzione delle metodologie di sviluppo che negli anni sono stati adottati nei classici sistemi software, passando per l'approccio Agile e alla definizione di DevOps e del Machine Learning.

L'autore ha inoltre condotto un '*Proof of Concept*' della pratica MLOps su un modello di predizione che in ottica del "*periodo COVID-19*" può ritornare utile ai pazienti e ai dottori, ovvero la creazione di una semplice Web App utile per il calcolo di probabilità di infezione al virus di un paziente sulla base di sintomi, malattie che egli presenta e comportamenti.

La tesi è articolata in quattro capitoli più un'appendice:

- Nel **primo capitolo** viene fornita un'introduzione alle metodologie classiche di sviluppo del software fino a quelle attuali come la metodologia Agile e sue sfumature.

- Nel **secondo capitolo** si introduce il Machine Learning e le sue applicazioni attuali e future.
- Il **terzo capitolo** si concentra sulla tematica principale della tesi, ovvero ‘Che cos’è il Machine Learning Operations’, vengono raccolti i suoi principi e le sue caratteristiche.
- Nel **quarto capitolo**, come già detto, un ‘*Proof of Concept*’, dove l’autore passo passo spiega con grafici e codice dei semplici passaggi definiti nelle pratiche MLOps.
- Nell’**appendice**, una breve panoramica della Web App ‘*Covid Probability Infection Checker*’.

Capitolo 1: Metodologie di sviluppo software

Nell' Ingegneria del Software è di importanza rilevante la suddivisione del processo di sviluppo del software adottando metodologie utili per ottimizzare le varie fasi di sviluppo. La suddivisione del processo di sviluppo comprende le fasi tipiche che includono lo studio o l'analisi, la progettazione, la realizzazione e quindi la programmazione, l'ispezione e il collaudo, la messa a punto, l'installazione, la manutenzione e l'ipotetica estensione. La scelta della metodologia dipende molto dal tipo di software che si vuole realizzare e soprattutto dal team di sviluppo che in base ai ruoli, attribuisce le varie attività, individuando gli attori specifici incaricati di ciascuna attività e l'ordine in cui le attività si svolgono. Esistono diverse metodologie di sviluppo software tra cui le *metodologie pesanti*, le *metodologie iterative* e le *metodologie agili*.

1.1 Le metodologie pesanti

Il processo di realizzazione del software nelle metodologie pesanti è strutturato in una sequenza lineare di fasi di progetto consecutive che comprendono: l'analisi dei requisiti, la progettazione, lo sviluppo, il collaudo e la manutenzione. Ogni fase viene eseguita solo una volta e i risultati di ogni fase rappresentano le precondizioni per la fase successiva.

1.1.1 Il modello a cascata

Un modello tipico delle metodologie pesanti è il modello a cascata (*waterfall model*). È un approccio di progettazione sequenziale, infatti, dopo aver completato una fase

e raggiunto una nuova fase, non è possibile tornare alla fase precedente.



Figura 1. *Semplice modello a cascata lineare.*

Il modello è stato progressivamente abbandonato dall'industria del software giacché il perseguimento della linearità richiede poca possibilità di modifica in fase di progetto a causa dei requisiti prefissati nella prima fase. Un'altra problematica rilevante è che l'utente finale è coinvolto solo dopo la programmazione nel processo di produzione e gli errori si individuano solo alla fine del processo di sviluppo. In definitiva, il modello a cascata può essere utilizzato per piccoli progetti con requisiti ben posti nella prima fase.

1.2 Le metodologie iterative

Il processo di realizzazione del software nelle metodologie iterative sopprimono i problemi delle metodologie pesanti in quanto scompone il processo di sviluppo in quattro fasi multiple, ciascuna ripetuta più volte. Le quattro fasi sono la *Pianificazione*, l'*Analisi dei rischi*, lo *Sviluppo* e la *Verifica*. I tempi di sviluppo sono molto più ristretti e dalla fase di verifica si torna poi a quella di pianificazione per applicare eventuali correzioni al risultato dello sviluppo.

1.2.1 Il modello a spirale

Un modello tipico della metodologia iterativa è il modello a spirale (*spiral model*). A differenza del modello a cascata, il modello a spirale presenta il coinvolgimento del cliente, infatti ha consapevolezza di cosa il team di sviluppo stia realizzando. In definitiva, il modello a spirale può essere utilizzato per progetti ampi e complessi che richiedono un'analisi continua del rischio.

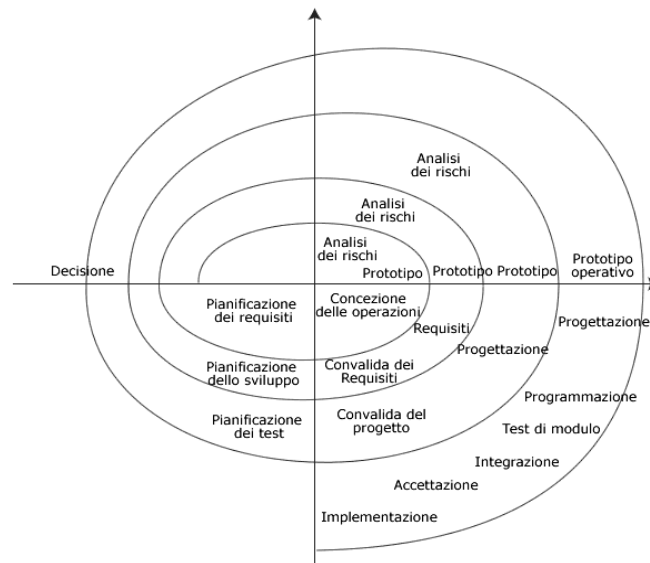


Figura 2. Modello a spirale (Fonte: Wikipedia)

1.3 Le metodologie Agili

Il processo di realizzazione del software nelle metodologie agili è fondato su un insieme di principi comuni derivati dal “*Manifesto per lo sviluppo agile del software*”. Essi riguardano *Valori, Principi, Metodi e Pratiche*. Il modello agile sopprime la rigidità dei metodi di sviluppo software tradizionali, proponendo un approccio meno strutturato e focalizzato sull'obiettivo di consegnare in tempi brevi e frequentemente software funzionante e di qualità. Nella metodologia Agile vengono riviste di continuo le specifiche, adeguandole durante l'avanzamento dello sviluppo del software mediante framework iterativi e incrementali oltre ad uno scambio di informazioni e di pareri tra gli sviluppatori e il committente. Il Manifesto riporta quattro valori fondamentali:

- Le persone e i rapporti sono più importanti dei processi e degli strumenti;
- Il software funzionante è più importante della documentazione;
- La collaborazione col cliente è più importante della negoziazione dei contratti;
- Rispondere al cambiamento è più importante che seguire un piano.

Tra i modelli che adottano il “Manifesto” si cita Scrum, il framework più diffuso ed utilizzato.

1.3.1 Scrum

Il modello di sviluppo agile Scrum presenta gli “sprint”, l’unità di base dello sviluppo adottato in questa metodologia. Ogni sprint è un piccolo progetto a sé stante e deve contenere tutto ciò che è necessario per permettere un avanzamento nel raggiungimento degli obiettivi finali. Al termine di ogni ciclo di lavoro, si effettua una riunione in cui verificare i risultati ottenuti e lo stato di avanzamento dei lavori dell’intero team di sviluppo. Si tratta dunque di un *approccio incrementale* che ottimizza, passaggio dopo passaggio la prevedibilità ed il controllo del rischio.

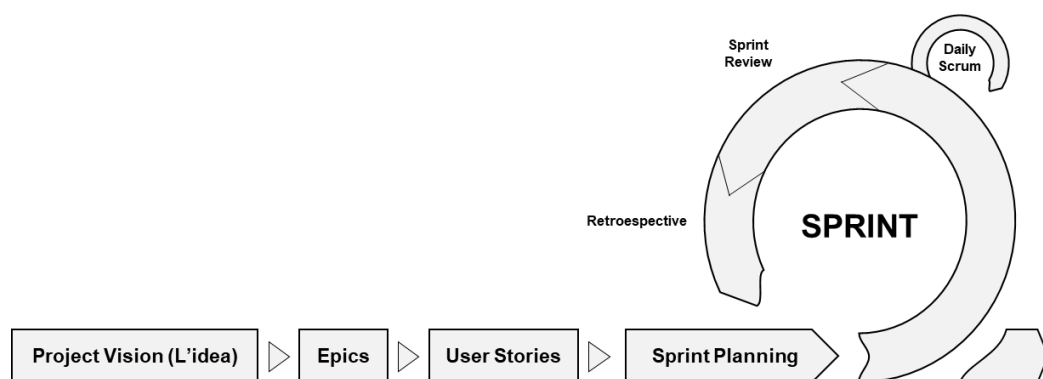


Figura 3. Modello Scrum in breve

Dalla **Figura 3** si notano diverse fasi di interesse che nella metodologia agile ha stravolto il concetto di sviluppo software:

- La fase di *Epic*, un *macro obiettivo* che non può essere raggiunto da un singolo sprint, si riferisce a una serie di requisiti che non sono stati ancora razionalizzati, privi di dettagli e devono essere suddivise in *User stories* più piccole prima di poter essere elaborate.
- Nella fase *User Story* viene definito un elenco degli elementi che devono essere realizzati all’interno di un progetto.
- Nello *Sprint* viene sviluppato un *incremento*¹ di prodotto funzionante, ognuno di essi prevede di ciò che deve essere costruito, la progettazione e il piano che guiderà la costruzione, il lavoro di sviluppo e l’incremento del prodotto risultante.

¹ L’incremento è la somma di tutti gli elementi completati durante uno sprint e durante gli sprint precedenti.

- Lo *Sprint Planning* pianifica il lavoro da eseguire nello Sprint.
- *Daily Scrum Meetings* è un incontro di 15 minuti (giornaliero), condotto quotidianamente per comprendere rapidamente l'avanzamento del lavoro da svolgere che è stato definito nel precedente Daily Scrum Meeting con l'obiettivo di creare dunque un piano per le prossime 24 ore.
- *Sprint Review* presenta l'incremento che sta per essere rilasciato.
- La *Retroespective* ha lo scopo di rielaborare ciò che è stato appreso dell'ultimo Sprint individuando gli elementi principali che sono andati bene o male e creando un piano per i miglioramenti del prodotto.

1.3.2 Il perfezionamento della metodologia Agile: DevOps

Un'estensione della metodologia Agile è DevOps in quanto amplia il concetto dei principi del "Manifesto" avvicinando la collaborazione del team dei programmatori (developers) con il team dei sistemisti (operators) estendendo il metodo Agile, oltre al codice, all'intero processo di sviluppo. Dunque DevOps e Agile non sono due metodi di sviluppo diversi ma sono uno il miglioramento dell'altro. Mentre la metodologia Agile risolve i problemi della tecnologia, DevOps cerca di risolvere più ampiamente un intero problema di business attraverso un insieme di pratiche, cultura e valori che guidano il cambiamento verso un software più solido, che raggiunge più presto la produzione e con meno intralci durante il suo ciclo di vita. Infatti DevOps rende chiaro l'obiettivo comune di Developers e Operators per puntare all'ottimizzazione dell'intero sistema invece che alle ottimizzazioni locali, cioè vuole raggiungere gli obiettivi del business risolvendo il problema del cliente, che ci sia da agire su ambiti del team dei Developers che Operators.

1.4 DevOps

DevOps è un modello di sviluppo che garantisce l'interdipendenza tra lo sviluppo software (*DEVELOPER*) e IT (*OPERATIONS*).

Negli ultimi anni, con l'avvento della "*Digital Transformation*"², DevOps ha l'obiettivo di aiutare un'organizzazione a sviluppare in modo più rapido ed efficiente i prodotti

² La *Digital Transformation* è un'evoluzione digitale nel supporto dei processi aziendali iniziata nell'ultimo decennio.

e servizi di maggiore qualità (Quality Assurance), di automatizzare lo sviluppo applicativo e i suoi processi interni, aumentare la collaborazione all'interno dei reparti IT e fare rilasci applicativi più veloci.

Nella **Figura 4** viene percepito come le componenti: Developers, Operations e Quality Assurance si intrecciano nella soluzione DevOps.

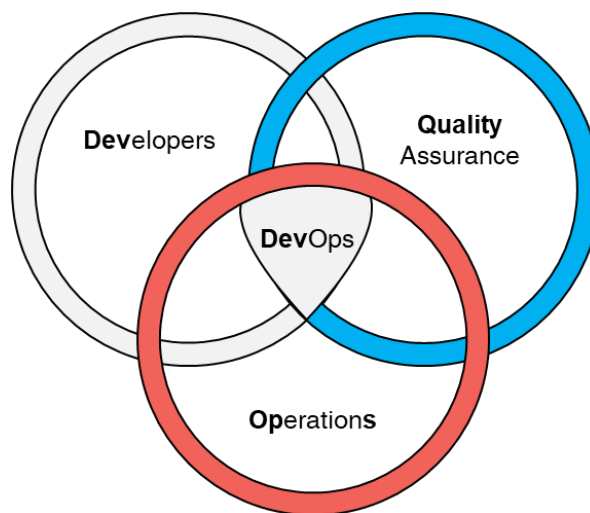


Figura 4. DevOps: intersezione tra Developer, Quality Assurance, Operations

Con la metodologia DevOps si nota dalla **Figura 5** come i rilasci applicativi più veloci riducono gli errori nell'operazione rispetto alle classiche metodologie di sviluppo:

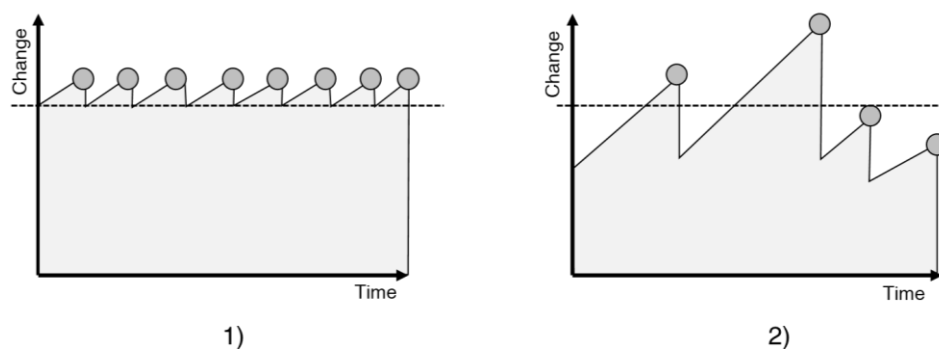


Figura 5. Differenza tra frequenza di rilasci tra 1) DevOps 2) Modello a Cascata.

Nella Figura 5.1 si hanno rilasci più frequenti e regolari (giorni, settimane) in modo da ridurre i rischi, nella Figura 5.2 si hanno rari rilasci, non regolari (quadrimestri,

anni) che implicano ad una realizzazione del software più esposta a rischi. Il ciclo di sviluppo della metodologia DevOps:

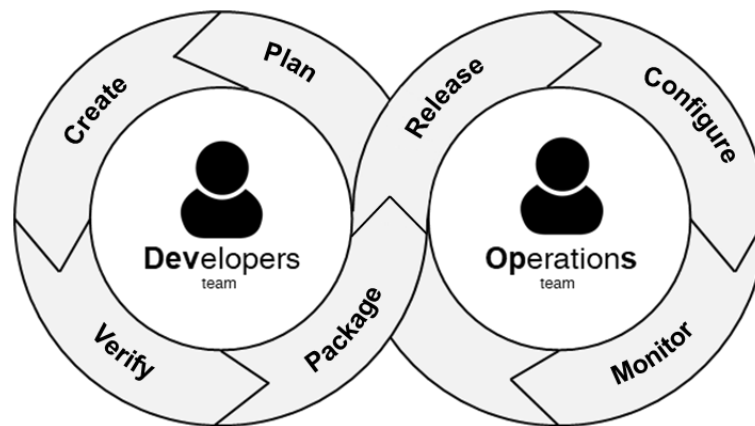


Figura 6. *Flusso di Interazione tra Developers e Operators*

Nella **Figura 6**, il Developers team è interessato alla: pianificazione, creazione, verifica e il package mentre l'Operations team al rilascio, alla configurazione e al monitoraggio. La forma "dell'infinito" del grafico interpreta la cooperazione continua tra le componenti DevOps.

La "cooperazione" tra team di sviluppatori e sistemisti portano diverse problematiche come la differenza di pensiero tra i team, la gestione del materiale complesso, tempo di deploy molto alti e ambiente di sviluppo non allineato rispetto alla produzione.

Per elidere queste problematiche, la metodologia DevOps mette a disposizione delle "best practice" tra cui il monitoraggio, i microservizi, il Continuous Integration, Il Continuous Delivery e il Continuous Deployment.

Nel **monitoraggio** viene effettuata l'acquisizione dei dati e dei log generati da applicazioni e infrastruttura suddividendoli in categorie e analizzarli esaminando le possibili cause primarie dei problemi o le modifiche impreviste.

Nella gestione dei **microservizi**, l'applicazione si basa su un gruppo di servizi di minori dimensioni. Ogni servizio esegue il proprio processo e comunica con gli altri servizi tramite un'interfaccia di programmazione di un'applicazione o API (*Application Programming Interface*) basati su metodi HTTP.

Nell' **Integrazione continua** (*Continuous Integration*) gli sviluppatori aggiungono "regolarmente" modifiche al codice in un repository centralizzato (al più remoto)

dove la creazione di build e i test vengono eseguiti *automaticamente* individuando e risolvendo i bug con maggiore tempestività, migliorando la qualità del software.

Nella **Consegna continua** (*Continuous Delivery*), il codice e le componenti del software vengono rilasciate su un repository locale o remoto definendo release utili per il Versioning Control System.

Nella **Distribuzione continua** (*Continuous Deployment*) le modifiche al codice vengono applicate a ogni build realizzata nella integrazione continua, testate e preparate per il rilascio in produzione in modo automatico. Le modifiche verranno distribuite dopo la fase di creazione di build. In questo modo gli sviluppatori hanno sempre a disposizione una build temporanea pronta e funzionante.

La **Figura 7** mostra il flusso di interazione tra il *continuous integration*, il *continuous delivery* e il *continuous deployment*.

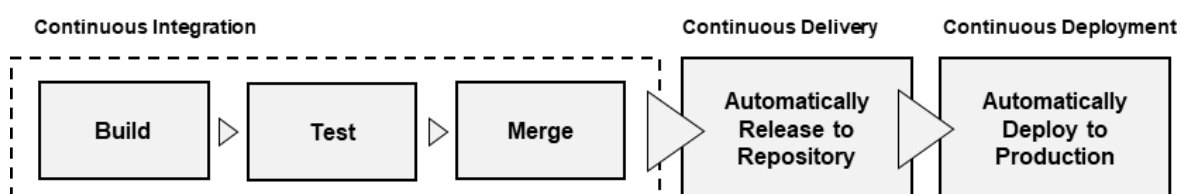


Figura 7. Flusso di Interazione tra Continuous Practices

1.4.1 Differenze tra Scrum e DevOps

Negli ultimi decenni, Scrum e DevOps vengono attuati entrambi ottenendo grandi vantaggi nello sviluppo del software unendo le pratiche vantaggiose DevOps e della metodologia di sviluppo SCRUM. Nella tabella seguente vengono mostrate le differenze di SCRUM e DevOps:

Caratteristica	SCRUM	DevOps
Consegna veloce e frequente	Obiettivo consegne veloci e frequenti	Obiettivo non importante
Creazione e Distribuzione	Sviluppo del software	Focalizzato sul corretto funzionamento e implementazione del

		software.
Specializzazione del team	In un team SCRUM tutti sono uguali, ogni membro può eseguire tutte le attività all'interno del team. Ciò impedisce ritardi e colli di bottiglia.	DevOps definisce i singoli team nello sviluppo e nella gestione del sistema. I dipendenti rimangono all'interno delle loro squadre, ma tutti comunicano spesso.
Comunicazione e documentazione	Gli incontri quotidiani e informali sono al centro di SCRUM. Ogni membro del team deve condividere i progressi, raggiungere gli obiettivi quotidiani e fornire assistenza quando necessario.	Le riunioni DevOps non sono quotidiane, ma richiedono molta documentazione per condividere le proprietà del software con tutti i team.
Dimensione del team	I team SCRUM rimangono piccoli, più piccolo è il team e più velocemente possono muoversi.	DevOps ha molti team che lavorano insieme e ogni team può lavorare secondo un metodo diverso.
Pianificazione	I team SCRUM lavorano in brevi sprint. Gli sprint durano raramente più di un mese.	DevOps mira alla massima affidabilità e si concentra sulla riduzione al minimo delle interruzioni dell'attività.
Ottimizzazione	Non previsto	Questo è il nucleo di DevOps. L'obiettivo generale è minimizzare le interruzioni e massimizzare l'efficienza.

1.4.2 Applicazione dei principi DevOps nei sistemi ML

DevOps, è sicuramente una soluzione utile per la produzione di classici sistemi software che seguono un ciclo di vita come l'individuazione dei requisiti, la progettazione, lo sviluppo, il test, la distribuzione e la manutenzione. Ma negli ultimi anni, con la forte diffusione dei modelli basati su algoritmi di Machine Learning, c'è stata la necessità di supportare anche il processo di sviluppo di applicativi basati su modelli di Machine Learning. È nata dunque una soluzione che interseca l'approccio del Machine Learning stesso con le pratiche DevOps. Questa soluzione prende il nome di MLOps (trattato nel **Cap. 3**) che sulla base delle pratiche messe a punto da DevOps, come il *Continuous Integration* e il *Continuous Deployment* si estendono in un approccio di Machine Learning. Nel *Continuous Integration* vengono trattati dati e modelli, nel *Continuous Deployment*, la distribuzione riguarda l'intera pipeline di addestramento del modello ML in analisi oltre di un solo pacchetto software o servizio. Inoltre, c'è anche il *Continuous Training*, proprietà unica per i sistemi ML, si occupa di riqualificare e servire automaticamente modelli con la necessità di dover riaddestrare nel tempo il modello di Machine Learning.

Capitolo 2: Il Machine Learning

Il Machine Learning è un sottoinsieme dell'Intelligenza Artificiale (AI). È un insieme di meccanismi che permettono a una “macchina intelligente” di migliorare le proprie capacità e prestazioni nel tempo.

La macchina, infatti, impara a svolgere compiti migliorando, tramite “esperienza”, le proprie capacità, le proprie risposte in base a funzioni che vengono implementate nell'algoritmo di Machine Learning. La macchina infatti è in grado di affrontare situazioni e problemi non previsti dalla programmazione iniziale, prendendo dunque, decisioni o compiendo azioni non programmate.

Un esempio esplicativo di funzionamento del Machine Learning è: la macchina apprende la “nuova conoscenza” dall'osservazione dell'ambiente esterno, impara i feedback delle sue azioni e dai suoi errori, oppure apprende dalla base di conoscenza (*un database detto Knowledge base o KB*).

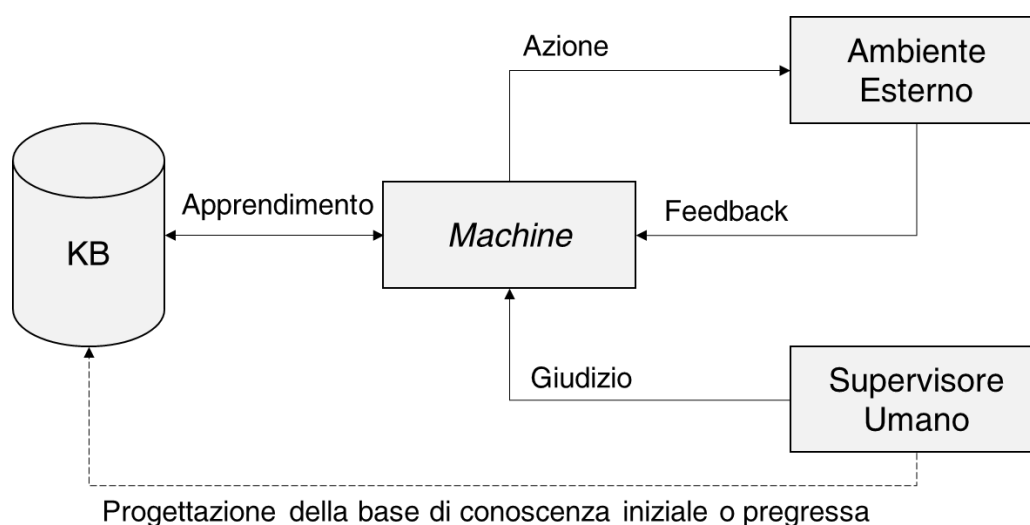


Figura 8. Processo semplificato di apprendimento automatico.

La **Figura 8** descrive il procedimento di apprendimento automatico e si evince che la macchina è al centro dell'apprendimento. Sulla base di dati presenti nella KB, la macchina compie un'azione verso l'ambiente esterno, riceve feedback dall'ambiente esterno, se è un feedback negativo ripete l'azione provando a migliorarla, se è un feedback positivo la impara e la attua. Il supervisore umano invece giudica il comportamento della macchina.

2.1 I diversi apprendimenti di una macchina

Il Machine Learning presenta diverse modalità di apprendimento che differiscono non solo per gli algoritmi utilizzati, ma per lo scopo per cui sono realizzate.

Infatti, in base al tipo di algoritmo utilizzato per permettere l'apprendimento automatico, si possono suddividere tre differenti sistemi di apprendimento: *Apprendimento supervisionato*, *Apprendimento non supervisionato*, *Apprendimento per rinforzo*.

2.1.1 Apprendimento supervisionato (*supervised learning*)

Viene fornito alla macchina una serie di modelli ed esempi, ogni esempio è composto da una serie di valori di input ed è accompagnato da un'etichetta (target) in cui il progettista del sistema ML indica il risultato, pertanto permettono di costruire un vero e proprio database di informazioni e di esperienze. La macchina dunque, quando si trova di fronte ad un problema, non dovrà fare altro che attingere alle esperienze inserite nel proprio sistema, analizzarle ed infine decidere quale risposta dare sulla base di esperienze già codificate. Alla macchina viene fornita una serie di esempi. Un'analogia di macchina con apprendimento supervisionato può essere un bambino che apprende dall'ambiente e dai genitori.

2.1.2 Apprendimento non supervisionato (*unsupervised learning*)

In tale tecnica, la macchina non ha esempi di input-output da testare. Sarà la macchina a classificare tutte le informazioni in proprio possesso, organizzarle ed impararle. La macchina impara dai propri errori e dalla propria esperienza. Un'analogia di macchina con apprendimento non supervisionato può essere un

bambino cresciuto nella giungla che poi pian piano riesce ad imparare a vivere.

2.1.3 Apprendimento per rinforzo (*reinforcement learning*)

Tale tecnica, è molto complessa, richiede una macchina in grado di migliorare il proprio apprendimento e comprendere le caratteristiche dell'ambiente circostante. Si basa sul meccanismo dei premi e delle ricompense. La macchina ha una "funzione di rinforzo" associata a un obiettivo da raggiungere che gli permette di valutare i feedback delle azioni. Infatti, l'obiettivo della macchina è quello di massimizzare il premio. Più il premio è alto, più la macchina è vicina alla soluzione. Un'analogia di macchina con apprendimento per rinforzo può essere un bambino in grado di apprendere l'azione che compie il genitore, se il bambino prova ad imitare il genitore e se l'azione che compie è giusta allora imparerà fino a quando non riuscirà ad apprendere per bene.

Esistono inoltre diversi approcci pratici per l'applicazione di algoritmi di machine learning tra cui *l'albero delle decisioni*, un grafo ad albero aciclico orientato come modello predittivo per indicare delle decisioni, le *reti neurali artificiali*, *algoritmi genetici*, analisi dei gruppi, ovvero un processo di raggruppamento di oggetti in modo che quelli inseriti nello stesso gruppo, chiamato *cluster* siano più simili tra loro rispetto a quelli in altri gruppi, reti di bayes e programmazione logica induttiva.

Nella seguente trattazione si analizza un caso di modellazione predittiva che applica sia l'apprendimento supervisionato e sia l'apprendimento non supervisionato.

2.2 La modellazione predittiva e le sue tecniche

La modellazione predittiva è una tecnica del Machine Learning in grado di "prevedere" che cosa potrebbe accadere in futuro.

Tale tecnica consiste nella capacità di apprendere modelli partendo dai dati a disposizione. I modelli appresi hanno la capacità di agire sui nuovi dati a disposizione fornendo diverse "capacità":

- *Classificazione*: capacità di definire un soggetto ad una specifica classe (es.

classificare un cliente di una banca in base alle informazioni personali ed economiche a disposizione);

- *Regressione*: definire relazioni funzionali tra le variabili considerate (es. definire la relazione tra l'età di una persona e il suo potenziale interesse verso una specifica campagna pubblicitaria);
- *Clustering*: raggruppare dati in insiemi omogenei (es. creare gruppi di utenti in base ai loro consumi e proporre offerte mirate);
- *Predizione*: fare predizioni o previsioni del comportamento futuro delle variabili considerate (es. fare una predizione di una possibile infezione del paziente ad un particolare virus come *l'esempio trattato nella seguente trattazione*);
- *Rilevamento di anomalie*: identificare cambiamenti nei dati (es. analisi automatica della qualità dei pezzi prodotti in un'azienda manifatturiera);
- *Adattamento di modelli*: permettere ai modelli appresi di evolvere nel tempo in base alla necessità (es. un utente può cambiare i propri interessi con l'età o le stagioni).

2.3 Quando può essere utile il Machine Learning?

La scelta di utilizzare il Machine Learning per la risoluzione di un problema ricade quando il sistema desiderato presenta almeno una delle seguenti esigenze:

- Il Sistema deve adattarsi all'ambiente in cui opera in maniera automatica;
- Il Sistema deve migliorare le proprie prestazioni rispetto allo svolgimento di un particolare compito;
- Il Sistema deve scoprire modelli, regolarità e nuove informazioni a partire da dati empirici.

Nella **Figura 9** si riporta una classificazione di alcuni campi applicativi in cui un'elevata complessità del problema indica generalmente un alto valore di business della soluzione.

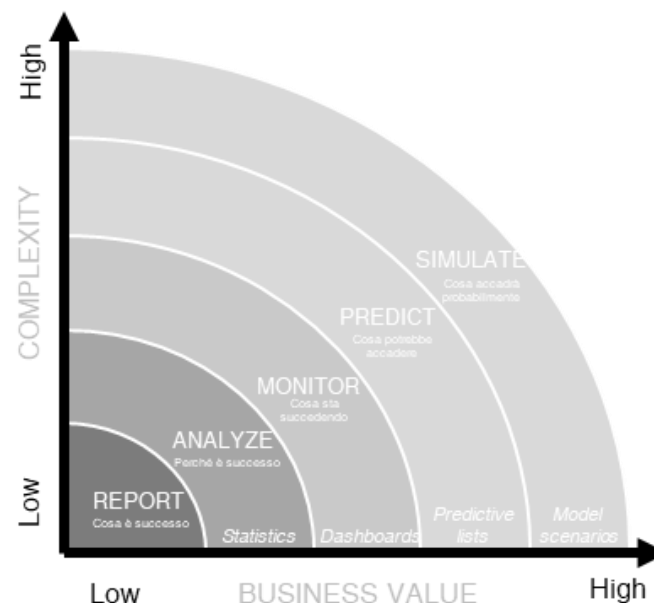


Figura 9. *Classificazione di campi applicativi (business value – complexity).*

2.4 Esempi di applicazioni di apprendimento automatico

Nel 2021 l'apprendimento automatico ha raggiunto un livello di quotidianità elevatissimo, basti pensare al classico *riconoscimento vocale* di cui sono dotati molti smartphone che permettono di attivare comandi tramite la propria voce o gestire elettrodomestici o dispositivi elettronici casalinghi (domotica). Esempio che a volte spaventa l'utilizzatore è la *pubblicità tracciante*, ovvero a seconda dell'utente di internet, vengono effettuate proposte pubblicitarie strettamente collegate agli interessi dell'utente stesso, le cui necessità e gusti vengono riconosciuti tramite l'analisi delle ricerche maggiormente effettuate in rete. La gestione del traffico ferroviario, cittadino, aereo, navale, può essere supportato da sistemi di Machine Learning che supportano l'*Intelligent Transportation System*. Altro esempio, molto discusso è la *guida autonoma* o veicoli senza pilota, grazie al quale mediante sensori, telecamere e sistemi di localizzazione installate nel veicolo sono in grado di realizzare un sistema di apprendimento per rinforzo, che permette di prendere decisioni su eventuali frenate, sterzate e stile di guida. Date le grandi potenzialità della modellazione predittiva, essa viene utilizzata anche per la *manutenzione predittiva* in modo tale da poter reagire in maniera tempestiva ad eventuali guasti di un sistema, rilevamento frodi, analisi di qualità nella gestione del magazzino e analisi delle reti sociali.

2.5 Evoluzione futura del Machine Learning

Come già detto nel precedente paragrafo, il Machine Learning e la branca di appartenenza: l'Intelligenza Artificiale, sono ormai attuali in vastissimi sistemi tecnologici e ancora molto deve essere fatto per poter perfezionare una serie di elementi, di algoritmi e strutture tecniche. Un possibile sviluppo, attualmente in atto, è l'approccio del Machine Learning nella medicina, nella capacità di una macchina di curare un paziente, diagnosticare attraverso immagini, gestione, ricerca e sviluppo, che grazie il supporto della conoscenza scientifica si è in grado di abbattere inefficienze dovute a errori umani.

Altra disciplina in evoluzione è sicuramente la scienza dei dati (*Data Science*) che grazie al supporto del Machine Learning sta diventando fondamentale per risolvere problemi complessi del mondo reale. Pertanto, molte aziende stanno investendo nei propri team di data science e nelle capacità di Machine Learning per sviluppare modelli predittivi.

Tale evoluzione, implica alla realizzazione di sistemi informatici che inglobano un modello di Machine Learning e pertanto servirà trovare un modo per gestire il modello ML e renderlo parte attiva del sistema informatico introducendo un'interazione tra modello e applicativo.

Capitolo 3: Il Machine Learning Operations (MLOps)

Come detto nel paragrafo 2.5, si è in una fase in cui quasi tutte le organizzazioni stanno cercando di incorporare il Machine Learning nei loro prodotti e processi aziendali. Infatti, i dati sono diventati linfa vitale di tutte le organizzazioni giacché le decisioni basate sui dati fanno sempre più la differenza con la concorrenza mondiale in ambito aziendale. La creazione di sistemi di Machine Learning aggiunge e riforma alcuni principi dell'SDLC (*Software Development Life Cycle*), dando origine a una nuova disciplina ingegneristica chiamata **Machine Learning Operations** o **MLOps**. *Google Trends* dimostra l'interesse della nuova pratica di sviluppo MLOps negli ultimi 5 anni:



Figura 10. Tendenze Google dell'argomento MLOps cercato negli ultimi 5 anni. Si nota un incremento nel 2019 ed un picco massimo nel 2021.

Il precursore della pratica MLOps è la Cina che ha registrato il maggior numero di ricerche durante il periodo specificato. L'Italia, non presente nella top 5, è 23° con il solo 3%, sintomo che sia ancora in fase di ricerca e sviluppo. Nella **Figura 11** è mostrata la top 5 completa:



Figura 11. Tendenze Google per area geografica dell'argomento MLOps cercato negli ultimi 5 anni.

3.1 Cos'è MLOps

MLOps è una disciplina ingegneristica che mira a unificare lo sviluppo dei sistemi Machine Learning (*dev*) e l'implementazione dei sistemi Machine Learning (*ops*) al fine di standardizzare e semplificare l'erogazione continua di modelli ad alte prestazioni in produzione. L'obiettivo di MLOps è quello di sostenere l'automazione e il monitoraggio in tutte le fasi della costruzione del sistema Machine Learning *"integrato"* ad un prodotto software e farlo funzionare continuamente in produzione, includendo le fasi di integrazione, test, rilascio e distribuzione del software.

Dunque le risorse di Machine Learning vengono trattate in modo coerente con tutte le altre risorse software con l'obiettivo di accelerare la consegna rapida di software intelligente. MLOps coinvolge diversi team di un'organizzazione, come il *team di sviluppo aziendale* o il *team di prodotto* che definiscono obiettivi aziendali definendo le Key Performance Indicator (*KPI*), il team di *Ingegneri dei dati* che acquisiscono e preparano i dati, i *Data Science* che progettano soluzioni Machine Learning e sviluppano modelli e i *team IT* o *Ingegneri DevOps* che configurano la distribuzione, monitoraggio insieme ai Data Scientist.

È aggiunta la figura del *designer* che è dedicato alla comprensione dei dati e alla progettazione del software basato sul Machine Learning. Il Developer Team è dedicato alla verifica dell'applicabilità del modello Machine Learning. L'Operations Team invece è dedicato alla produzione utilizzando le pratiche DevOps.

Un'estensione della **Figura 6** è mostrata nella **Figura 12**:

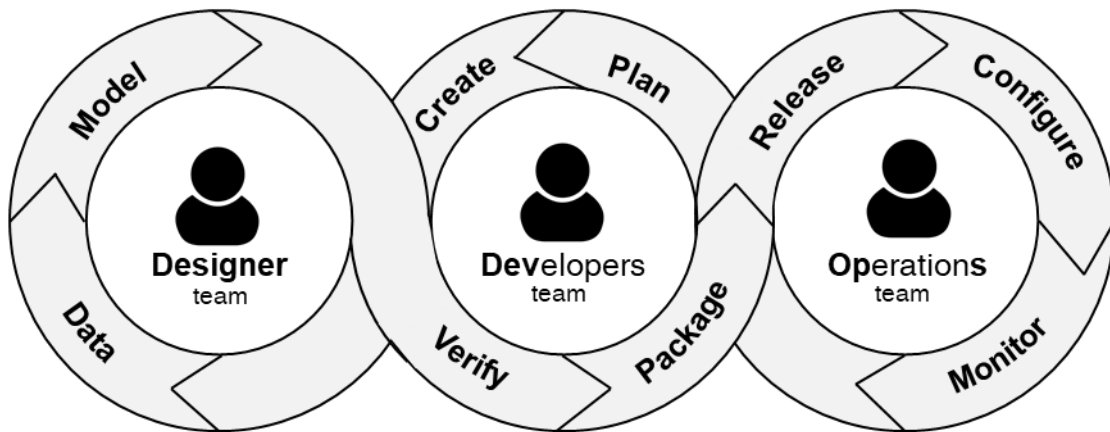


Figura 12: MLOps estende lo schema della Figura 6 aggiungendo la figura del Designer.

Dalla **Figura 12**, si individuano dunque le diverse fasi che compongono lo sviluppo MLOps:

- La prima fase è dedicata alla comprensione degli obiettivi, dei dati e alla progettazione del software basato sul Machine Learning. Si identificano i potenziali utenti, si progettano soluzioni di Machine Learning e si valutano gli sviluppi del progetto. Si definiscono i casi d'uso ML e si progetta mirando a ispezionare i dati disponibili che saranno necessari per addestrare il modello. Si estraggono i requisiti funzionali e non funzionali del modello ML. Sulla base dei requisiti definiti si progetta l'architettura dell'applicazione ML.
- La seconda fase è dedicata alla sperimentazione e allo sviluppo ML. Si eseguono in modo iterativo i diversi passaggi come identificare, perfezionare l'algoritmo ML. L'obiettivo di questa fase è fornire un modello ML di qualità stabile che verrà eseguito in produzione.
- La terza fase è dedicata a fornire il modello ML precedentemente sviluppato in produzione utilizzando pratiche DevOps consolidate come test, controllo delle versioni, distribuzione continua e il monitoraggio.

Tutte e tre le fasi sono interconnesse e si influenzano a vicenda. Ad esempio, la decisione progettuale durante la fase di progettazione si propagherà nella fase di sperimentazione e influenzerà infine le opzioni di implementazione durante la fase operativa finale. Tuttavia, molte ricerche mostrano come lo sviluppo di modelli Machine Learning comprendono solo una parte molto piccola dell'intero processo.

Esistono infatti molti altri processi, configurazioni e strumenti che verranno integrati nel sistema.

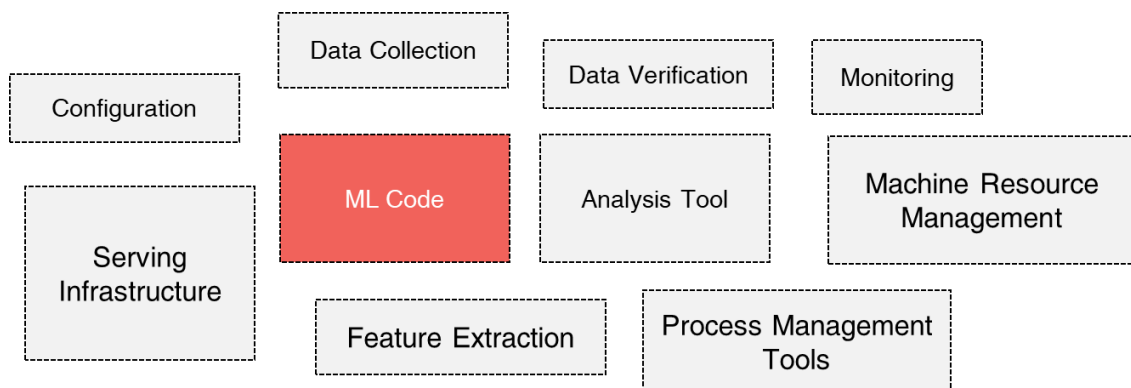


Figura 13. La realizzazione del ML code coinvolge tutti i team interessati, dai Data Scientist ai DevOps e agli Ingegneri ML.

3.2 I problemi che risolve MLOps

La gestione di tali sistemi non è un compito facile e ci sono numerosi colli di bottiglia e problematiche che devono essere risolti. Di seguito sono elencate le principali sfide per risolvere queste problematiche:

- Carenza di Data Scientist che siano bravi sia a sviluppare e sia a distribuire applicazioni web scalabili,
- Continua modifica degli obiettivi del modello e visto che esistono molte dipendenze con i dati che cambiano continuamente è difficile stare al passo con la formazione continua del modello stesso e degli obiettivi aziendali in evoluzione,
- Divario di comunicazione tra team tecnici e aziendali con un linguaggio comune difficile da trovare per collaborare.
- Valutazione del rischio.

3.3 I principi di MLOps

Il Machine Learning Operations segue una serie di principi: il versioning, la riproducibilità, il test, l'automazione, il feature store, il continuous integration, il continuous distribution e il continuous monitoring.

3.3.1 Il Versioning

L'obiettivo del Versioning è monitorare i modelli e i set di dati (*Dataset*) associandoli con una cifra alfanumerica individuando le modifiche o aggiornamenti che vengono effettuate su di essi. I modelli ML possono essere riaddestrati in base ai nuovi dati di addestramento visto che possono degradarsi nel tempo, pertanto c'è la necessità di aggiornarli altrimenti possono essere soggetti ad attacchi e dunque richiedono una revisione. Gli strumenti di Versioning più diffusi sono:

Tool	Licenza
IBM Watson ML	Proprietario
DVC	Open-source
MLflow	Open-source
Git LFS	Open-source

Grazie al “versioning” è possibile ripristinare e confrontare rapidamente i modelli a una versione di pubblicazione precedente in caso di problematiche nelle versioni successive.

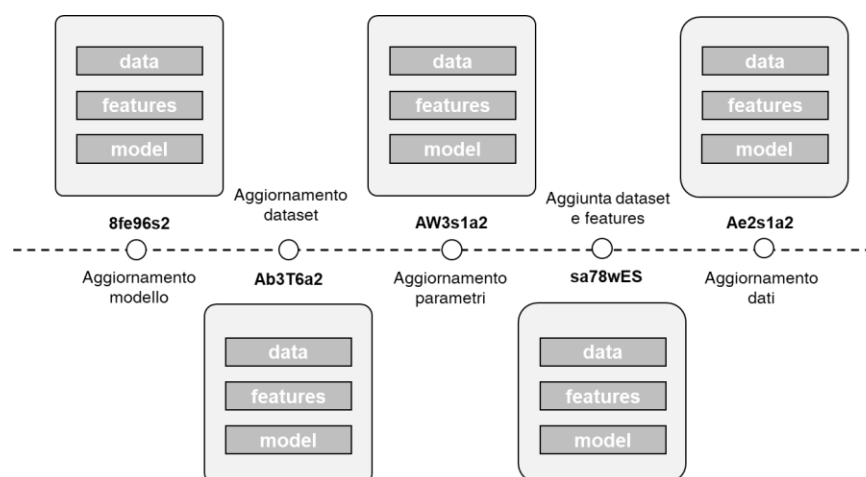


Figura 14. Versioning dei dati nel tempo.

3.3.2 La Riproducibilità

La riproducibilità in un flusso di lavoro di Machine Learning determina che ogni fase dell'elaborazione dei dati, dell'addestramento del modello ML e della distribuzione

del modello ML dovrebbe produrre risultati identici con lo stesso input.

3.3.3 Il Test

Si distinguono tre ambiti per i test nei sistemi Machine Learning:

- Test per funzionalità e dati:
 - Convalida dei dati,
 - Verifica delle conformità dei dati alle policy (es. GDPR),
 - Verifica dell'importanza delle funzionalità utile a capire se aggiungono un potere predittivo.
- Test per lo sviluppo di modelli:
 - Verifica dell'obsolescenza del modello, se il modello addestrato non include dati aggiornati viene considerato obsoleto e possono influenzare la qualità della previsione del sistema ML.
 - Convalida delle prestazioni di un modello,
 - Verifica della correttezza per le prestazioni del modello ML,
 - Unit test convenzionali per qualsiasi creazione di funzionalità,
 - Verifica degli algoritmi se prendono decisioni allineate agli obiettivi definiti.
- Test per l'infrastruttura ML:
 - Verifica della correttezza algoritmica,
 - Verifica di integrazione dell'intera pipeline ML.

I test vengono poi classificati in *“misura della prontezza complessiva del sistema ML”* per la produzione. Il test ML viene calcolato come segue:

- Per ogni test viene assegnato “mezzo punto” per l'esecuzione manuale della prova.
- Viene assegnato un “punto pieno” se esiste un sistema in atto per eseguire automaticamente il test.
- Sommare il punteggio ottenuto dai test per le funzionalità dei dati, per lo sviluppo dei modelli e per l'infrastruttura ML.

Sulla base del risultato ottenuto è possibile ragionare sulla prontezza del sistema ML per la produzione.

La tabella seguente fornisce gli intervalli di interpretazione:

Punti	Descrizione
0	Progetto di ricerca
(0-1)	Progetto non del tutto testato dunque possibilità di lacune e inaffidabilità del sistema
(1-2)	Progetto testato ma che potrebbero comunque essere necessari ulteriori investimenti
(2-3)	Progetto ragionevolmente testato ma necessita di monitoraggio automatizzato
(3-5)	Progetto con forte livello di test con monitoraggio automatizzato
>5	Progetto con eccezionale livello di test e monitoraggio automatizzato

3.3.4 L' Automazione

L'obiettivo di un team MLOps è automatizzare l'implementazione di modelli Machine Learning nel sistema software principale. Ciò significa automatizzare i passaggi completi del flusso di lavoro del Machine Learning senza alcun intervento manuale.

Un esempio di attuazione dell'automazione è l'adattamento del trigger per l'addestramento e la distribuzione automatica del modello che possono essere eventi di calendario, messaggistica, eventi di monitoraggio, modifiche ai dati.

Si hanno tre livelli di automazione:

- Processo manuale, eseguito all'inizio dell'implementazione del Machine Learning. Ogni fase della pipeline del Machine Learning sarà manuale, come la preparazione e la convalida dei dati, l'addestramento e il test del modello.
- Automazione della Pipeline ML, dove viene eseguito in maniera automatica il training del modello. Ogni volta che sono disponibili nuovi dati, viene attivato il processo di riaddestramento del modello. Questo processo include anche le fasi di convalida dei dati e del modello.
- Automazione della Pipeline CI/CD, dove viene introdotto un sistema CI/CD per eseguire distribuzioni di modelli ML veloci e affidabili in produzione. Infatti, si crea, si testa e si distribuiscono automaticamente i componenti dati, modello ML e pipeline di training ML.

La figura seguente mostra la pipeline ML automatizzata con routine CI/CD

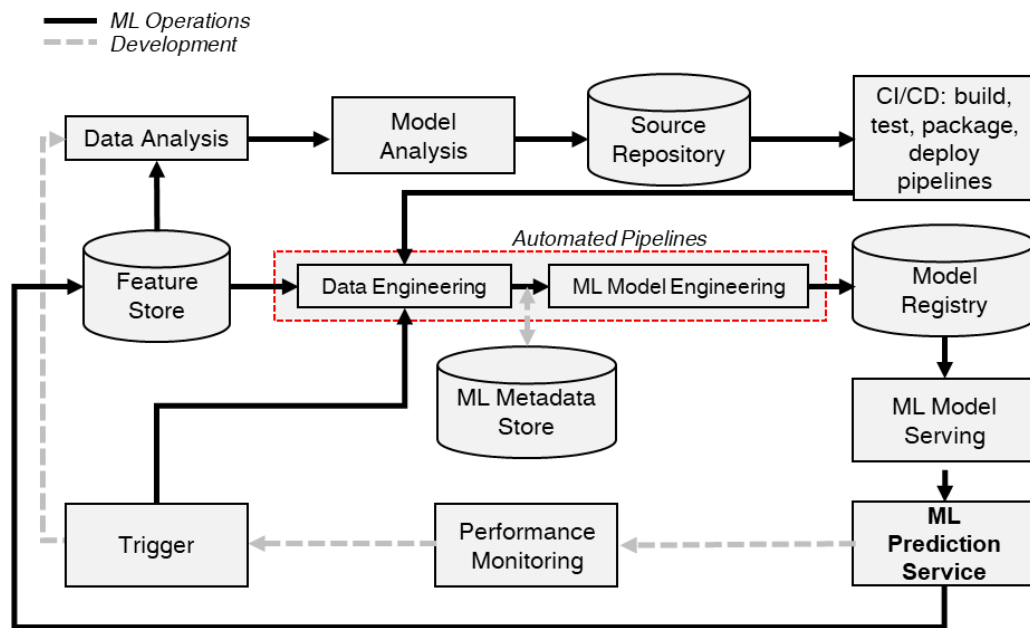


Figura 15. Pipeline di automazione del modello ML

La pipeline della **Figura 15** è descritta nella tabella seguente:

Fase MLOps	Risultato dell'esecuzione della fase
Model Analysis	Estrazione dei dati, convalida dei dati, preparazione dei dati, addestramento del modello, valutazione del modello, test del modello.
Continuous Integration e Distribution	Creazione codice sorgente ed esecuzione test continua, distribuzione dell'implementazione del modello.
Trigger automatico	Addestramento del modello ad ogni trigger, (la pipeline viene eseguita automaticamente in produzione)
Modello di consegna continua	Servizio di previsione del modello distribuito
Monitoraggio (raccolta dati sulle prestazioni del modello su dati in tempo reale)	Trigger per eseguire la pipeline o per avviare un nuovo ciclo di esperimento

Gli strumenti per l'automazione della pipeline di machine learning sono:

Tool	Licenza	Osservazione
DVC	Open-source	DVC può essere utilizzato per creare pipeline che possono essere automatizzate e riprodotte.
Tensorflow	Open-source	Integrazione delle pipeline con Google
Kubeflow	Open-source	Kubeflow crea pipeline ed esperimenti automatizzati per l'apprendimento automatico, in grado di monitorare anche i modelli.
MLflow	Open-source	Piattaforma per il ciclo di vita del ML.

3.3.5 Il Feature Store

L'obiettivo di un Feature Store è elaborare i dati da varie origini contemporaneamente e trasformarli in feature, che verranno utilizzate dalla pipeline di training del modello e dalla pubblicazione del modello.

Le caratteristiche principali del featurizing store sono la possibilità di condividere le feature da parte di più team di Data Scientist che lavorano contemporaneamente, la creazione di una pipeline di elaborazione automatizzata affidabile di grandi quantità di dati, la possibilità di utilizzare diverse origini di dati come i data lake e data warehouse (grandi blocchi di dati che sono stati archiviati per essere utilizzati dai modelli e non sono aggiornati in tempo reale) e streaming di dati (dati online che provengono costantemente da fonti come eventi registrati su un sistema), tutto in una volta.

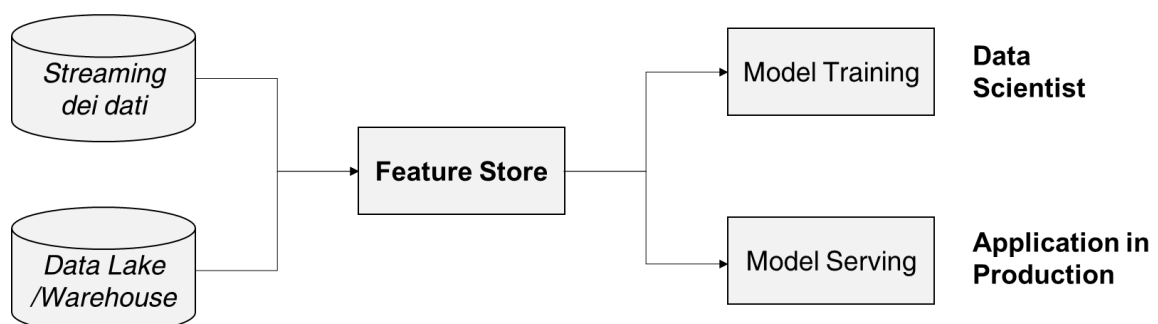


Figura 16. Feature Store

3.3.6 Il Continuous Integration

L'integrazione continua in ML significa che ogni volta che un codice o un dato viene aggiornato, la pipeline ML viene rieseguita in modo che tutto sia versionato, riproducibile e condivisibile tra progetti e il team. Ogni ripetizione può consistere in training o generazione di nuovi report, facilitando il confronto tra le altre versioni in produzione. Alcuni esempi di un flusso di lavoro CI:

- Esecuzione e controllo delle versioni e della valutazione per ogni commit nel repository.
- Esecuzione e confronto degli esperimenti per ogni richiesta pull con un determinato ramo.

Un'estensione del CI è il **Continuous Delivery** in quanto consegna automaticamente tutte le modifiche al codice in un ambiente di test e/o di produzione dopo la fase di compilazione. Ciò significa che oltre ai test automatici definiti nel CI, si ha un processo di rilascio automatizzato distribuendo l'applicazione testata e funzionante.

3.3.7 Il Continuous Distribution

La distribuzione continua è un metodo per automatizzare la distribuzione della nuova versione in produzione. Questa pratica semplifica la ricezione del feedback degli utenti, poiché le modifiche sono più rapide e costanti, nonché nuovi dati per la riqualificazione o nuovi modelli. Un esempio di flusso di lavoro di Continuous Distribution è il test dell'output del modello in base a un input noto.

La distribuzione del modello Machine Learning fornisce la registrazione, il monitoraggio e le notifiche per garantire che i modelli del Machine Learning siano stabili per essere distribuiti a destinazione.

3.3.8 Il Continuous Monitoring

Una volta che il modello Machine Learning è stato distribuito, deve essere monitorato per assicurare che il modello funzioni come previsto. È un processo continuo, quindi è importante identificare gli elementi per il monitoraggio e creare

una strategia per il monitoraggio del modello prima di raggiungere la produzione.

È necessario dunque monitorare:

- Le modifiche durante l'intera pipeline generando una notifica in caso di modifica dei dati,
- se i dati non corrispondono allo schema che è stato specificato nella fase di addestramento inviare un avviso,
- le prestazioni computazionali di un sistema ML,
- il degrado della qualità predittiva del modello ML sui dati forniti.

Il seguente grafico mostra che lo stato di salute di un sistema di Machine Learning si basa su caratteristiche nascoste che non sono facili da monitorare.

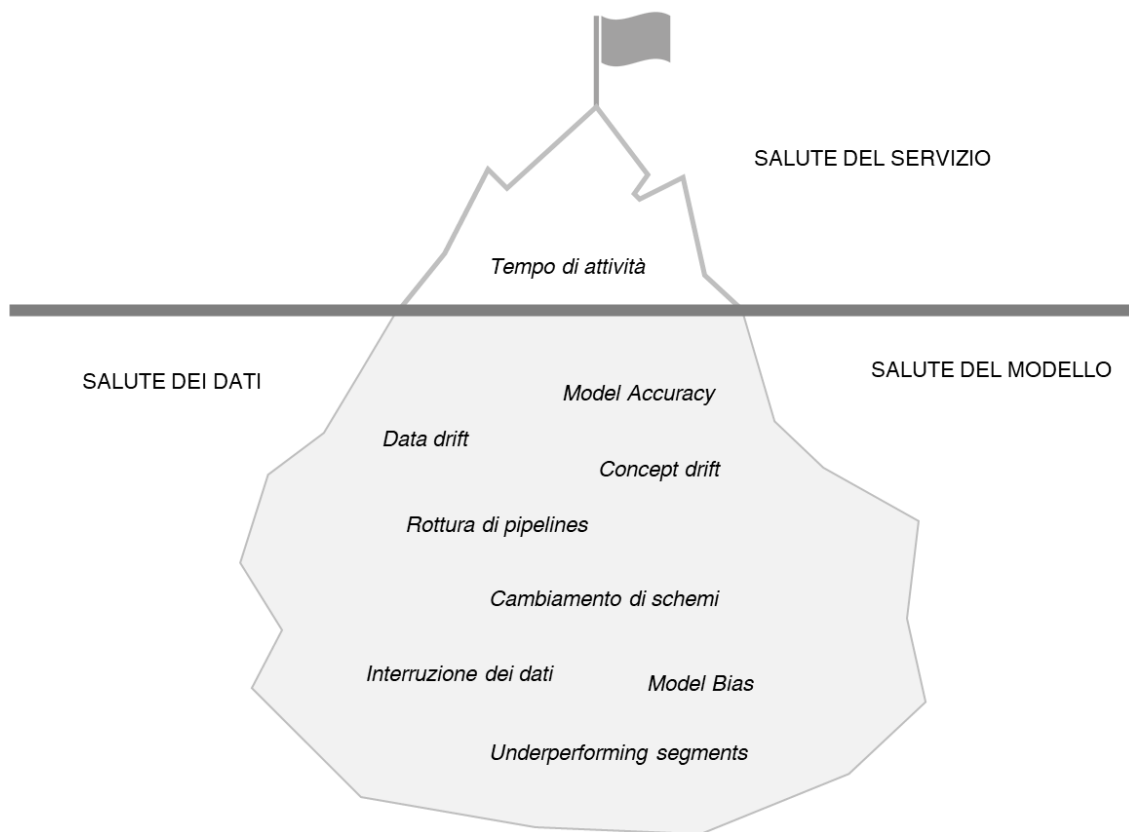


Figura 17. Analogia salute di un sistema ML con l'iceberg

L' analogia dimostra come la salute del servizio dipende dal tempo di attività mentre la salute dei dati e salute del modello discendono da tanti fattori difficili da monitorare.

In definitiva, si riepilogano in una tabella i principi del Machine Learning Operations per la creazione del software includendo tre livelli in cui possono verificarsi modifiche sui Dati, sul Modello ML e sul Codice.

Principi MLOps	Dati	Modello ML	Codice
Versioning	1) Pipeline per la preparazione dei dati 2) Archivio delle caratteristiche 3) Dataset 4) Metadati	1) Pipeline di addestramento del modello ML 2) Modello ML 3) Monitoraggio dell'esperimento	1) Codice applicazione 2) Configurazione
Test	1) Convalida dei dati (rilevamento degli errori) 2) Test dell'unità di creazione delle funzionalità	1) Test delle specifiche del modello 2) Il modello ML viene convalidato prima di essere reso operativo 3) Test di obsolescenza del modello ML 4) Test della pertinenza e della correttezza del modello ML 5) Test dei requisiti non funzionali (sicurezza, correttezza, interpretabilità)	1) Unit test 2) Test di integrazione della pipeline end-to-end

Automazione	1) Trasformazione dei dati	1) Pipeline di addestramento del modello ML 2) Selezione di iperparametri/parametri	1) Distribuzione del modello ML con CI/CD 2) Build dell'applicazione
Riproducibilità	1) Dati di backup 2) Versionamento dei dati 3) Estrazione dei metadati 4) Versionamento della progettazione delle funzionalità	1) L'ordine delle funzionalità è lo stesso 2) Lo pseudo-codice del modello è documentato	1) Tutte le versioni delle dipendenze in development e production sono identiche 2) Riproduzione dei risultati fornendo immagini di contenitori o macchine virtuali
Distribuzione	1) Il Feature Store	1) Containerizzazione dello stack ML 2) API REST 3) On-premise, cloud o edge.	1) On-premise cloud o edge
Monitoraggio	1) Modifiche alla distribuzione dei dati (formazione vs. dati di pubblicazione) 2) Funzionalità di formazione vs. offerta	1) Decadimento del modello ML 2) Stabilità numerica 3) Prestazioni computazionali del modello ML	1) Qualità predittiva dell'applicazioni sulla fornitura dei dati

Insieme ai principi MLOps, seguono anche le migliori pratiche del progetto ML:

Migliori pratiche MLOps	Dati	Modello ML	Codice
Documentazione	1) Fonti dei dati 2) Decisioni, come/dove ottenere i dati 3) Metodi di etichettatura	1) Criteri di selezione del modello 2) Progettazione degli esperimenti 3) Pseudocodice del modello	1) Processo di distribuzione 2) Come eseguire localmente
Struttura del progetto	1) Cartella dati per i dati grezzi ed elaborati 2) Una cartella per la pipeline di ingegneria dei dati 3) Cartella di prova per i metodi di ingegneria dei dati	1) Una cartella che contiene il modello addestrato 2) Una cartella per i notebook 3) Una cartella per l'ingegneria delle funzionalità 4) Una cartella per l'ingegneria del modello ML	1) Una cartella per gli script bash/shell 2) Una cartella per i test 3) Una cartella per i file di distribuzione (es. file Docker)

3.4 Implementazioni MLOps

Esistono 3 modi per implementare MLOps:

- *MLOps livello 0* (processo manuale), implementato quando i modelli ML vengono modificati o addestrati raramente. Ogni passaggio è manuale, dall'analisi dei dati fino alla convalida. C'è disconnessione tra ML e Operations. Le iterazioni di rilascio non sono frequenti e una nuova versione

del modello è distribuita solo un paio di volte l'anno. Non presenta le continuous practice (CI/CD). Mancanza di monitoraggio attivo delle prestazioni del modello nel tempo. I modelli non riescono ad adattarsi ai cambiamenti nelle dinamiche dell'ambiente o ai cambiamenti nei dati che descrivono l'ambiente.

- *MLOps livello 1* (automazione della pipeline ML), implementato per soluzioni che operano in un ambiente in continua evoluzione e devono affrontare in modo proattivo i cambiamenti nel comportamento degli utenti/clienti, nelle tariffe e in altri indicatori. A differenza del livello 0, viene adottato il continuous training, ovvero l'addestramento automatico utilizzando dati aggiornati basati su trigger di pipeline. C'è simmetria tra implementazione operativa (in produzione) e sperimentale utile per testare e modificare il modello o l'applicazione e calcolarne le differenze. Ulteriori componenti aggiuntivi sono la convalida dei dati e dei modelli, il feature store, la gestione dei metadati e il trigger di pipeline ML su richiesta, a valle di un evento esterno o in un programma.
- *MLOps livello 2* (automazione pipeline CI/CD), implementato per soluzione che devono riqualificare i loro modelli ogni giorno e ridistribuirli su migliaia di server contemporaneamente. Le caratteristiche del livello 2 sono lo sviluppo e sperimentazione, ovvero la prova iterativa di nuovi algoritmi ML e nuovi modelli, l'integrazione continua della pipeline e quindi creazione di codice sorgente ed esecuzione di vari test, la distribuzione continua della pipeline e del modello, trigger automatico e monitoraggio raccogliendo prestazione basate su dati in tempo reale.

3.5 Applicazioni MLOps

Sulla base delle esperienze progettuali maturate dalla “Reply³” è possibile riconoscere l'utilità di MLOps sul versionamento dei dati, versionamento del modello e il monitoring delle performance.

Viene descritto una problematica incorsa in un progetto di un modello in grado di

³ Esperienze prelevate dal seguente link: <https://www.reply.com/machine-learning-reply/it/mlops>

predire i clienti che avrebbero disdetto il loro contratto a distanza di un mese. Si è notato che dopo aver effettuato un riaddestramento del modello c'è stato un forte calo nelle performance. Per capire l'origine del problema si è deciso di seguire una pratica dell'MLOps ovvero confrontare le versioni dei dataset utilizzati e analizzare le possibili anomalie presenti. Infatti, si è scoperto che alcune feature numeriche contenevano dei valori di testo, e che il problema era sorto a valle di una modifica apportata alla fase di preparazione del dato. Un'altra problematica risolta dalle pratiche MLOps è il versionamento del modello e il confronto tra le versioni del modello. Nel corso di un progetto di un modello in grado di classificare i clienti di un sito web in base ai loro comportamenti sul sito, si è rilevato un problema, ovvero il modello ha iniziato a classificare tutti gli utenti come 'principianti', la risoluzione è stata confrontare le versioni del modello, identificare l'origine del problema e risolvere quel problema, in questo caso il problema era nato poiché nel codice sorgente c'era una feature avanzata che l'utente non eseguiva mai. Sempre in ambito e-commerce è possibile dimostrare un'ulteriore applicazione dell'MLOps, attraverso un algoritmo in grado di predire gli acquisti dei propri clienti in base al loro comportamento su una piattaforma e-commerce. L'accuratezza di tale algoritmo però era alta solo in un determinato arco della giornata. Dopo aver analizzato il set di dati utilizzato per l'addestramento, si è scoperto che il modello era stato addestrato solo per il comportamento dell'utente in un determinato arco della giornata. Effettuando un riaddestramento con un set di dati rappresentativo del comportamento dell'utente lungo l'arco dell'intera giornata è stato possibile aumentare le performance del modello considerevolmente.

Un caso d'uso di notevole importanza dell'MLOps è in ambito *Healthcare (sanità)*. Infatti, con un'enorme quantità di dati medici generati, le organizzazioni sanitarie possono sfruttare questa mole di conoscenza per fornire risultati migliori con maggiore precisione. Soprattutto in questo periodo di *pandemia mondiale del virus SARS-CoV-2 (COVID-19)*. L'Intelligenza Artificiale può aiutare le organizzazioni sanitarie a fornire le previsioni e i risultati più semplici per i pazienti, riducendo i costi e personalizzando in modo significativo l'assistenza ai pazienti. Il caso trattato nell'esempio di applicazione MLOps risponde proprio a questa esigenza attraverso una Web App facilmente accessibile dagli utenti realizzata dall'autore.

Capitolo 4: Esempio di Applicazione MLOps

In questo capitolo viene trattato un *Proof of Concept*⁴ delle pratiche MLOps applicato ad un semplice applicativo basato su un modello di Machine Learning. Per semplicità, si segue un processo più esaustivo di quello spiegato nei precedenti paragrafi. Infatti, verranno trattati gli obiettivi del modello di Machine Learning, verrà scelto un dataset per l'estrazione dei dati, verranno preparati i dati e poi addestrati. Successivamente, la pipeline verrà automatizzata e distribuita secondo alcuni principi e tools ed infine il monitoraggio con l'obiettivo di ottimizzare il modello e renderlo sempre disponibile e aggiornato ad effettuare predizioni sempre più precise.

L'ambiente di lavoro dell'autore è il seguente:

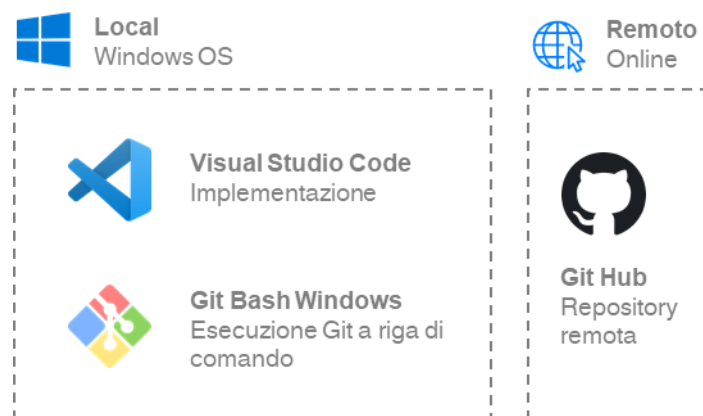







Figura 18. Ambiente di lavoro dell'autore

⁴ Il *Proof of Concept* è una piccola realizzazione di bozza progettuale per tracciare un progetto, testare l'idea o l'ipotesi di progetto al fine di dimostrarne la fattibilità.

Si è lavorato in locale sul sistema operativo Windows 10 con ambiente di sviluppo *Visual Studio Code* ed esecuzione dei comandi *Git* con *GitBash Windows*. In remoto, si è lavorato con *GitHub*. Gli strumenti principali che verranno utilizzati sono mostrati nella tabella seguente:

Tools		Utilizzo
	GitHub	Repository remota e versionamento del codice
	GitHub Actions	Automazione CI e CD
	DVC	Versionamento di dati e modelli, automazione della pipeline
	CML	Continuous Machine Learning
	Heroku	Distribuzione web app

4.1 Obiettivi del modello Machine Learning

L'obiettivo del modello da realizzare è quello di predire una certa probabilità di infezione al COVID-19 in base ai sintomi o alle malattie che il paziente manifesta e guidarlo a come comportarsi in caso di alta probabilità di infezione. Ad esempio, se il sistema predice che il paziente X abbia una probabilità di infezione tra il 75% e il 100% allora verrà consigliato di consultare immediatamente un operatore sanitario e di rimanere nel suo domicilio. Se il sistema predice che il paziente Y abbia una probabilità di infezione tra il 50% e il 75% allora verrà consigliato di auto medicarsi in casa e se vuole può chiamare il medico di fiducia per ulteriori consultazioni utili per sollevarsi dai sintomi. Infine, il modello verrà 'inglobato' in una semplice Web App per renderla utilizzabile dall'utente.

4.2 Il dataset scelto

Scelto l'obiettivo, si iniziano a cercare i dati di input appropriati e i tipi di modelli da provare per quel tipo di dati. Si è trovato un *dataset*⁵ che contiene una serie di possibili sintomi legati al COVID-19 e il risultato del tampone o test sierologico per capire se una X persona, con quei sintomi, ha il COVID oppure no (target).

⁵ La fonte del dataset è: <https://www.kaggle.com/midouazerty/symptoms-covid-19-using-7-machine-learning-98/data> ed è di tipo statico, ovvero un file.csv

I sintomi (le colonne) presenti nel dataset sono:

Sintomo	Si (presenza del sintomo) %	No (non presenza del sintomo) %
Breathing Problem (Problema di respirazione)	67%	33%
Fever (Febbre)	79%	21%
Dry Cough (Tosse secca)	79%	21%
Sore Throat (Mal di gola)	73%	27%
Running Nose (Naso che cola)	54%	46%
Asthma (Asma)	46%	54%
Chronic Lung Disease (Malattia polmonare cronica)	47%	53%
Headache (Mal di Testa)	50%	50%
Heart Disease (Malattia cardiaca)	46%	54%
Diabetes (Diabete)	48%	52%
Hyper Tension (Ipertensione)	49%	51%
Fatigue (Fatica)	47%	53%
Gastrointestinal (Gastrointestinali)	45%	55%

Dall' analisi dei sintomi presenti nel dataset si nota come il mal di gola, la tosse secca, la febbre e il problema di respirazione siano i sintomi più probabili in presenza del COVID-19. Altre colonne presenti nel dataset sono: abroad travel

(viaggi all'estero), contact with COVID Patient (*contatti con un paziente COVID*), attended large gathering (*partecipazioni a raduni affollati*), visited public exposed places (*luoghi pubblici esposti visitati*), family working in public exposed places (*famiglia che lavora in luoghi pubblici esposti*), wearing masks (*maschera indossata*), sanitization from market (*sanificazione dal mercato*) e la colonna che determina se una persona è affetta o meno dal virus: COVID-19.

L'origine dei dati è puramente studiata per essere utilizzata nella modellazione del modello di Machine Learning e dunque non potranno essere utilizzati per "contesti reali"

4.3 Preparazione dei dati

La preparazione dei dati include attività come la progettazione, la pulizia, la formattazione e il controllo di valori anomali all'interno del dataset.

Da questo punto in poi l'autore lavora con il linguaggio di programmazione **Python** che grazie alle sue librerie e tools sarà possibile effettuare tutte le operazioni del caso in questione, come il *data analysis*, la *suddivisione dei dati in training set e test set* e la *predizione del modello*. Per la preparazione dei dati, l'autore utilizza le seguenti librerie:

```
1 import pandas as pd
2 import numpy as np
3 import matplotlib
4 import seaborn as sns
```

Pandas è una libreria utile per eseguire analisi pratiche e reali dei dati in Python, **Numpy** è una libreria utile per eseguire calcoli scientifici in Python fornendo una serie di routine utili per operazioni veloci su array, **Matplotlib** è una libreria utile per la creazione di grafici statici matematici, animate e interattive in Python, **Seaborn** è una libreria utile al supporto di Matplotlib fornendo interfacce di alto livello per disegnare grafici statistici informativi. Nel file *data_preparation.py*⁶ si inizia dal caricamento del dataset (riga 5) e nella stampa (riga 6) delle informazioni utili del

⁶ Il file *data_preparation.py* è disponibile al seguente link: https://github.com/LaErre9/POC_MLOps-Covid_Probability_Infection_Checker/blob/main/data/data_preparation/data_preparation.py

dataset come l'indice dtype, le colonne, i valori non null e l'utilizzo della memoria.

```
5 covid = pd.read_csv('data.csv')
6 covid.info()
```

```
RangeIndex: 5434 entries, 0 to 5433
Data columns (total 21 columns):
#   Column                                     Non-Null Count  Dtype
---  -
0   Breathing Problem                         5434 non-null   int64
1   Fever                                     5434 non-null   int64
2   Dry Cough                                5434 non-null   int64
3   Sore throat                              5434 non-null   int64
4   Running Nose                             5434 non-null   int64
5   Asthma                                   5434 non-null   int64
6   Chronic Lung Disease                     5434 non-null   int64
7   Headache                                 5434 non-null   int64
8   Heart Disease                            5434 non-null   int64
9   Diabetes                                 5434 non-null   int64
10  Hyper Tension                            5434 non-null   int64
11  Fatigue                                  5434 non-null   int64
12  Gastrointestinal                         5434 non-null   int64
13  Abroad travel                            5434 non-null   int64
14  Contact with COVID Patient               5434 non-null   int64
15  Attended Large Gathering                 5434 non-null   int64
16  Visited Public Exposed Places            5434 non-null   int64
17  Family working in Public Exposed Places  5434 non-null   int64
18  Wearing Masks                           5434 non-null   int64
19  Sanitization from Market                 5434 non-null   int64
20  COVID-19                                5434 non-null   int64
dtypes: int64(21)
memory usage: 891.6 KB
```

Nella riga 7 invece viene generato una statistica descrittiva del dataset utilizzata per visualizzare alcuni dettagli statistici di base come percentile, media, std ecc. del frame di dati.

```
7 covid.describe().to_csv("dataset_statics.csv")7
```

	Breathing Problem	Fever	Dry Cough	.
count	5434	5434	5434	.
mean	0,666175929	0,786345234	0,792602135	.
std	0,471621133	0,409923567	0,405480268	.
min	0	0	0	
25%	0	1	1	
50%	1	1	1	
75%	1	1	1	
max	1	1	1	

⁷ Il risultato della tabella indica solo le colonne più rilevanti per questioni di dimensioni

4.3.1 Pulizia dei dati

Uno dei processi più comuni di pulizia dei dati è la gestione dei valori mancanti. Fondamentalmente, ci sono due modi per gestire i valori mancanti: rimuovere le righe con valori mancanti riducendo il dataset, oppure assegnare i valori mancanti con NaN. Sempre nel file *data_preparation.py* si verificano i dati mancanti e si salvano i risultati in una tabella per renderli leggibili:

```

7  # Verifica dei dati mancanti
8  missing_values = covid.isnull().sum()
9  percent_missing = covid.isnull().sum()/covid.shape[0]*100
10 value = {
11     'missing_values ':missing_values,
12     'percent_missing %':percent_missing
13 }
14 frame=pd.DataFrame(value)
15 frame.to_csv('missing_value.csv')

```

	missing_values	percent_missing %
Breathing Problem	0	0
Fever	0	0
Dry Cough	0	0
Sore throat	0	0
Running Nose	0	0
Asthma	0	0
Chronic Lung Disease	0	0
Headache	0	0
Heart Disease	0	0
Diabetes	0	0
Hyper Tension	0	0
Fatigue	0	0
Gastrointestinal	0	0
Abroad travel	0	0
Contact with COVID Patient	0	0
Attended Large Gathering	0	0
Visited Public Exposed Places	0	0
Family working in Public Exposed Place	0	0
Wearing Masks	0	0
Sanitization from Market	0	0
COVID-19	0	0

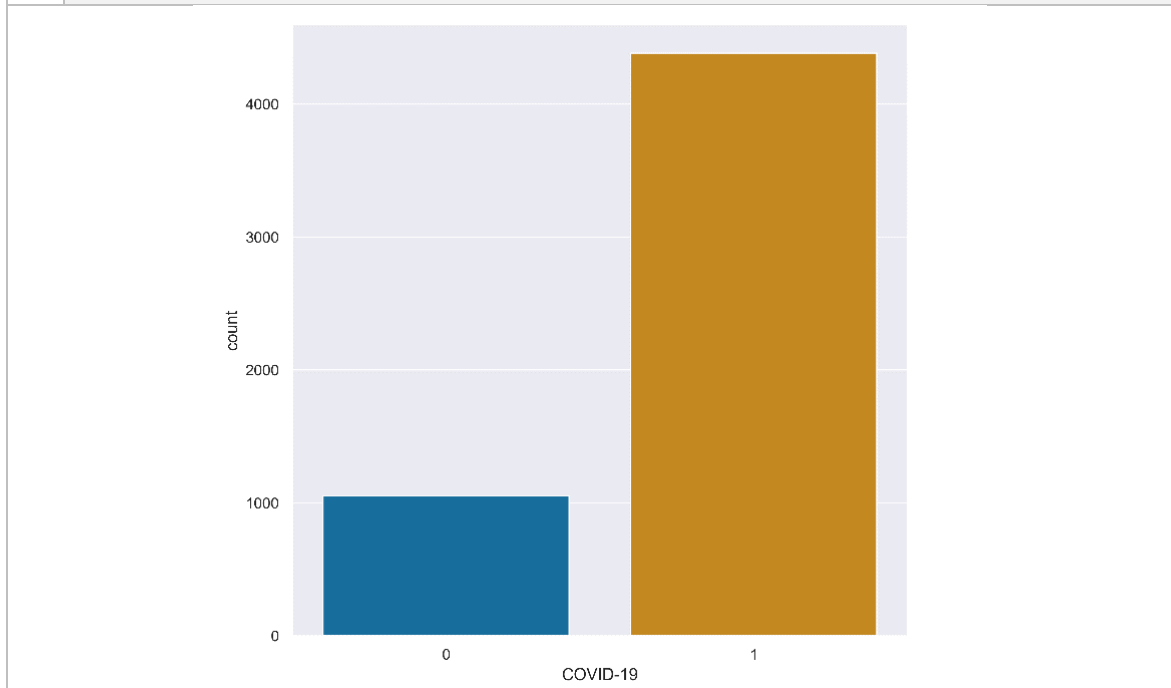
Nel dataset in esame, non ci sono “missing values”.

4.3.2 Visualizzazione dei dati

Anche per la visualizzazione dei dati si avvale del file *data_preparation.py* dove sarà possibile estrarre dal dataset statistiche utili per l'analisi dei dati. Nel caso in questione, l'autore ha ricavato alcune statistiche con le seguenti istruzioni:

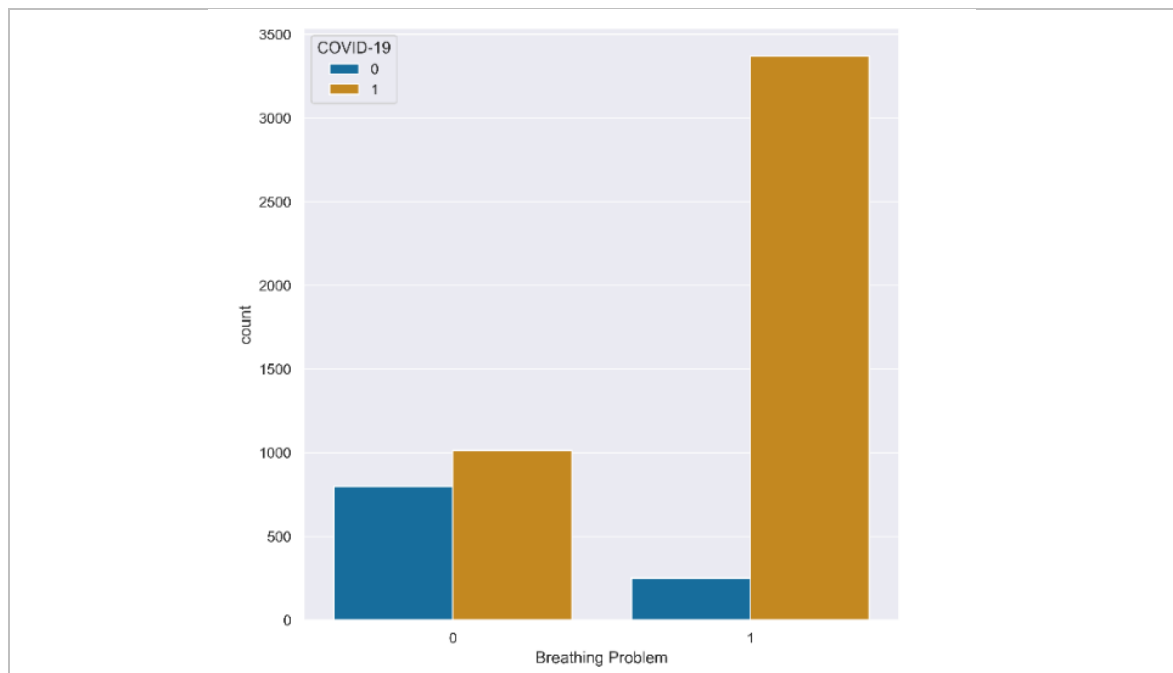
Si ricorda che 1 = Yes e 0 = No. Il primo grafico indica quante persone sono positive al COVID-19 su 5433.

```
1 sns_plot = sns.countplot(x='COVID-19', data=covid)
2 figure = sns_plot.get_figure()
3 figure.savefig('COVID-19.png', dpi = 400)
```



Nel dataset sono presenti circa 4000 casi di positività (1) e circa 1000 casi di negatività (0).

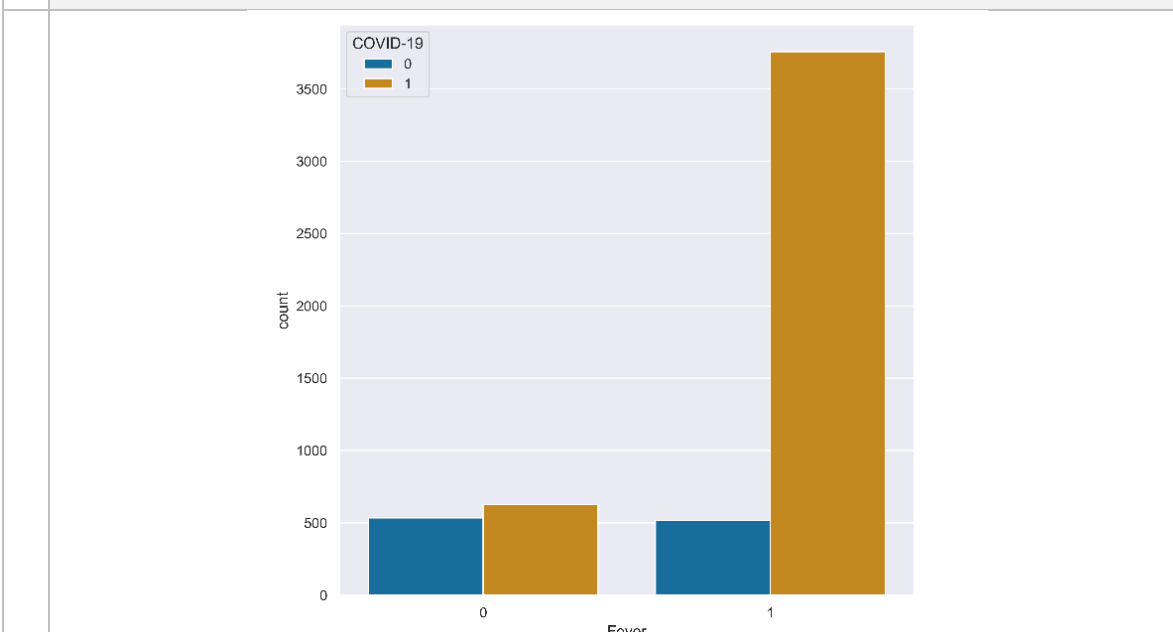
```
1 sns_breathing = sns.countplot(x='Breathing Problem',
    hue='COVID-19', data=covid)
2 figure1 = sns_breathing.get_figure()
3 figure1.savefig('BreathingProblem.png', dpi = 400)
```



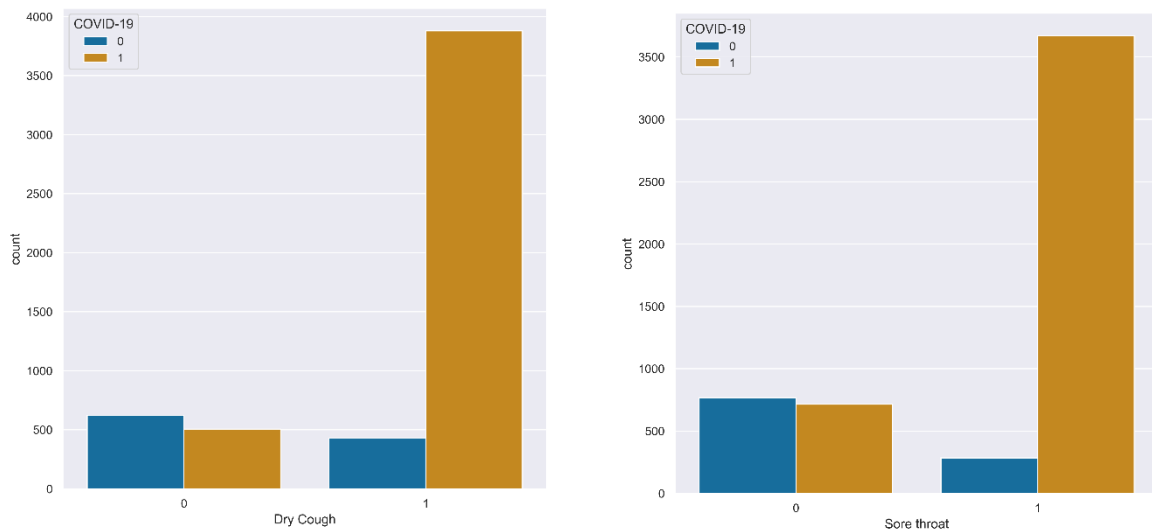
Il secondo grafico indica quante persone manifestano problemi di respirazione e quante persone, con questo sintomo, sono risultate positive su 5433.

Il grafico successivo indica quante persone manifestano febbre e sono risultate positive al COVID-19.

```
1 sns_fever = sns.countplot(x='Fever', hue='COVID-19',  
data=covid)  
2 figure2 = sns_fever.get_figure()  
3 figure2.savefig('Fever.png', dpi = 400)
```



Stessa cosa per tosse secca e mal di gola:



Si è scelto di analizzare questi sintomi poiché sono quelli predominanti per la predizione del sistema, infatti dai grafici si nota come la persona che ha quel sintomo sia anche risultato positivo al COVID-19.

4.3.3 Training dei dati, suddivisione dei dati e predizione

Una delle pratiche più importanti del Machine Learning è il training dei dati. Questo viene fatto perché i vari modelli osservano i dati che lo sviluppatore passa e la macchina imparerà i pattern e le ricorrenze. I dati dunque verranno divisi in training set e test set. Nel training set, i dati verranno utilizzati esclusivamente in fase di addestramento imparando le relazioni tra le variabili di input e il target. Nel test set, viene testato il modello su altre osservazioni che il modello non ha mai visto. Nel file *myTraining.py* l'autore ha utilizzato le stesse librerie dei paragrafi e creato una funzione `data_split(data, ratio)` per la suddivisione in training set e test set:

```

1  from numpy.random.mtrand import seed
2  def data_split(data, ratio):
3      np.random.seed(42)
4      shuffled = np.random.permutation(len(data))
5      test_set_size = int(len(data) * ratio)
6      test_indices = shuffled[:test_set_size]
7      train_indices = shuffled[test_set_size:]
   return data.iloc[train_indices], data.iloc[test_indices]
```

Nella funzione si passa il dataset (data) e ratio che sarà utile per il calcolo del **test_set_size**, ovvero la dimensione del test set. Nella riga 2 viene impostato un seed casuale di generatori di numeri pseudo casuale e nella riga 3 **shuffled** mescolerà gli indici dei nostri dati in modo che il **train_set** possa essere generato casualmente. Nella riga 4, supponendo che la lunghezza dei dati è 5433 e ratio è 0.2 ($5433 * 0.2 = 1086$) allora il **test_set_size** è 1086, e dunque **test_indices** andrà dall'indice iniziale fino a 1086 mentre **train_indices** andrà da 1086 a 5433.

```
8 df = pd.read_csv('data.csv')
9 train, test = data_split(df, 0.2)
10 X_train = train[['Breathing Problem', 'Fever', 'Dry Cough',
    'Sore throat', 'Running Nose', 'Asthma', 'Chronic Lung
    Disease', 'Headache', 'Heart Disease', 'Diabetes', 'Hyper
    Tension', 'Fatigue ', 'Gastrointestinal ', 'Abroad travel',
    'Contact with COVID Patient', 'Attended Large Gathering',
    'Visited Public Exposed Places', 'Family working in Public
    Exposed Places', 'Wearing Masks', 'Sanitization from
    Market']].to_numpy()

11 X_test = test[['Breathing Problem', 'Fever', 'Dry Cough',
    'Sore throat', 'Running Nose', 'Asthma', 'Chronic Lung
    Disease', 'Headache', 'Heart Disease', 'Diabetes', 'Hyper
    Tension', 'Fatigue ', 'Gastrointestinal ', 'Abroad travel',
    'Contact with COVID Patient', 'Attended Large Gathering',
    'Visited Public Exposed Places', 'Family working in Public
    Exposed Places', 'Wearing Masks', 'Sanitization from
    Market']].to_numpy()

12 Y_train = train[['COVID-19']].to_numpy().reshape(4348,)
13 Y_test = test[['COVID-19']].to_numpy().reshape(1086,)
```

Nella riga 8 si carica il dataset e nella riga 9 si suddivide il dataset in training set e test set con ratio = 0,2. Nella riga 10 si ha **X_train** che include tutte le variabili indipendenti utilizzate per addestrare il modello. **X_test** (riga 11) invece include tutte le variabili indipendenti che non verranno utilizzate nella fase di addestramento ma verranno utilizzate per fare previsioni e testare l'accuratezza del modello. Nella riga

12 si ha **Y_train** che include il target per la previsione e nella riga 13 si ha **Y_test** utilizzato per testare l'accuratezza della previsione.

`To_numpy()` viene utilizzata per convertire il dataframe in un array numpy e con `reshape` si dà una nuova forma a un array senza modificarne i dati.

Infine, sempre nel file *myTraining.py*⁸, si passa alla modellazione. Inizialmente, l'autore utilizza **LogisticRegression()** e fornisce alcuni dati di input al modello e calcola la predizione.

```
14 clf = LogisticRegression()
15 clf.fit(X_train, Y_train)
16 inputFeatures = [1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0,
17                  0, 0, 0, 0, 0]
18 infProb = clf.predict_proba([inputFeatures])[0][1]
19 print(infProb)

0.6512933809962855
```

Nell'array di **inputFeatures** (riga 16) fissiamo ad 1 tutti i sintomi che una X persona ha, si suppone infatti una persona che abbia un problema di respirazione (breathing problem), febbre (fever), tosse secca (dry cough), presenta una malattia cardiaca (heart disease), ipertensione (hyper tension) e mostra fatica (fatigue). Il risultato della predizione sarà circa 0.65, dunque la X persona ha una probabilità del 65% di essere infetto al COVID secondo il modello ML.

⁸ Il file *myTreaning.py* è disponibile al seguente link: https://github.com/LaErre9/POC_MLOps-Covid_Probability_Infection_Checker/blob/main/myTraining.py

4.4 Addestramento e sperimentazione del modello

Nell'addestramento e sperimentazione del modello si provano diversi algoritmi di Machine Learning che sulla base della accuratezza del modello scelto, si studia la soluzione migliore. In questa fase si adotta il controllo della versione a tutti i componenti del sistema ML con specifici strumenti e si testa il modello per assicurarci che il sistema funzioni come previsto.

A proposito di *test*, il team è incentivato a implementare test per scoprire le fonti di errore il prima possibile nel ciclo di sviluppo in modo da poter ridurre a valle l'aumento dei costi e il possibile tempo sprecato. Le pratiche MLOps adottano dunque la possibilità di sviluppare test automatici ogni qual volta che si implementa una modifica al sistema e continuare a svilupparli senza ripercussioni al sistema stesso.

Esistono diversi tipi di test: i *test del software* che assicurano che il codice segua i requisiti del progetto, implementato specialmente in DevOps come Unit Testing, Integration Testing, Acceptance Testing, i *test del modello* che definiscono se il modello funziona correttamente e i *test dei dati* che verificano la correttezza dei dati garantendo che il set di dati analizzato segua uno schema e contenga dati sufficienti. Il framework utilizzato durante la composizione dei test è la metodologia *Arrange Act Assert* dove 'Arrange' imposta i diversi ingressi su cui testare, 'Act' applica gli input sul componente che vogliamo testare e l'Assert conferma che se si è ricevuto l'output previsto. In Python esistono molti strumenti tra cui si ricorda *PyTest* che consente di implementare facilmente il framework AAA.

4.4.1 Analisi dell'accuratezza degli algoritmi di addestramento

Si provano dunque diversi algoritmi di training, che in base al dataset, avranno una differente accuratezza nel definire una predizione. Nel file *accuracy.py*⁹ vengono calcolate le diverse accuratezze per selezionare il miglior algoritmo di Machine Learning.

⁹ Il file *accuracy.py* è disponibile al seguente link: https://github.com/LaErre9/POC_MLOps-Covid_Probability_Infection_Checker/blob/main/models/accuracy.py

Logistic Regression

```
1 clf = LogisticRegression()  
2 clf.fit(X_train, Y_train)  
3 y_pred = clf.predict(X_test)  
4 accuracy_logreg = clf.score(X_test, Y_test)*100
```

Accuracy: 97.237569

Random Forest Regressor

```
5 clf1 = RandomForestRegressor(n_estimators=1000)  
6 clf1.fit(X_train, Y_train)  
7 accuracy_randomforest = clf1.score(X_test, Y_test)*100
```

Accuracy: 92.055318

Gradient Boosting Regressor

```
8 GBR = GradientBoostingRegressor(n_estimators=100, max_depth=4)  
9 GBR.fit(X_train, Y_train)  
10 accuracy_gbr=GBR.score(X_test, Y_test)*100
```

Accuracy: 89.572748

K Neighbors Classifier

```
11 knn = KNeighborsClassifier(n_neighbors=20)  
12 knn.fit(X_train, Y_train)  
13 y_pred = knn.predict(X_test)  
14 accuracy_knn = knn.score(X_test, Y_test)*100
```

Accuracy: 96.224678

Decision Tree Classifier

```
15 tree = tree.DecisionTreeClassifier()  
16 tree.fit(X_train, Y_train)  
17 y_pred1 = tree.predict(X_test)  
18 accuracy_decisiontree = tree.score(X_test, Y_test)*100
```

Accuracy: 98.434622

Gaussian Naive Bayes

```
19 model = GaussianNB()  
20 model.fit(X_train, Y_train)  
21 accuracy_gaussian= model.score(X_test, Y_test)*100
```

Accuracy: 76.611418

Costruzione tabella in ordine di score decrescenti

```
22 models = pd.DataFrame({
    'Model': ['KNN', 'Logistic Regression', 'Random Forest',
    'Naive Bayes', 'Decision Tree', 'Gradient Boosting
    Classifier'],
    'Score': [accuracy_knn, accuracy_logreg,
    accuracy_randomforest, accuracy_gaussian, accuracy
    _decisiontree, accuracy_gbk]})

23 print(models.sort_values(by='Score', ascending=False))
```

	Model	Score
4	Decision Tree	98.434622
1	Logistic Regression	97.237569
0	KNN	96.224678
2	Random Forest	92.055318
5	Gradient Boosting Classifier	89.572748
3	Naive Bayes	76.611418

In definitiva, si nota come l'algoritmo di ordinamento `DecisionTree()` sia il migliore come accuratezza. Tuttavia, anche il `LogisticRegression()` ha un'ottima accuracy ed entrambi risolvono il problema di classificazione, ma essendo che i dati presenti nel dataset sono linearmente separabili allora l'autore continuerà ad utilizzare l'algoritmo di ordinamento `LogisticRegression()`.

4.4.2 Data Versioning Control (DVC) dei dati

Dai principi definiti dall'MLOps, il Versioning è uno dei processi essenziali per la gestione degli esperimenti sul progetto ML. DVC è un software basato su Git e un suo obiettivo principale è sostituire file di grandi dimensioni (ad esempio un dataset con dimensioni molto grandi tanto da non essere supportato da Git) con piccoli "metafile" che puntano ai dati originali. In questo modo è possibile mantenere questi metafile insieme al codice sorgente del progetto in un repository mentre i file di grandi dimensioni vengono conservati in un archivio di dati remoto accessibile da

altri componenti del team. Ecco un flusso di lavoro per il Versioning dei dati:

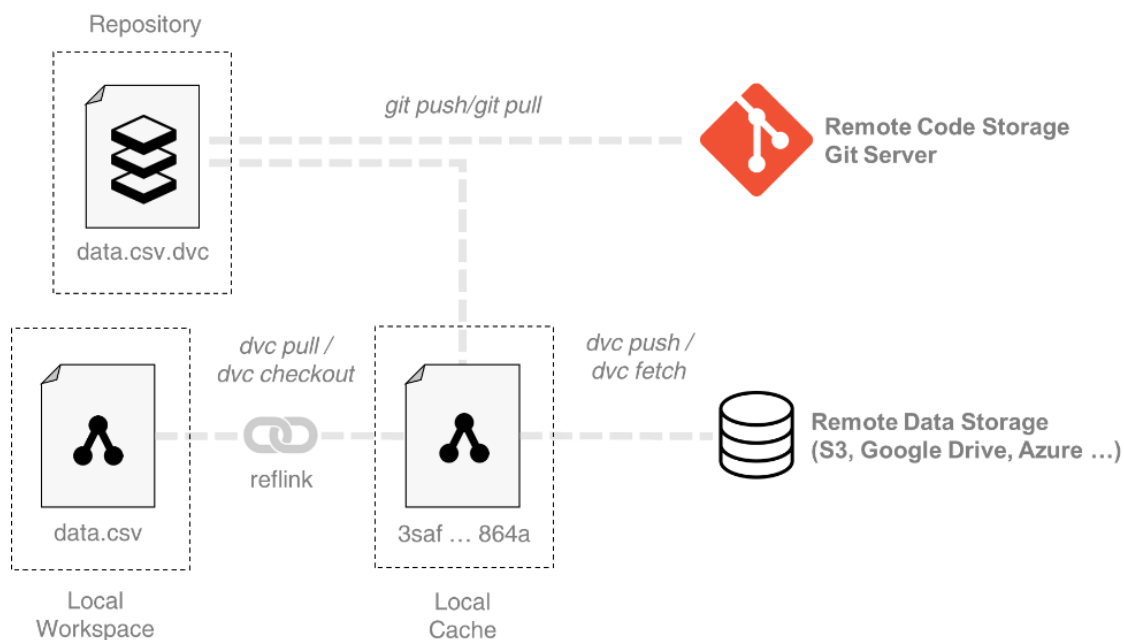


Figura 19. Flusso di lavoro DVC

Quindi DVC è uno strumento che funziona con Git. Git gestirà il codice, DVC aiuterà a gestire il dataset separatamente. Per configurare il dataset remoto sarà dunque necessaria l'installazione di DVC con `$ pip install DVC` (in Git Bash).

1	<code>\$ dvc init</code> (se non ancora fatto) per l'impostazione
---	---

Esistono diversi tipi di archiviazione remota, tra cui *Google Drive*. Si crea dunque una directory in Drive: 'remote' e si copia l'ultima parte dell'url

<https://drive.google.com/drive/folders/1ABCDEF8OVjm-PkYZG6YN4lBA1GLr-XrFsJ> e lo si copia nel comando:

1	<code>\$ dvc remote add -d myremote gdrive://1ABCDEF8OVjm-PkYZG6YN4lBA1GLr-XrFsJ</code>
---	---

Si aggiorna il repository Git con i comandi:

2	<code>\$ git add .</code>
3	<code>\$ git commit -m "commit DVC config"</code>
4	<code>\$ git push</code>

Si salva in locale, nella directory del repository, il dataset che si vuole trasferire in remoto. Lo si trasferisce seguendo i seguenti comandi su GitBash

5	\$ dvc add data.csv in automatico verrà creato un file chiamato <i>data.csv.dvc</i> in locale che verrà poi trasferito sul repository con
6	\$ git add data.csv.dvc .gitignore e
7	\$ git commit -m "commit DVC config file" e
8	\$ git push
9	\$ dvc push per il trasferimento del file sull'archivio remoto chiedendo l'autenticazione attraverso un ' <i>verification code</i> ' che verrà indicato dopo aver aperto il link proposto dopo aver digitato il comando. Verrà dunque caricato il file sull'archivio remoto Google Drive.

Si suppone che un altro sviluppatore vuole accedere ai dati e al codice, e dunque l'operazione che farà sarà:

	\$ git clone del repository
	\$ dvc pull data.csv e anche in questo caso chiederà l'autenticazione attraverso un ' <i>verification code</i> '.
	Inizierà il download e sarà dunque presente nella directory

Visto che nel paragrafo 4.3, si nota che la memoria occupata dal dataset è circa 891.6 KB si può evitare questo passaggio e dunque continuare ad usare il repository GitHub per il tracking del file 'data.csv'.

4.4.3 Test del modello ML

MLOps richiede di testare il modello ML, si comprende dunque il problema, si esplorano i dati, si elaborano i dati e si costruisce un modello utilizzando algoritmi di apprendimento automatico. È sempre bene testare il modello misurando le qualità del modello stesso e vedere quanto bene può fare rispetto al caso d'uso in questione. Per il test, si utilizza come algoritmo di apprendimento il **LogisticRegression()** e si valuta l'*accuratezza* (accuracy), la *precisione* (precision), la *specificità* (specificity) e la *sensibilità* (sensitivity). L'accuratezza è calcolata come il rapporto:

$$\text{accuratezza} = \frac{\text{numero totale di } \textbf{campioni correttamente previsti}}{\text{numero totale di } \textbf{campioni}}$$

Anche in questo caso si scriverà sul file myTraining.py

Calcolo dell'accuratezza	
1	<code>clf = LogisticRegression()</code>
2	<code>clf.fit(X_train, Y_train)</code>
3	<code>y_pred = clf.predict(X_test)</code>
4	<code>accuracy_logreg = clf.score(X_test, Y_test)*100</code>
Accuracy: 0,97237569	

Il calcolo della precisione è:

$$precisione = \frac{veri\ positivi\ (TP)}{veri\ positivi\ (TP) + falsi\ positivi\ (FP)}$$

5	<code>precision = precision_score(Y_test, y_pred)</code>
Precision: 0,9754738015607581	

La metrica di precisione può essere utilizzata per valutare il classificatore quando il set di dati è un set di dati bilanciato, se è sbilanciato non può essere utilizzato. Per valutare un classificatore ogni volta che il set di dati è sbilanciato, si costruisce una matrice di confusione, nell'esempio in questione, è usato la matrice di confusione con due classi obiettivo:

	Predicted:NO	Predicted:YES
Actual:NO	Vero Negativo (TN)	Falso Positivo (FP)
Actual:SI	Falso Negativo (FN)	Vero Positivo (TP)

dove:

Veri positivi (TP): sono i casi in cui il "Sì" previsto apparteneva effettivamente alla classe "Sì".

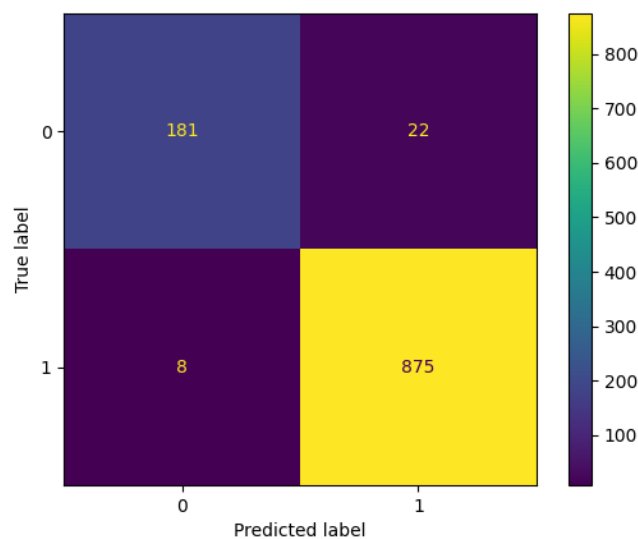
Veri negativi (TN): sono i casi in cui il "No" previsto apparteneva effettivamente alla classe "No".

Falsi positivi (FP): sono i casi in cui il "Sì" previsto apparteneva effettivamente alla classe "No".

Falsi negativi (FN): sono i casi in cui il "No" previsto apparteneva effettivamente alla classe "Sì".

Si costruisce la matrice di confusione su Python nel seguente modo: si prelevano i valori dalla matrice di confusione e definirli come tn, fp, fn, tp e poi graficarli a video in una matrice che mi determina il numero di veri negativi (181), veri positivi (875), falsi positivi (22) e falsi negativi (8)

```
6 tn, fp, fn, tp = confusion_matrix(Y_test, y_pred).ravel()
  # matrice di confusione a video
7 plot_confusion_matrix(clf, X_test, Y_test)
8 plt.show()
```



Sulla base dei valori ottenuti nella matrice di confusione si valutano poi la specificity:

```
9 specificity = tn / (tn + fp)
```

```
Specificity: 0,8916256157635468
```

e la sensitivity:

```
10 sensitivity = tp / (tp + fn)
```

```
Sensitivity: 0,9909399773499433
```

Si salvano queste valutazioni in un file.json che servirà nel prossimo paragrafo.

```
11 import json
12 with open("metrics.json", 'w') as outfile:
13     json.dump({ "accuracy": acc_logreg, "specificity":
specificity, "sensitivity":sensitivity, "precision":
precision}, outfile)
```


4.4.4 Track del modello ML con DVC e GitHub Actions

Una delle pratiche della sperimentazione del modello è il tracking con GitHub Actions. Viene confrontato il modello ML su diversi rami Git di un progetto in modo tale che durante la sperimentazione si vuole sapere quanto stia andando bene rispetto al modello che c'è sul ramo principale, ovvero quello in produzione. È possibile svolgere questa operazione utilizzando le DVC pipelines, CML e GitHub Actions e verificare le miglorie attraverso una tabella che confronta le diverse metriche adottate in diverse sperimentazioni. La pipeline è un metodo per mettere insieme tutti i passaggi dall'acquisizione dei nostri dati alla modellazione e garantire che i file di metriche (*metrics.json*) siano sempre generati dallo stesso insieme di processi e che siano facili da confrontare tra i commit e tra i rami in un modo più naturale per i dati numerici.

A riga di comando su Git Bash, si configura la pipeline DVC:

```
1 $ dvc init – per inizializzare dvc nel repository
```

Nella directory locale si crea un file **dvc.yaml**¹⁰ in cui viene scritta la pipeline che deve essere eseguita in riga di comando. La pipeline è basata su tre componenti: input, output e comando. L'autore usa soltanto il comando poiché l'output è già stato processato e scaricato dalla fonte.

```
2 stages:
   #fase della pipeline
3 train:
   #comando
4   cmd: python myTraining.py
   #dipendenze
5   deps:
   - myTraining.py
   - data.csv
6   metrics:
   - metrics.json:
       cache: false
```

¹⁰ Il file può essere generato in maniera automatica attraverso il comando `dvc run` riconoscendo tutte le dipendenze e gli argomenti da passare, il file è disponibile al seguente link:

https://github.com/LaErre9/POC_MLOps-Covid_Probability_Infection_Checker/blob/main/dvc.yaml

Verrà creato anche un secondo file: **dvc.lock**. All'interno del file si può trovare il percorso e un codice hash per ogni file di ogni fase in modo che DVC possa tenere traccia delle modifiche. Tenere traccia di queste modifiche è importante perché ora DVC saprà quando una fase deve essere rieseguita o meno.

```
schema: '2.0'
stages:
  train:
    cmd: python myTraining.py
    deps:
      - path: data.csv
        md5: 02b3a4aa918c639fb02b6d3cd98914c4
        size: 234007
      - path: myTraining.py
        md5: d8452f7c93722709dd9bf2069c4ae3ee
        size: 4770
    outs:
      - path: metrics.json
        md5: bb7d10954900192c72c9dd340b3dc570
        size: 131
```

Si passa di nuovo sul Git Bash e si digita:

```
1 $ dvc repro – eseguirà quanto presente in dvc.yaml
2 $ git add . – per aggiornare il repository online
3 $ git commit -m “setup DVC pipeline” – commit
4 $ git push
```

Successivamente si crea il flusso di lavoro per GitHub Actions e dunque si crea nello stesso repository un'ulteriore directory `.github/workflows/train.yaml` utilizzando il codice CML (Continuous Machine Learning). CML è un tool per implementare il *continuous integration* e *delivery* per MLOps. Nel caso in esame è usato per il confronto delle metriche. Nel file `train.yaml` verrà scritto:

```
name: covid-model
on: [push]
jobs:
  run:
    runs-on: ubuntu-latest
```

	<pre> container: docker://dvcorg/cml-py3:latest steps: - uses: actions/checkout@v2 - name: cml_run env: REPO_TOKEN: \${{ secrets.GITHUB_TOKEN }} run: pip install -r requirements.txt dvc repro git fetch --prune #differenze del branch sperimentale con il main dvc metrics diff --show-md main > report.md cml-send-comment report.md </pre>
1	\$ git add . – per aggiornare il repository
2	\$ git commit -m “make CML workflows”
3	\$ git push

Si passa al repository del modello su GitHub e si crea un nuovo branch dove poter fare le diverse sperimentazioni (A).

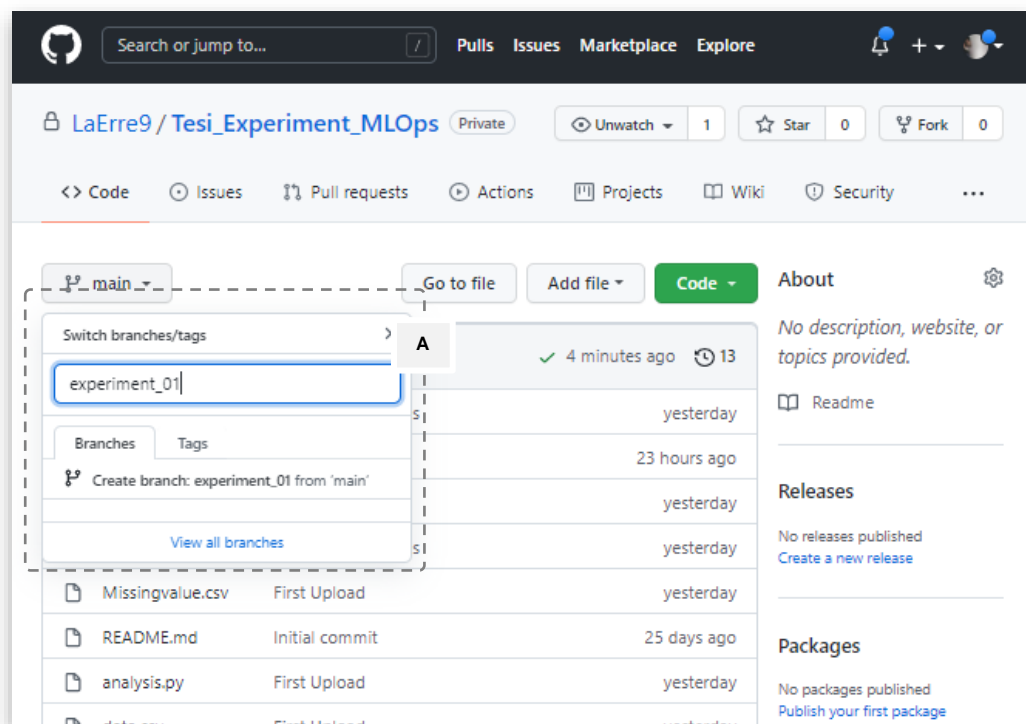


Figura 20. Creazione nuovo branch repository GitHub

Si entra nel nuovo branch (in questo caso `experiment_01`) e si modifica il file `myTraining.py` modificando l'algoritmo di Machine Learning `LogisticRegression()` in `DecisionTreeClassifier()` e verificare le differenze.

File: myTraining.py
Aggiungere se non c'è: <code>from sklearn import tree</code> Modificare: <code>clf = tree.DecisionTreeClassifier()</code>

Si fa **commit** per attuare le modifiche:

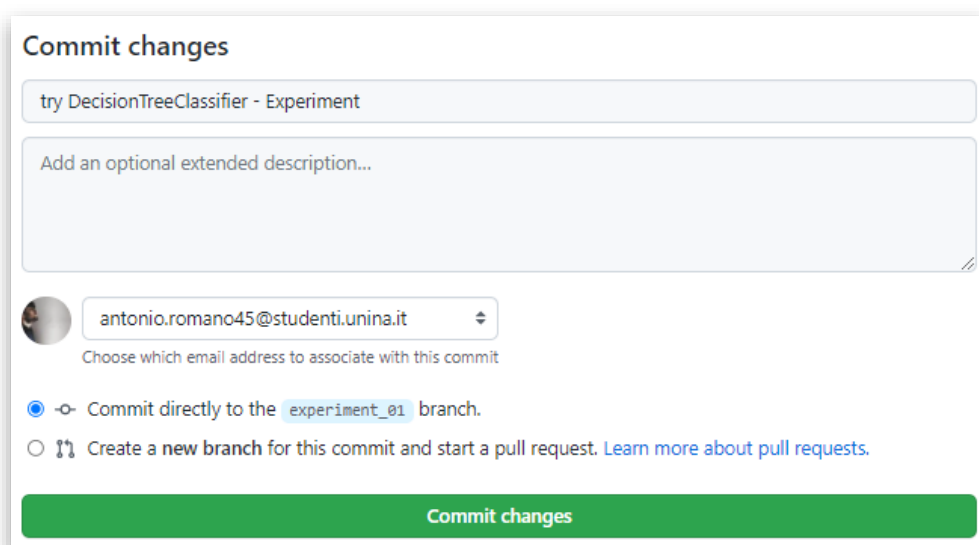


Figura 21. *Commit changes su GitHub*

Andando in `< > code` compare il button “**Compare & Pull request**”

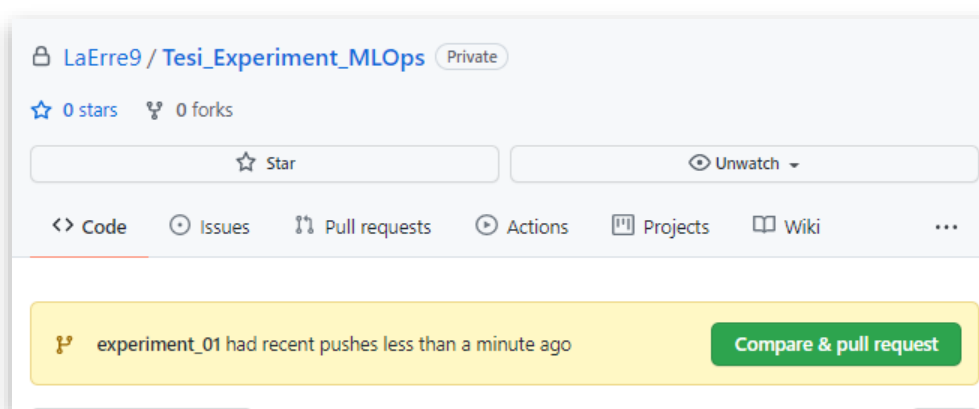


Figura 22. *Compare & Pull request su GitHub*

Si clicca e si crea un “pull request”¹¹

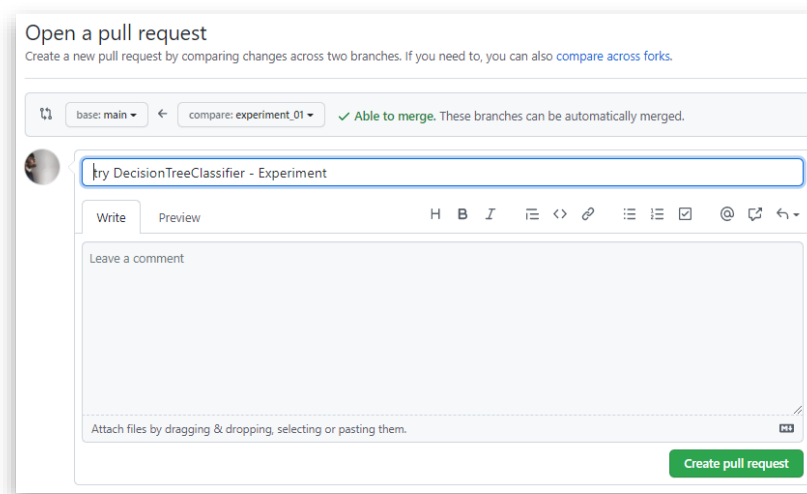



Figura 23. Creazione Pull request su GitHub

Dopo qualche minuto, se tutto è andato a buon fine, uscirà la spunta di  **check ok**, si ricarica la pagina e il risultato sarà (B):

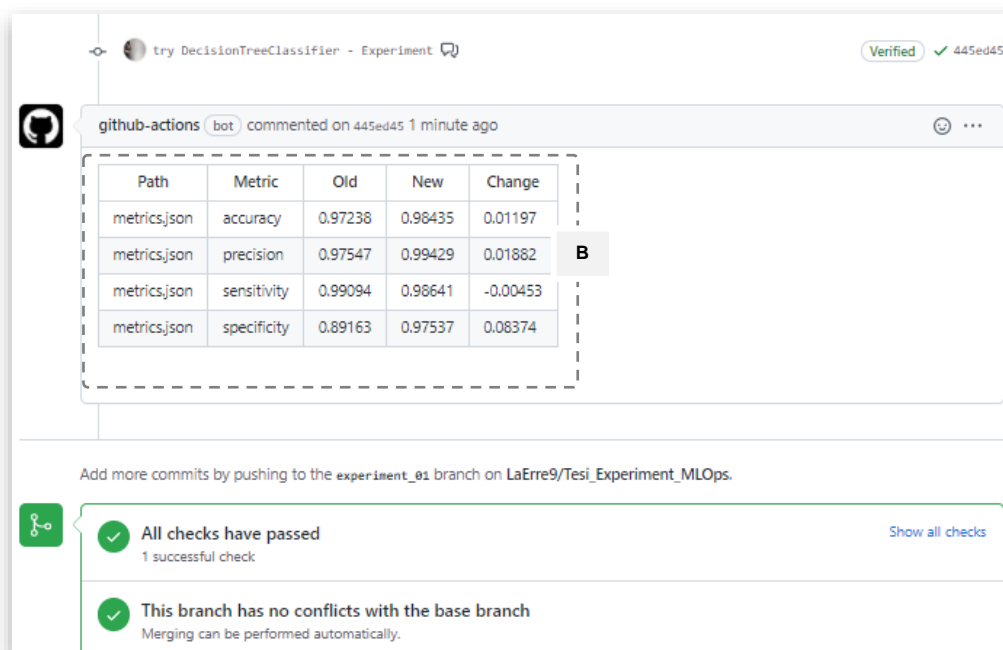


Figura 24. DVC metrics difference tra il main branch e experiment_01 branch
(Decision Classifier)

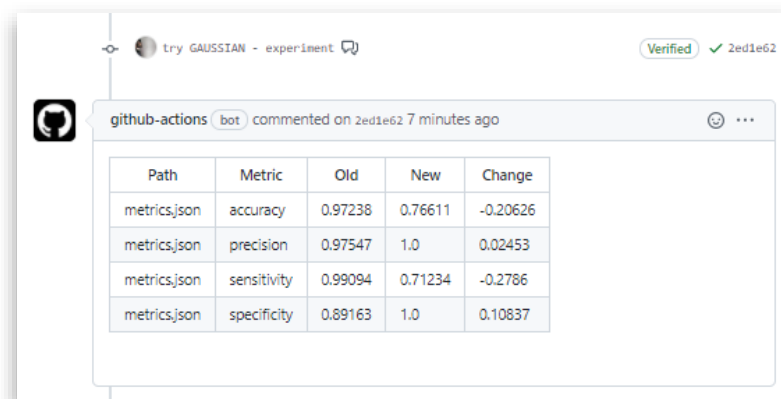
¹¹ Il pull request consente di informare i componenti del team che contribuiscono al repository delle modifiche che inviano ad un branch. Una volta aperta è possibile discutere con altri componenti del team, rivedere le potenziali modifiche e aggiungere dei commit prima che le modifiche vengano unite nel branch principale.

Si nota dunque una tabella con le differenze delle metriche calcolate nel branch principale e nel branch sperimentale. Si nota inoltre, come il modello non sia peggiorato, anzi, sia migliorato in tutto tranne per la sensitivity, pertanto, si potrebbe unire la modifica (merge branch) e continuare con l'algoritmo di apprendimento Decision Tree Classifier. Si prova a fare la stessa cosa anche per gli altri due algoritmi di apprendimento: **GaussianNB** e **Kneighbors**.

Si modifica di myTraining.py con algoritmo di apprendimento **GaussianNB**

```
Aggiungere se non c'è: from sklearn.naive_bayes import
GaussianNB
Modificare: clf = GaussianNB()
```

Si fa il **commit** della modifica (try GAUSSIAN – experiment) e GitHub Action ricalcola le modifiche fatte e la tabella delle differenze delle metriche tra gli algoritmi di apprendimento aggiungendola nello stesso pull request. Si notano dei peggioramenti e dunque questa modifica può essere scartata.



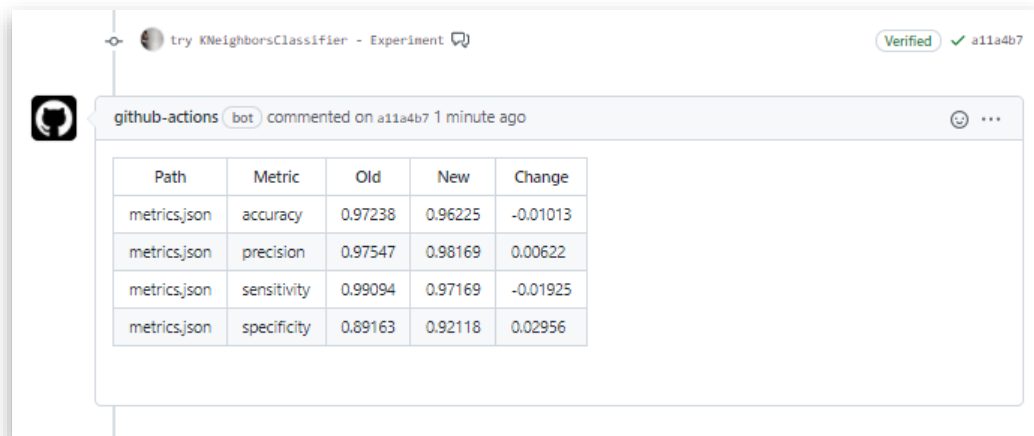
Path	Metric	Old	New	Change
metrics.json	accuracy	0.97238	0.76611	-0.20626
metrics.json	precision	0.97547	1.0	0.02453
metrics.json	sensitivity	0.99094	0.71234	-0.2786
metrics.json	specificity	0.89163	1.0	0.10837

Figura 25. DVC metrics difference tra il main branch e experiment_01 branch (GaussianNB)

Si prova di nuovo a modificare myTraining.py con algoritmo di Machine Learning **KNeighborsClassifier**

```
Aggiungere se non c'è: from sklearn.neighbors import
KNeighborsClassifier
Modificare: clf = KNeighborsClassifier(n_neighbors=20)
```

Commit della modifica (try KNeighborsClassifier – Experiment) e si attende il calcolo di GitHub Action.



The screenshot shows a GitHub Actions workflow comment for 'try KNeighborsClassifier - Experiment'. It includes a table with DVC metrics comparing the 'main' branch (Old) and the 'experiment_01' branch (New). The metrics are accuracy, precision, sensitivity, and specificity. The table shows a decrease in accuracy and sensitivity, and a slight increase in precision and specificity.

Path	Metric	Old	New	Change
metrics.json	accuracy	0.97238	0.96225	-0.01013
metrics.json	precision	0.97547	0.98169	0.00622
metrics.json	sensitivity	0.99094	0.97169	-0.01925
metrics.json	specificity	0.89163	0.92118	0.02956

Figura 26. DVC metrics difference tra il main branch e experiment_01 branch (KNeighborsClassifier)

Anche questa soluzione può essere scartata, in quanto presenta dei peggioramenti sia in accuratezza e sia in sensibilità.

In definitiva questo procedimento è utile soprattutto in team, in quanto, nelle fasi di modifiche e aggiornamenti, per evitare che si possa modificare accidentalmente il modello in produzione, si possono effettuare sperimentazioni per migliorare il modello o risolvere problemi. Grazie a GitHub si ha una panoramica di chi sta modificando e cosa sta modificando in modo da avere tutto organizzato.

4.4.5 Test di predizione del modello ML

Per assicurarsi che il modello calcoli la probabilità attesa, si fa riferimento al risultato che propone il **LogisticRegression()** e se si testa un nuovo modello si verifica in maniera automatica se è accettabile il risultato oppure no. Anche in questo caso si fa l'utilizzo di CML e GitHub Actions e si verifica la differenza del risultato attraverso una tabella che confronta le diverse metriche adottate nelle sperimentazioni. Il file è *test_model.py*¹²

```
1 clf = LogisticRegression()
```

¹² Il file *test_model.py* è disponibile al seguente link: https://github.com/LaErre9/POC_MLOps-Covid_Probability_Infection_Checker/blob/main/test/test_model.py

```

2 clf.fit(X_train, Y_train)
3 y_pred = clf.predict(X_test)

#Calcolo della prediction
4 inputFeatures = [1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0,
0, 0, 0, 0, 0]
5 infProb = clf.predict_proba([inputFeatures])[0][1]
6 accuracy_logreg = clf.score(X_test, Y_test)

# Write results to file
7 with open("test/test_score_and_prediction.json", 'w') as
  outfile:
      json.dump({ "Accuracy": accuracy_logreg,
"Prediction": infProb}, outfile)

```

Si scrivono i risultati in un file *test_score_and_prediction.json* che sarà utile per GitHub Actions per il calcolo delle differenze tra il branch di produzione e quello di sperimentazione.

Si crea poi un file per l'automazione del test, anche in questo caso nella directory **.github/workflows**, *test.yaml*¹³:

```

1 name: prediction-test
2 on: [push]
3 jobs:
4   run:
5     runs-on: ubuntu-latest
6     container: docker14://dvcorg/cml-py3:latest
7     steps:
8       - uses: actions/checkout@v2
9       - name: cml-run
10      env:
11        REPO_TOKEN: ${{ secrets.GITHUB_TOKEN }}
12      run: |
13
14        # Run Test
15        pip install -r requirements.txt

```

¹³ Il file *test.yaml* è disponibile al seguente link: https://github.com/LaErre9/POC_MLOps-Covid_Probability_Infection_Checker/blob/main/test/test.py

¹⁴ Docker fornisce flussi di lavoro e collaborazione per creare applicazioni.


```

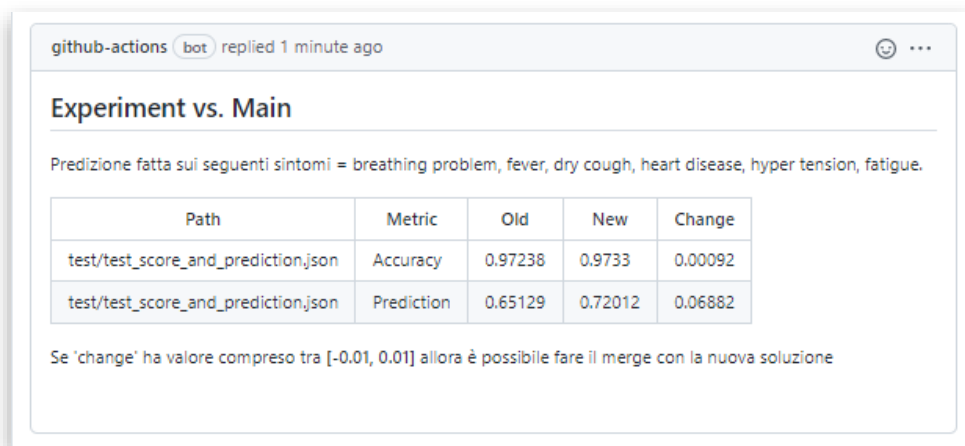
14      python test/test_model.py

      # Report the difference
15      echo "## Experiment vs. Main" > report.md
16      echo "Predizione fatta sui seguenti sintomi =
breathing problem, fever, dry cough, heart disease, hyper
tension, fatigue." > report.md
17      git fetch --prune
18      dvc metrics diff main --targets
test/test_score_and_prediction.json --show-md >> report.md

19      cml-send-comment report.md

```

Come negli esempi precedenti, si calcola la differenza con il comando `$ dvc metrics diff main` e lo si carica su un `report.md` utile a GitHub per visualizzare meglio le differenze di risultati previsti dai modelli ML. Verranno caricate le modifiche sul GitHub per aggiornare il repository. Si suppone che uno sviluppatore X decide di cambiare il modello poiché ritiene migliore secondo i suoi calcoli. Creerà un branch di sperimentazione e modificherà il modello in `LogisticRegressionCV()`¹⁵. In automatico, dopo il commit dello sviluppatore X, GitHub calcolerà la differenza e se tutto è andato bene verranno mostrati i risultati in un pull request:



github-actions (bot) replied 1 minute ago

Experiment vs. Main

Predizione fatta sui seguenti sintomi = breathing problem, fever, dry cough, heart disease, hyper tension, fatigue.

Path	Metric	Old	New	Change
test/test_score_and_prediction.json	Accuracy	0.97238	0.9733	0.00092
test/test_score_and_prediction.json	Prediction	0.65129	0.72012	0.06882

Se 'change' ha valore compreso tra [-0.01, 0.01] allora è possibile fare il merge con la nuova soluzione

Figura 27. DVC metrics difference tra il main branch e experiment_03 branch

¹⁵ La differenza che c'è tra `LogisticRegressionCV` e `LogisticRegression` è che quest'ultimo si limita ad addestrare la logistica regressione sui dati forniti, mentre `LogisticRegressionCV` divide il set di dati Train in diverse combinazioni Train/Validation Set prima dell'addestramento e aiuta a verificare che il modello sia valido su diverse suddivisioni di dati

I risultati mostrati nella **Figura 27** non entrano nel range $[-0.01, 0.01]$ e pertanto non verrà fatto il merge con il main branch di produzione. Questo test è stato fatto per garantire che durante le possibili modifiche che si fanno nel team si eviti di alterare il risultato di predizione che `LogisticRegression()` ha calcolato.

Un valido aggiornamento del codice potrebbe essere quello di convalidare il modello dopo una modifica dei dati, accettandosi che i risultati siano quelli previsti e automaticamente fare il merge con il main branch.

4.5 Test automatizzati per modelli di Machine Learning

Un'ulteriore pratica dell'MLOps è la possibilità di controllare il modello in produzione ed eseguire una suite di test automatizzati sul modello stesso. L'obiettivo del controllo automatico è capire se il modello funziona su una qualche altra macchina poiché testare il modello in locale non è davvero sufficiente, non è rigoroso. Pertanto, serve creare un controllo ermetico in cui si sa davvero qual è l'ambiente di esecuzione, qual è l'hardware, qual è il sistema operativo e poi renderlo disponibile per tutti i membri del team funzionante. Questa pratica è davvero utile per accelerare i cicli di sviluppo a lungo termine.

Innanzitutto, c'è la necessità di configurare un workflow di GitHub Actions in grado di attivarsi ogni qualvolta che avviene una modifica al codice. Dunque si va nella cartella `.github/workflows/` e si crea un file di configurazione: `auto_test.yml`¹⁶.

```
name: covid-model-auto-test
# ogni volta che viene rilevata un push al repository GitHub
# avvierà una macchina (trigger)
on: [push]
jobs:
  run:
    # una macchina che avrà come SO Ubuntu
    runs-on: ubuntu-latest
    container: docker://dvcorg/cml-py3:latest
    steps:
      - uses: actions/checkout@v2
      - name: sanity-check
```

¹⁶ il file `auto_test.yml` è disponibile al seguente link: https://github.com/LaErre9/POC_MLOps-Covid_Probability_Infection_Checker/blob/main/.github/workflows/auto_test.yml

```
env:
  REPO_TOKEN: ${{ secrets.GITHUB_TOKEN }}
# esegue
run: |

  # l'installazione dei requisiti e myTraining.py
  pip install -r requirements.txt
  python myTraining.py
```

Per la prova del controllo automatico, si suppone che uno *sviluppatore X* del team vuole modificare il codice *myTraining.py* e vuole cambiare l'algoritmo di apprendimento perché secondo lui **DecisionTreeClassifier()** è meglio di **LogisticRegression()** e dunque:

```
1 clf = DecisionTreeClassifier()
```

Il codice è sicuramente eseguibile in locale ma non è condiviso con altri sviluppatori del team. Dunque:

```
2 $ git add . – per aggiornare il repository online
3 $ git commit -m “change model - Sviluppatore X” – commit da parte
  dello sviluppatore X
4 $ git push origin experiment_02 – push su un branch sperimentale
```

In automatico, GitHub Actions si attiverà ed eseguirà *myTraining.py* ma volutamente si è sbagliato in riga 1 per verificare come GitHub controlla qualsiasi tipo di errore, infatti compare una X e una segnalazione di **checks falliti**:

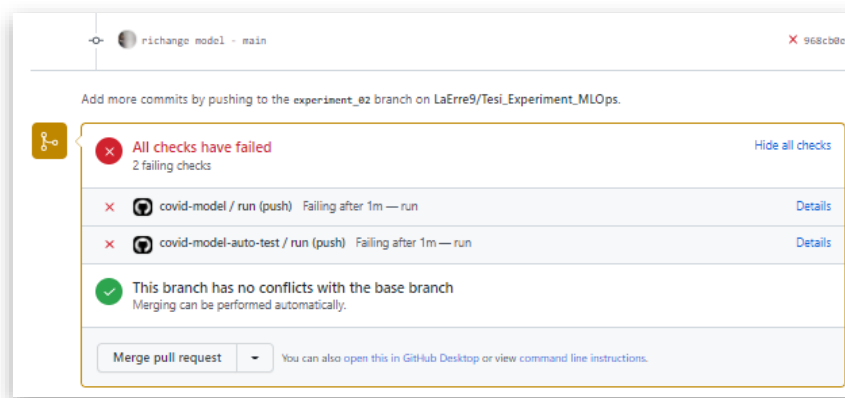


Figura 28. Error checks in CML files

Cliccando su [Details](#) è possibile studiare perché l'esecuzione è fallita:

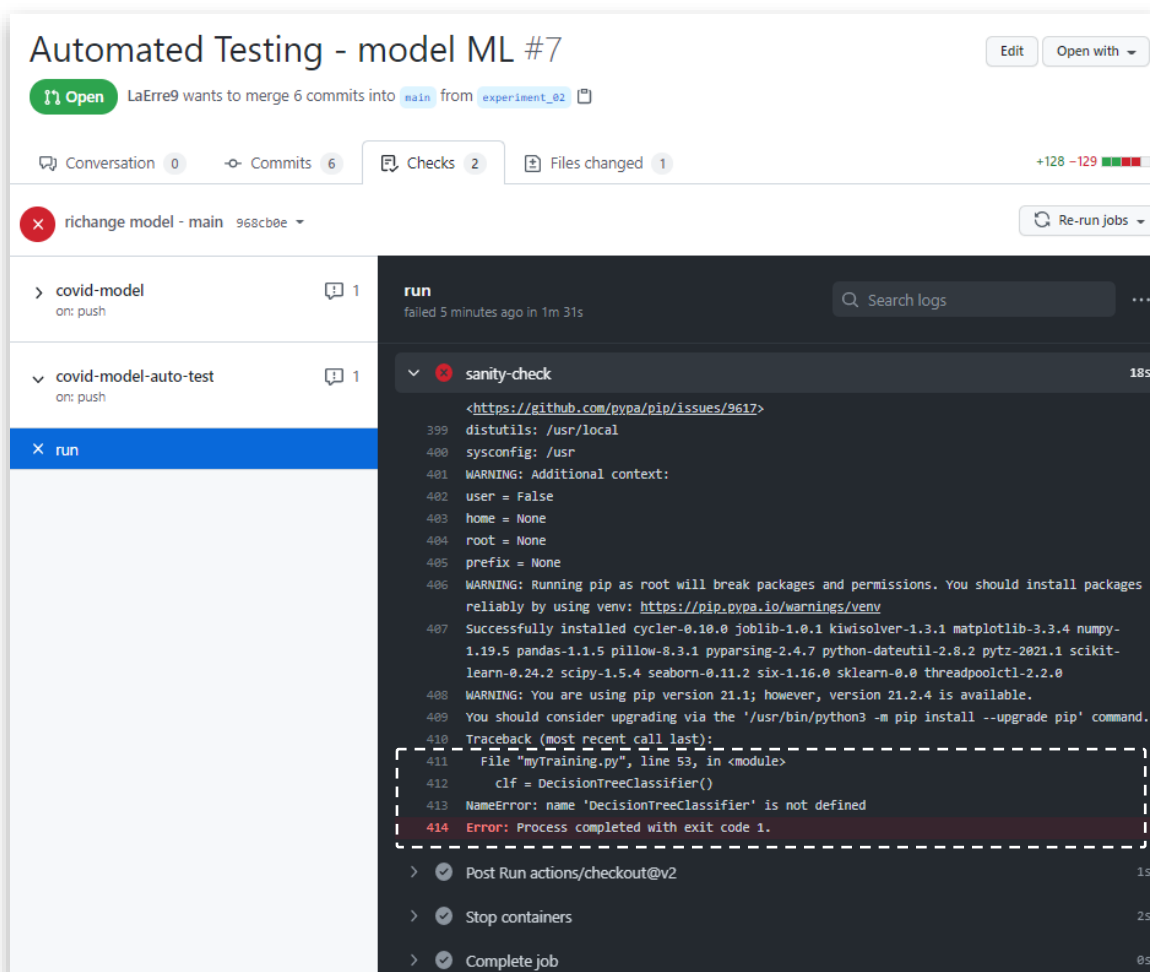


Figura 29. Ricerca dell'errore nel runner di GitHub Actions

GitHub Actions ci indica, infatti, l'errore fatto nella riga 1. Si suppone che lo sviluppatore Y sia più esperto e riesce a trovare la soluzione, dunque farà sulla sua macchina:

```
$ git pull origin experiment_02 e modificherà quanto segue nel file
myTraining.py:
clf = tree.DecisionTreeClassifier()
e condividerà le modifiche nel repository
$ git add . – per aggiornare il repository online
$ git commit -m “fixed change model - Sviluppatore Y” – commit
da parte dello sviluppatore Y
$ git push origin experiment_02 – push su un branch sperimentale
```

In automatico GitHub Actions si riaziona e verificherà se il problema sarà risolto indicandoci un “check ok”.

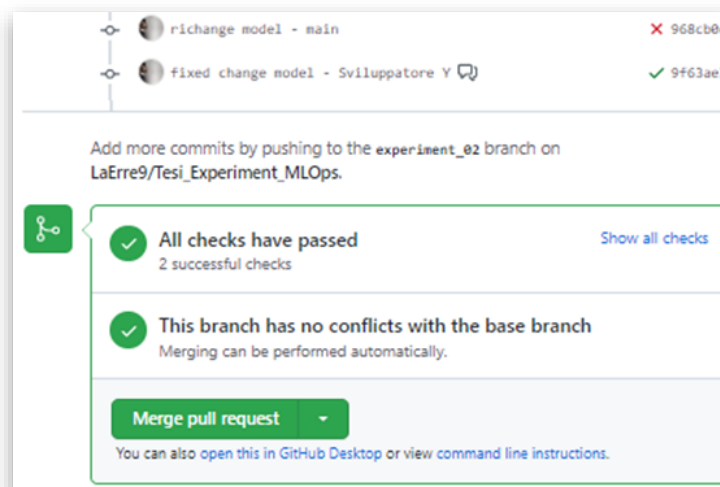


Figura 30. Ricerca dell'errore nel runner di GitHub Actions

Si suppone di creare un test automatizzato (si può aggiungere una suite di test che verranno automaticamente eseguiti ad ogni push). Si mostra una parte del codice nel file *test.py*¹⁷ nella directory /test che lo *sviluppatore X* ha pensato di caricare.

```
# Generazione di alcuni dati per la validazione
1 clf = LogisticRegression()
2 clf.fit(X_train, Y_train)
3 X_test, y = make_regression(1000, n_features=11) #Errore
# Test sul modello
4 y_hat = clf.predict(X_test)
```

Nel file *auto_test.yaml* si aggiunge all'ultima riga **python test/test.py** in modo tale che l'actions eseguirà il file *test.py*. Nel test non viene valutata nessuna statistica sul risultato del modello ma viene valutato se il modello è completo. Idealmente, si vuole, anche in questo caso, che ogni volta che avviene una modifica al nostro codice sul modello si azionino i test a valle di un push nel repository. Si suppone che lo *Sviluppatore X*, carica sul repository questa versione del codice *test.py* con:

¹⁷ Il file *test.py* è disponibile al seguente link: https://github.com/LaErre9/POC_MLOps-Covid_Probability_Infection_Checker/blob/main/test/test.py

```
$ git add . – per aggiornare il repository online
$ git commit -m “fixed change model - Sviluppatore X” – commit
da parte dello sviluppatore X
$ git push origin experiment_test – push su un branch sperimentale
```

In automatico comparirà “**compare & create pull request**” e si creerà il pull request nel quale dopo un po' di tempo restituirà se il test è stato superato oppure no. Nel caso dello *sviluppatore X* il risultato è:

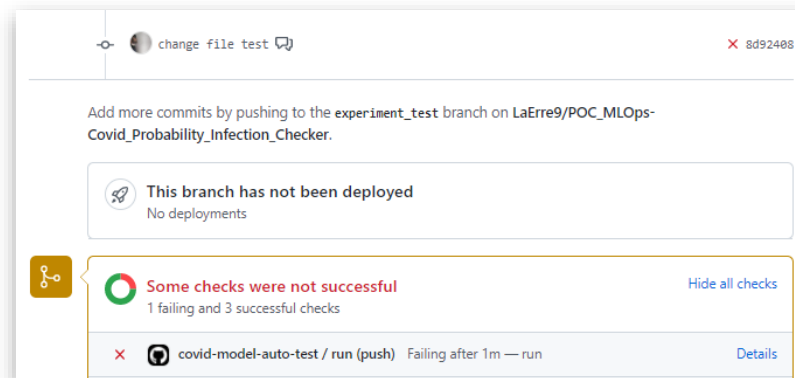


Figura 31. Errore di GitHub Actions nel pull request

Nel runner verrà indicato l'errore:

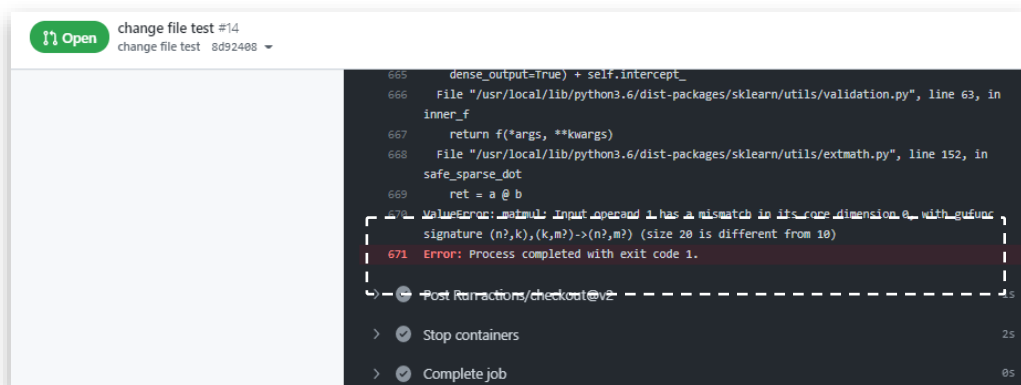


Figura 32. Errore nel runner di GitHub Actions

Anche in questo caso, lo *sviluppatore Y* riesce ad intuire il problema e modifica il codice *test.py*:

```
# Generazione di alcuni dati per la validazione
clf = LogisticRegression()
```

```
1 clf.fit(X_train, Y_train)
2 X_test, y = make_regression(1000, n_features=20)
3 # Test sul modello
  y_hat = clf.predict(X_test)
```

Caricherà le modifiche sul repository e il risultato sarà:

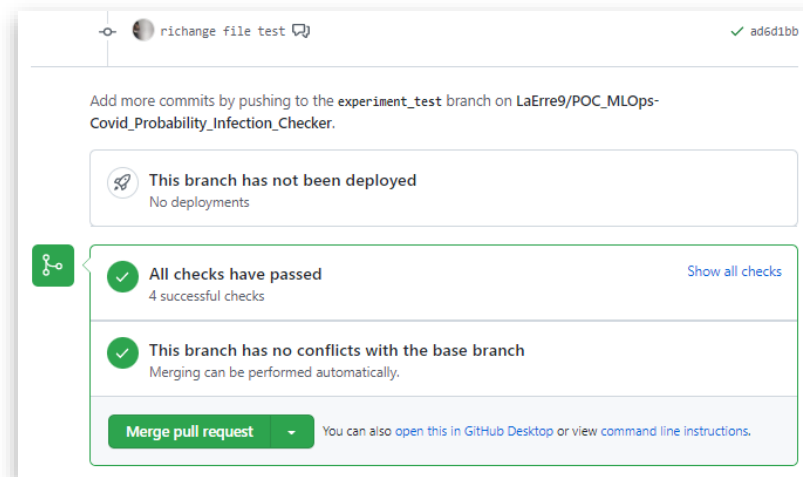


Figura 33. Check ok nel Pull Request

In definitiva, questa pratica è importante perché si ha una panoramica sul funzionamento del modello in un unico posto e se si ha un team, allora questo repository sarà accessibile a tutti. Si è dunque standardizzato il controllo sapendo che il modello ha funzionato grazie i “**check ok**” di GitHub Actions e che i test sono stati superati.

In definitiva, come detto nel paragrafo **3.3.3**, si classificano i test fatti in “*misura della prontezza complessiva del sistema ML*” per la produzione. Viene assegnato “mezzo punto” per l’esecuzione manuale della prova e assegnato un “punto pieno” se esiste un sistema in atto per eseguire automaticamente il test. Nel caso in esame, ci sono 4 sistemi di automatizzazione test appena trattati. Si somma il punteggio che è uguale a 4,5 e pertanto secondo il paragrafo è **un progetto con forte livello di test con monitoraggio automatizzato.**

4.6 Distribuzione del modello ML e dell'applicazione

Una delle pratiche più importanti di MLOps è la distribuzione del modello. Per rendere il modello usufruibile agli utenti, l'autore ha pensato di creare una pagina web che interagirà con il modello di Machine Learning che calolerà la probabilità. Nel caso in esame, si utilizza un framework Web come Flask che consente di sviluppare una WebApp per il front-end del modello. Grazie alle componenti di Flask, è possibile gestire le variabili Python in HTML in modo tale da creare una vera e propria pagina web dal quale l'utente riempirà il form e le sarà indicato il risultato. Sulla propria macchina è necessario installare Flask con `$ pip install Flask`, dopodiché si crea un file chiamato `main.py`¹⁸. Si mostra una parte del codice:

```
# importo Flask
1 from flask import Flask, render_template, request
  # creo un'app Flask
2 app = Flask(__name__)
  # import pickle per il passaggio delle informazioni presenti
  # nel file model.pkl
3 import pickle
4 file = open('model.pkl19', 'rb')
5 clf = pickle.load(file)
6 file.close()

7 @app.route('/', methods=["GET", "POST"])
```

La riga 7 è molto importante perché viene definito una route all'indirizzo home ('/') aggiungendo i metodi "GET" per recuperare gli elementi inseriti nel form e "POST" per aggiungere l'elemento. Il tutto verrà collegata alla funzione `def covid_checker()`:

```
9 def covid_checker():
    if request.method == "POST":
        myDict = request.form
        breating = int(myDict['Breathing Problem'])
```

¹⁸ Il file `main.py` è disponibile al seguente link: https://github.com/LaErre9/POC_MLOps-Covid_Probability_Infection_Checker/blob/main/main.py

¹⁹ Il file `model.pkl` è utile per memorizzare i risultati dell'algoritmo ML


```

fever = int(myDict['Fever'])
dry = int(myDict['Dry Cough'])
sore = int(myDict['Sore throat'])
running = int(myDict['Running Nose'])
asthma = int(myDict['Asthma'])
chronic = int(myDict['Chronic Lung Disease'])
headache = int(myDict['Headache'])
heart = int(myDict['Heart Disease'])
diabetes = int(myDict['Diabetes'])
hyper = int(myDict['Hyper Tension'])
fatigue = int(myDict['Fatigue '])
gastrointestinal = int(myDict['Gastrointestinal '])
abroad = int(myDict['Abroad travel'])
contact = int(myDict['Contact with COVID Patient'])
attended = int(myDict['Attended Large Gathering'])
visited = int(myDict['Visited Public Exposed places'])
family = int(myDict['Family working in Public Exposed
Places'])
wearing = int(myDict['Wearing Masks'])
sanitization = int(myDict['Sanitization from Market'])

```

In `def covid_checker()` si sono definite tutte le colonne del dataset e `myDict` che indica un dizionario Python utile per la raccolta di dati non ordinati, modificabili e indicizzati. Verranno inserite inoltre in un array `inputFeatures` che verrà caricato di valori utili per il calcolo di `infProb`:

```

inputFeatures = [breathing, fever, dry, sore, running, asthma,
chronic, headache, heart, diabetes, hyper, fatigue,
gastrointestinal, abroad, contact, attended, visited, family,
wearing, sanitization]

infProb = clf.predict_proba([inputFeatures])[0][1]

if infProb >= 0 and infProb <= 0.50:
    str1 = "frase da passare."
    return render_template('show.html', inf =
round((infProb*100), 0), text = str1)

elif infProb > 0.50 and infProb <= 0.75:

```

```
str2 = "frase da passare."  
return render_template('show.html', inf =  
round((infProb*100), 0), text = str2)  
  
elif infProb > 0.75 and infProb <= 1:  
str3 = "frase da passare."  
return render_template('show.html', inf =  
round((infProb*100), 0), text = str3)  
  
return render_template('index.html')
```

L' autore ha deciso di suddividere i risultati in tre range in modo tale da aggiungere al risultato tre commenti in base al risultato ottenuto dall'utente che ha compilato il form. Da notare la funzione `render_template()` che 'renderizza' la pagina HTML che dovrà essere visualizzata ed inoltre accetta una serie di parametri utili per passare dei valori dal codice Python al codice HTML.

Nel caso in esame, ad esempio, l'autore ha renderizzato la pagina 'show.html' e passato come parametri il valore della probabilità 'inf' e poi 'text' per il testo. In HTML, per utilizzare i parametri passati nella funzione `render_template()`, si usa la notazione a doppia parentesi graffa (i file sono trattati nell'appendice):

```
<p> La probabilità di infezione del paziente è {{inf}}%.  
{{text}} </p>
```

L'esecuzione del codice indicherà poi di accedere al link <http://127.0.0.1:5000/> oppure localhost:5000 e visualizzare sul browser quanto fatto. Ovviamente questa non è una buona soluzione per la distribuzione, poiché c'è la necessità di rendere la Web App disponibile a tutti. È possibile farlo con una piattaforma cloud che consente di creare, fornire, monitorare e distribuire una Web App: **Heroku** (piano di distribuzione free). Viene pertanto installato Heroku su Windows (disponibile anche per Mac e Linux) e creato l'account Heroku. Viene installato Gunicorn attraverso `$ pip install gunicorn`. Gunicorn è un server WSGI http Python compatibile con il framework Flask. Per comunicare ad Heroku che stiamo utilizzando il server Gunicorn, si crea un file 'Procfile' nella directory principale e si scrive:

```
web: gunicorn --bind 0.0.0.0:$PORT main:app
# si scrive main poichè l'implementazione di Flask è sul file
main.py
```

Inoltre, ad Heroku si deve indicare i requisiti che il file `main.py` e `Procfile` hanno bisogno. Lo si fa semplicemente con `$ pip freeze > requirements.txt`²⁰. Per procedere alla creazione della web app è possibile usare le righe di comando utilizzando opportunamente i comandi di Heroku (Heroku CLI), oppure usare la dashboard di Heroku online. L'autore userà la dashboard per la dimostrazione. Nella dashboard di Heroku, si clicca 'Create new app' e si inserisce il nome dell'app e la regione di appartenenza:

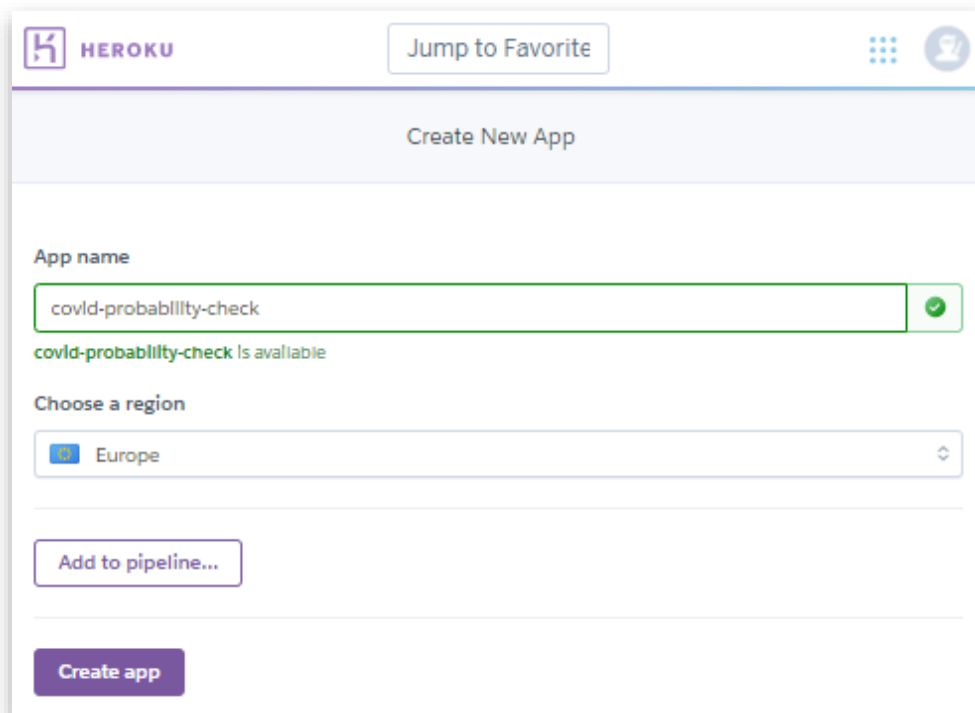


Figura 34. Configurazione di Heroku

²⁰ Il file `requirements.txt` è disponibile al seguente link: https://github.com/LaErre9/POC_MLOps-Covid_Probability_Infection_Checker/blob/main/requirements.txt

Si configurano le regole di deploy per la web appena creata:

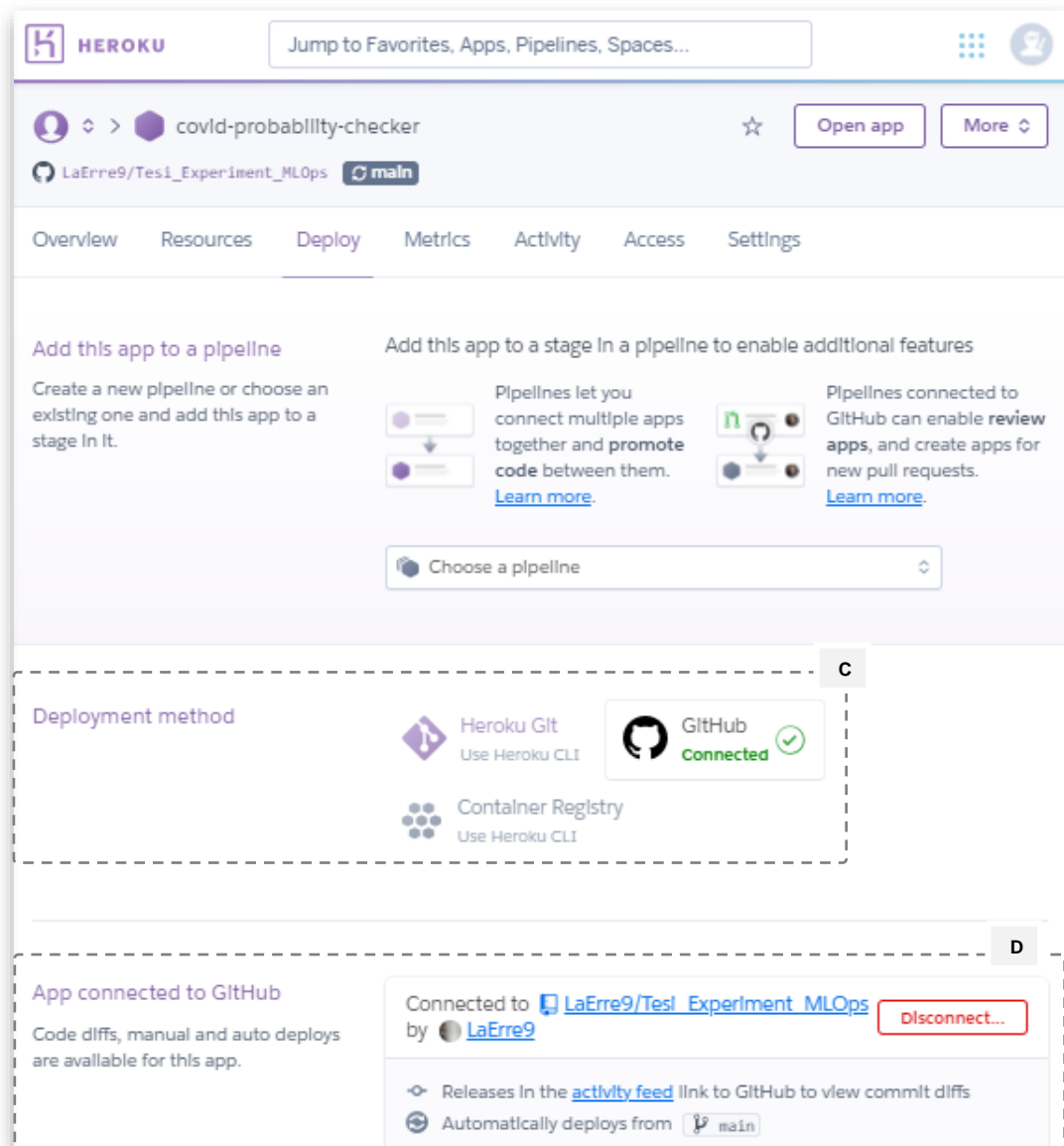
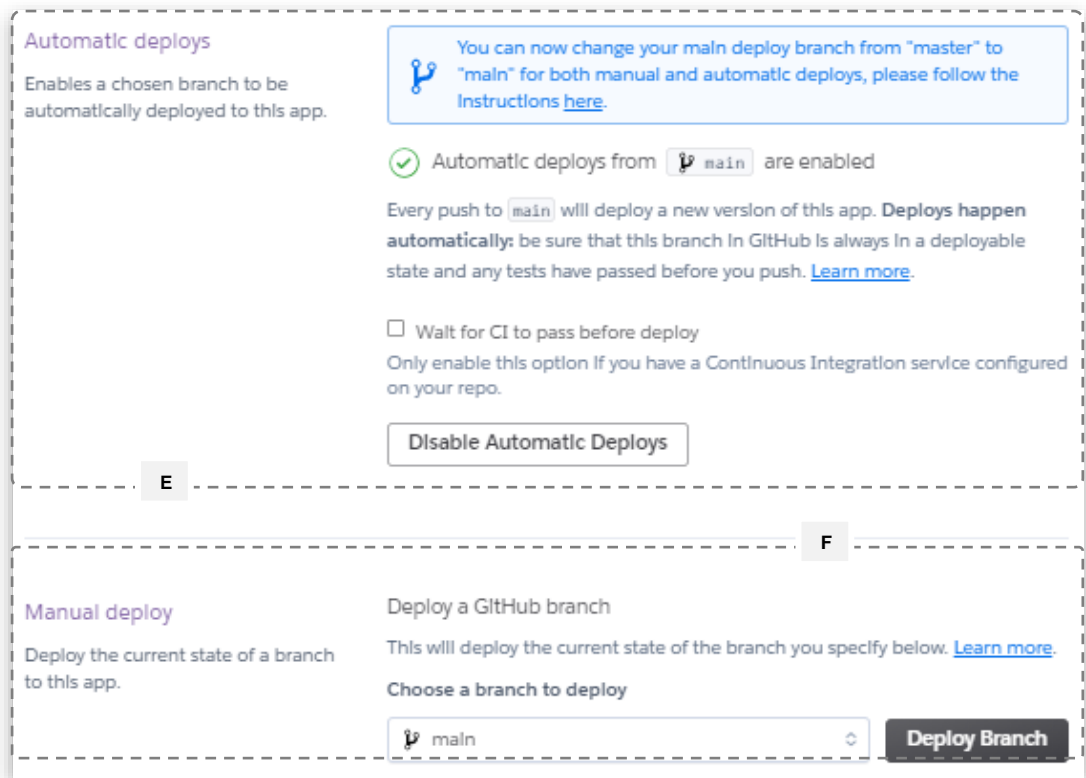


Figura 35. Configurazione deployment della Web App

(C). Si sceglie il metodo di distribuzione, nel caso in esame GitHub (**Connected**).

(D). Si connette il repository in cui sono memorizzati i file della Web App e il Profilo del GitHub connesso.



The screenshot shows the Heroku deployment configuration interface. It is divided into two main sections: 'Automatic deploys' and 'Manual deploy'.

Automatic deploys: This section is labeled with a dashed box and the letter 'E'. It contains a title 'Automatic deploys' and a description 'Enables a chosen branch to be automatically deployed to this app.' A blue callout box states: 'You can now change your main deploy branch from "master" to "main" for both manual and automatic deploys, please follow the [Instructions here](#).' Below this, a green checkmark icon indicates 'Automatic deploys from `main` are enabled'. A note says 'Every push to `main` will deploy a new version of this app. **Deploys happen automatically:** be sure that this branch in GitHub is always in a deployable state and any tests have passed before you push. [Learn more](#).' There is an unchecked checkbox 'Wait for CI to pass before deploy' with a note 'Only enable this option if you have a Continuous Integration service configured on your repo.' At the bottom of this section is a button 'Disable Automatic Deploys'.

Manual deploy: This section is labeled with a dashed box and the letter 'F'. It contains a title 'Manual deploy' and a description 'Deploy the current state of a branch to this app.' It also has a description 'Deploy a GitHub branch' and 'This will deploy the current state of the branch you specify below. [Learn more](#).' Below this is a label 'Choose a branch to deploy' and a dropdown menu showing 'main'. To the right of the dropdown is a button 'Deploy Branch'.

Figura 36. Configurazione Automatic Deploy su Heroku

(E). Si abilita l'*Automatic Deploy* in quanto Heroku distribuisce tutti i push del ramo indicato. Pertanto ad ogni aggiornamento che verrà fatto sul branch di produzione del repository verrà aggiornata in automatico anche la Web App.

(F). L'ultimo passaggio, distribuzione del Branch scelto (main nel caso in esame). Si attiverà un Activity che ci indicherà il susseguirsi delle operazioni di deployment che Heroku sta facendo:



The screenshot shows the Heroku Build Log interface. At the top, it says 'Activity Feed > Build Log' and 'ID 162e2925-bf67-4071-9917-4bb9b8203cc4'. The main area contains a log of build steps: 'Building on the Heroku-20 stack', 'Using buildpack: heroku/python', 'Python app detected', 'No Python version was specified. Using the same version as the last build: python-3.9.6', 'No change in requirements detected, installing from cache', 'Using cached install of python-3.9.6', 'Installing pip 20.2.4, setuptools 47.1.1 and wheel 0.36.2', 'Installing SQLite3', 'Installing requirements with pip', 'Discovering process types', 'Procfile declares types -> web', 'Compressing...', 'Done: 158.7M', 'Launching...', 'Released v5', and a URL 'https://covid-probability-checker.herokuapp.com/ deployed to Heroku'. At the bottom, it says 'Build finished'.

Figura 37. Build Log su Heroku

Nel frattempo, su GitHub, nella sezione 'Environments' si nota l'attività di deployment:

- **Active**, se la Web App è pronta per l'utilizzo
- **Pending**, se c'è stato un push nel repository, verrà aggiornata l'applicazione alle ultime modifiche fatte.
- **Inactive**, inattiva.

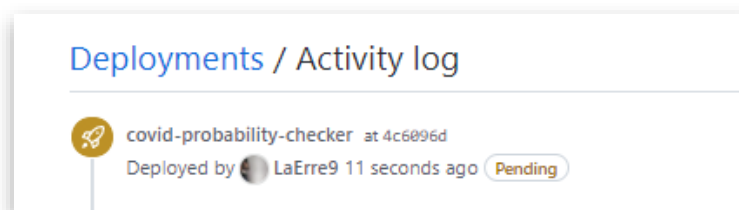



Figura 38. Pending deployment su GitHub

Appena sarà attiva, sarà possibile collegarsi al link indicato nel Build Log (Figura 37): <https://covid-probability-checker.herokuapp.com/>²¹

²¹ Si riporta il lettore ad approfondire la Web App nell'appendice.

SARS-CoV-2 (COVID-19) Home About
Probability Infection Checker



Coronavirus Probability Infection Checker

Indica **"Yes"** se presenti i seguenti sintomi, malattie o informazioni utili per il calcolo della possibile probabilità di infezione al virus SARS-CoV-2 (COVID-19).

Sintomi

Presenta questi sintomi?

Breathing Problem (*Problema di respirazione*)
Difficoltà respiratorie, respiro affannoso, respiro pesante, sensazione di respiro incompleto


Fever (*Febbre*)
Temperatura corporea > 37.5 °C (> 99.5 °F)

Dry Cough (*Tosse secca*)
Tosse secca, irritativa, persistente, non accompagnata dalla presenza di secrezioni catarrali

Sore throat (*Mal di gola*)
Dolore alla gola, difficoltà a deglutire

Running Nose (*Naso che cola*)
Perdita eccessiva di muco dal naso, o dai seni paranasali, verso l'esterno o verso la gola

Asthma (*Asma*)
Fiato corto dopo piccolo sforzo fisico, tosse frequente e raschiamento della gola



Scannerizza il **QRCode** per accedere alla **Web App**

Figura 39. Web App Covid Probability Infection Checker

L'*Automatic Deploy* è il punto forte di Heroku, in quanto soddisfa la pratica di distribuzione di MLOps (Continuous Deploy) poiché per ogni push sul branch di produzione (main) distribuirà una nuova versione dell'app. In questo modo, si è sempre sicuri che la versione in produzione abbia superato i test automatici di GitHub ed è pronta all'utilizzo dopo ogni aggiornamento/modifica. Infatti, essendo che il repository GitHub è stato configurato per il Continuous Integration è possibile attendere il passaggio della CI prima della distribuzione.

4.6.1 Release su GitHub

Per la consegna continua (Continuous delivery) del modello su GitHub è possibile creare delle Releases.

Su GitHub, si va alla pagina principale del repository e a destra della pagina (G), fare click su **'Releases'** o **'Latest Release'**.

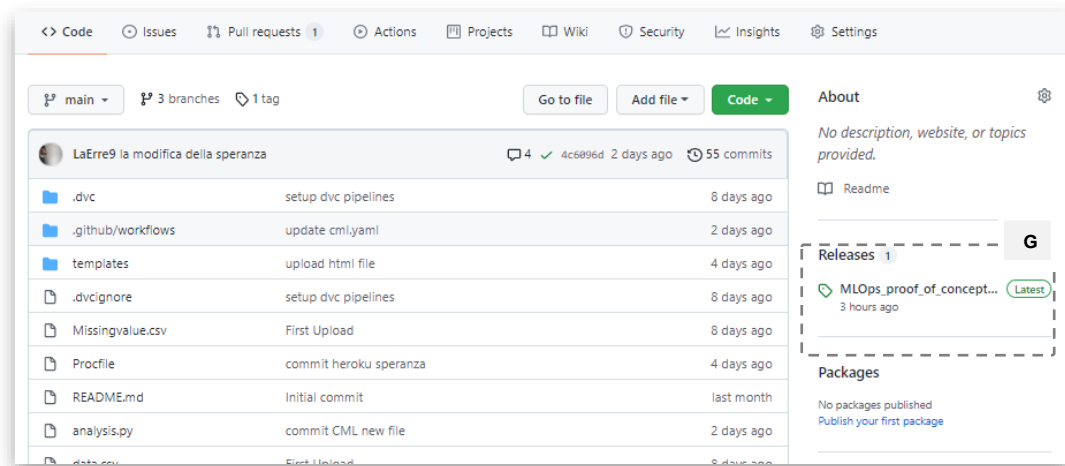


Figura 40. Release su GitHub

Se già creata, fare click **'Draft Release'** e creare una nuova Release

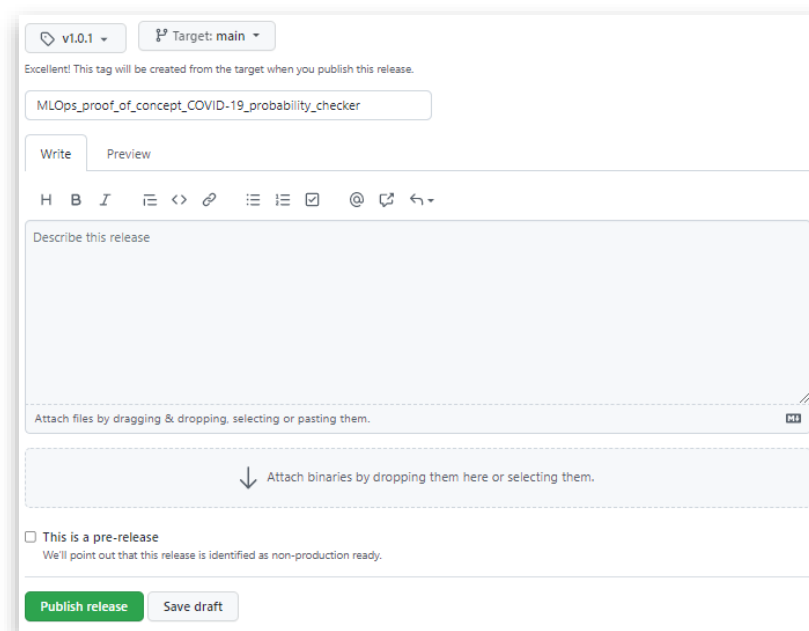


Figura 41. Creazione di una release su GitHub

Fare click su “Publish Release” e successivamente, verranno attivate le azioni presenti nel repository. Dopo la creazione della release è possibile modificarla, cancellarla, visualizzare la versione e l’identificativo dell’ultimo commit della release.

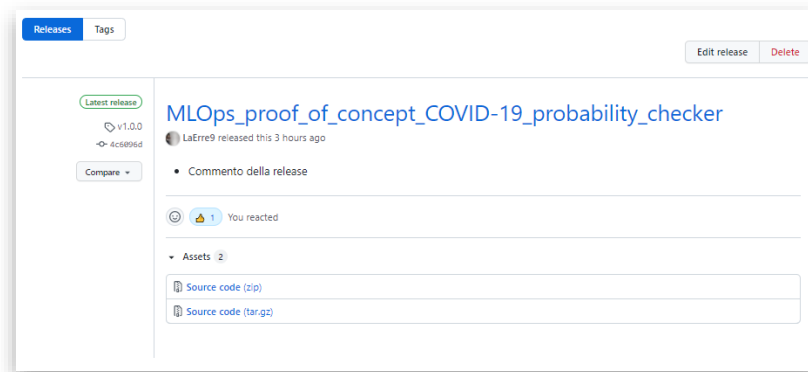


Figura 42. Archivio release su GitHub

È possibile automatizzare la release creando un file *auto_release.yaml*²² (GitHub Actions):

```
# Name of the workflow
name: Release

# Run ogni volta che viene fatto il commit tag con "v" (e.g.
# "v0.1.4")
on:
  push:
    tags:
      - "v*"

# Crea automaticamente una release GitHub, con i dettagli
# della release specificati (i relativi commit)
jobs:
  release:
    name: "Release"
    runs-on: "ubuntu-latest"
    steps:
      - uses: "marvinpinto/action-automatic-releases@latest"
```

²² Il file *auto_release.yaml* è disponibile al seguente link: https://github.com/LaErre9/POC_MLOps-Covid_Probability_Infection_Checker/blob/main/.github/workflows/auto_release.yaml

```
with:
  repo_token: "${{ secrets.GITHUB_TOKEN }}"
  prerelease: false
  title: "POC_MLOps_COVID-
19_probability_infection_checker"
```

Si attiverà quando il commit è seguito da:

```
$ git commit -m "commit funzionante", commit o merge commit
$ git push origin main
Ad ogni git tag si azionerà auto_release.yaml
#importante aggiungere un tag del tipo "v.X.Y.Z"
$ git tag v1.2.3
$ git push origin v1.2.3
```

Dopo aver fatto tutti i check del caso, GitHub mostrerà l'ultima release rilasciata indicando i Commits fatti e da chi l'ha fatto. Inoltre è possibile anche editare e compararla con la release precedente. Si nota come la release è rilasciata da GitHub Actions (G)

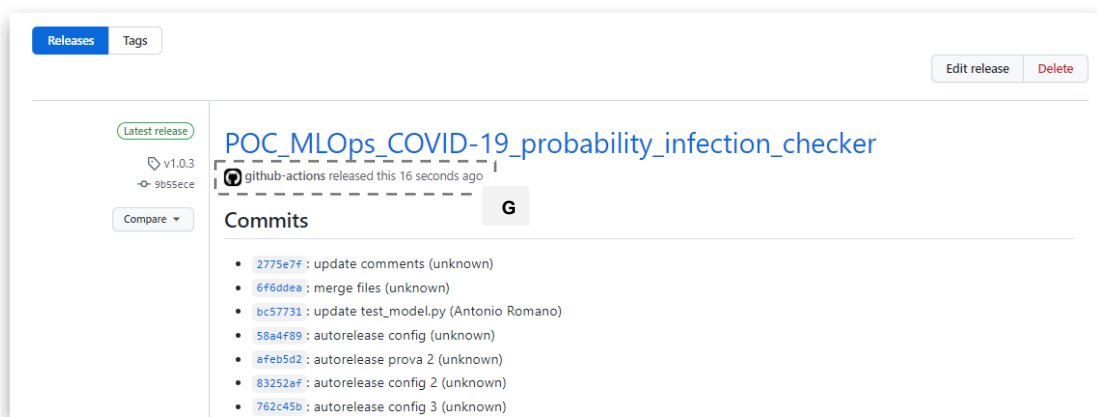


Figura 43. Release automatico su GitHub

4.7 Continuous Integration per il modello ML

Come abbiamo già detto, l'integrazione continua è una delle idee adottate da DevOps, ovvero quella pratica che dopo una modifica al codice, si riceve un feedback rapido e come questa modifica abbia influenzato il progetto finale.

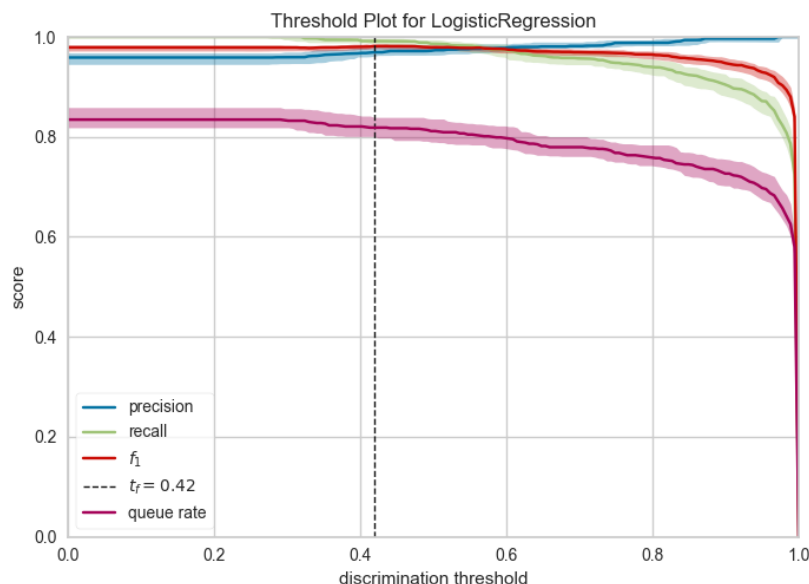
Anche in questo caso viene utilizzato GitHub Actions. Pertanto, l'analisi viene fatta

nel file *myTraining.py*:

Per visualizzare il comportamento del modello nel tempo, si utilizzano i visualizzatori di classificazione. Per fare questo c'è la necessità di installare la libreria **yellowbrick** e importare *DiscriminationThreshold* (soglia di discriminazione)

```
from yellowbrick.classifier import DiscriminationThreshold
# Report dei test score di Logistic Regression
train_score = clf.score(X_train, Y_train) * 100
test_score = clf.score(X_test, Y_test) * 100
# Scrittura dei punteggi nel file score.txt
with open("score.txt", 'w') as outfile:
    outfile.write("Training variance explained:
%2.1f%%\n" % train_score)
    outfile.write("Test variance explained:
%2.1f%%\n" % test_score)

# Visualizzatore
visualizer_report = DiscriminationThreshold(clf)
visualizer_report.fit(X_train, Y_train)
visualizer_report.score(X_test, Y_test)
visualizer_report.show("report.png")
```



Da notare, in breve, alcuni parametri che il plot fornisce: la *Precision*, definita come la riduzione del numero di falsi positivi, il *Recall*, il punteggio *F1* che è definito come

la media armonica tra Precision e Recall e Il *queue rate* che descrive la percentuale di istanze che devono essere riviste.

Anche in questo caso si utilizza GitHub Actions per la configurazione del sistema di Continuous Integration per ricostruire automaticamente il modello e rivalutarlo nuovamente ad ogni modifica. Per farlo, si crea nella directory **.github/workflows** il file *cml.yaml*²³

```
# Nome del flusso di lavoro
name: covid-cml
# Trigger che si attiva ogni volta che il repository riceve un
push
on: [push]
jobs:
  run:
    # flusso eseguito su macchina ubuntu
    runs-on: ubuntu-latest
    # container
    container: docker://dvcorg/cml-py3:latest
    steps:
      - uses: actions/checkout@v2
      - name: cml_run
        env:
          REPO_TOKEN: ${{ secrets.GITHUB_TOKEN }}
        run: |
          # Ad ogni push GitHub Actions esegue:
          pip install -r requirements.txt
          python myTraining.py

          # Stampa degli score su report.md
          echo "## Model score" > report.md
          cat score.txt >> report.md

          # Stampa del report come immagine
          echo "## Data visual" >> report.md
          cml-publish report.png --md >> report.md
          # Stampa dei report nel pull request
          cml-send-comment report.md
```

²³ Il file *cml.yaml* è disponibile al seguente link: https://github.com/LaErre9/POC_MLOps-Covid_Probability_Infection_Checker/blob/main/.github/workflows/cml.yaml

Si caricano gli aggiornamenti fatti sul branch `experiment_02` in modo da poter fare le sperimentazioni del caso e creare un pull request dedicato al monitoring del modello nel tempo. Si è fatto un esempio fittizio aggiungendo una tolleranza all'algoritmo di apprendimento: `LogisticRegression(tol=1100)`.

Dunque dopo il `$ git push origin experiment_02` delle modifiche al codice, si aziona **GitHub Actions** e dopo qualche minuto mostrerà quanto valutato e testato:

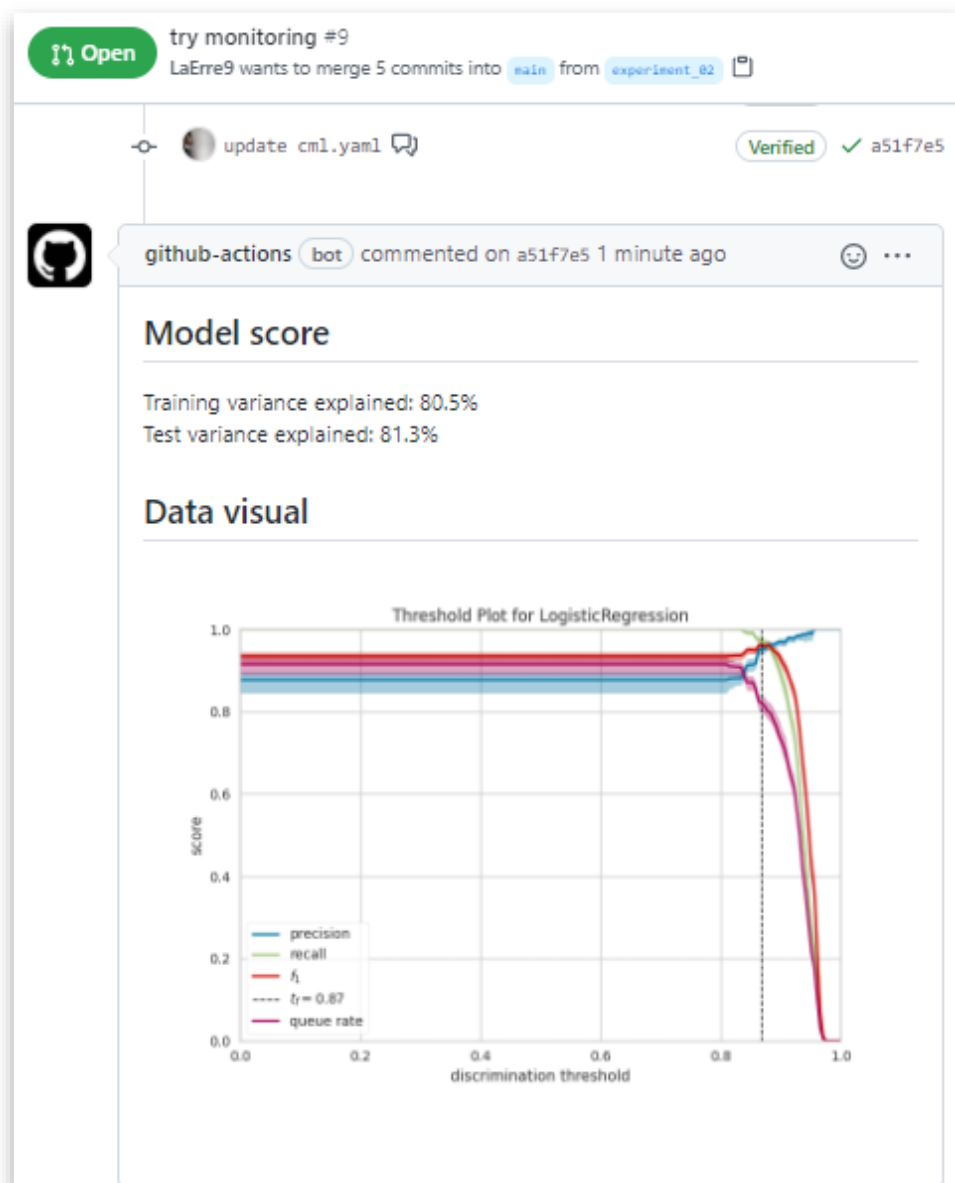


Figura 44. Risultato calcolato da GitHub Actions e inserito nel Pull Request (test errato)

Si nota subito che dal 'threshold plot' il modello non è ottimizzato al meglio poiché t_f non è compreso tra 0.4 e 0.6, pertanto, uno sviluppatore più esperto, può accorgersi

di questa anomalia visionando il pull request e andrà opportunamente a modificare il codice. In questo caso, verrà modificato in `LogisticRegression(tol=0.001)`.

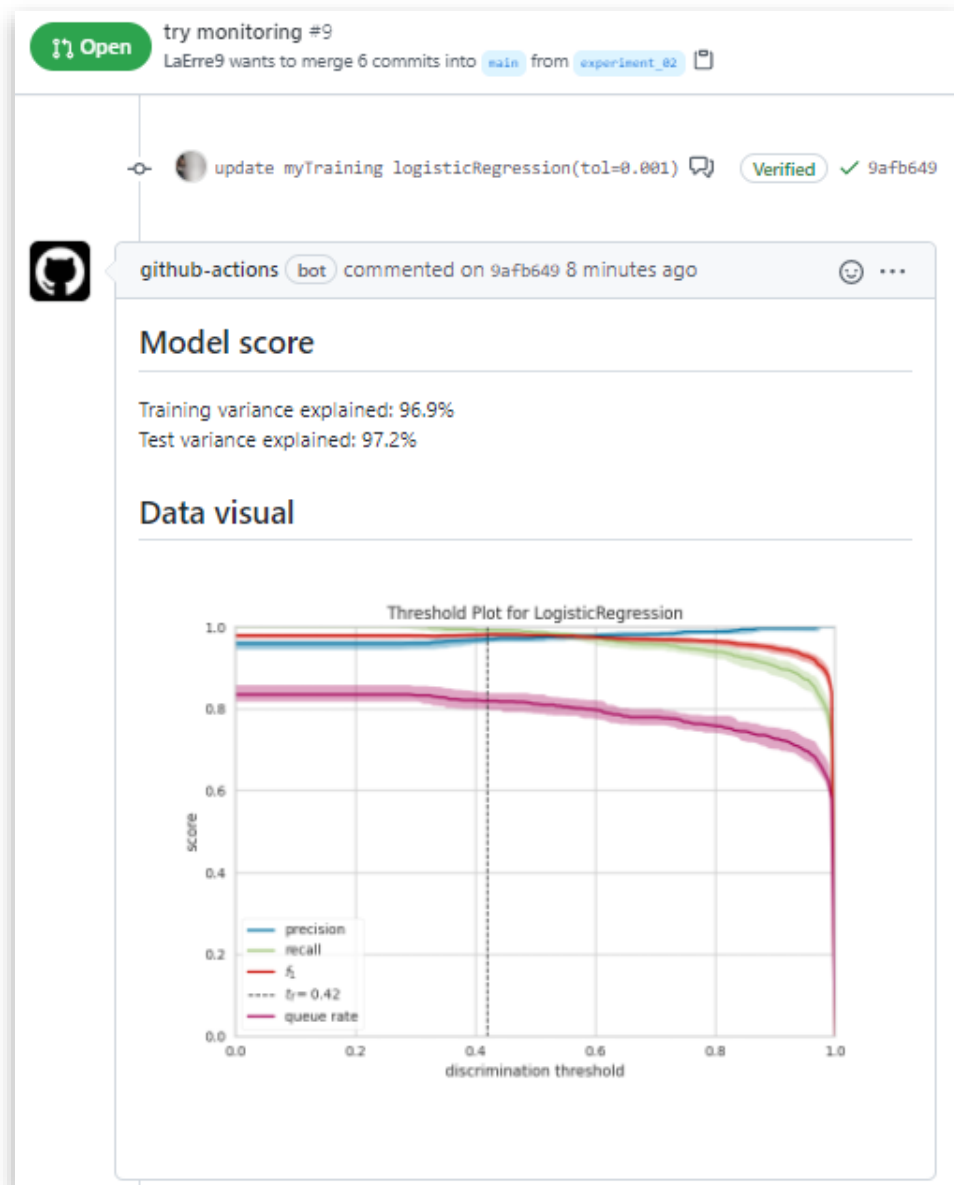


Figura 45. Risultato calcolato da GitHub Actions e inserito nel Pull Request (test esatto)

Dal model score si notano infatti miglioramenti e soprattutto nel 'threshold plot' il t_f è compreso tra il 0.4 e il 0.6. Pertanto, per attuare le modifiche fatte, si può fare il **merge pull request** e unire le modifiche nel branch di produzione 'main'.



Figura 46. Merge su GitHub

Dopo il merge, il branch di produzione sarà dunque aggiornato all'ultima versione, valutata, testata, migliorata e pronta. Grazie ad Heroku, inoltre, quest'ultimo aggiornamento verrà automaticamente distribuito e la Web App sarà aggiornata.

Accesso al repository

Per accedere al repository GitHub, utilizzare il **QR CODE**:



altrimenti, è possibile accedere al seguente link:

https://github.com/LaErre9/POC_MLOps-Covid_Probability_Infection_Checker

Conclusioni

La seguente trattazione ha mostrato come lo sviluppo del software sia passato da procedimenti fissi e manuali a pratiche dinamiche e automatiche adottando principi e tools utili per lo sviluppo del software, in particolare allo sviluppo di applicazioni che inglobano un modello di Machine Learning. Si è notato come *MLOps* sia una sorta di reincarnazione di *DevOps* per le esigenze delle aziende guidate dalla tecnologia che fanno molto affidamento sugli algoritmi di Machine Learning. Pertanto si è dimostrato attraverso il *'Proof of Concept'* come *MLOps* facilita la comunicazione tra i diversi componenti del team di sviluppo, anche con ruoli differenti come la collaborazione tra i *data scientist* che sviluppano modelli ML e gli *sviluppatori* che sviluppano il software. Grazie al tool **GitHub** e alla sezione *pull request* sarà possibile vedere tutte le modifiche che verranno fatte dai componenti del team. Grazie al Continuous Machine Learning (CML) sarà possibile ricevere un feedback immediato su quello che succede al modello e gli interessati, tempestivamente, possono risolvere problemi in caso di errori. Sempre grazie a GitHub si può avere il controllo di tutto il contenuto del repository, verificare e modificare il workflow e i file per **GitHub Actions** con il quale si possono verificare le differenze di codice tra un commit all'altro, si possono visualizzare i dati che si stanno trattando e con **DVC** tenerli sempre aggiornati su un determinato cloud, il tutto opportunamente collegati tra di loro, con l'ambiente di sviluppo, con l'infrastruttura per il training e i risultati. Tenerli tutti in un unico posto, in cui tutto il team di sviluppo può modificare e sviluppare sia una pratica *MLOps* vantaggiosa in ambito di produzione del software. Infine, grazie al supporto di **Heroku**, sarà possibile avere la Web App sempre aggiornata in maniera automatica. L'autore ha racchiuso tutta l'architettura del *'Proof of Concept'* in uno schema illustrativo in cui dimostra l'interazione tra le parti della produzione del software.

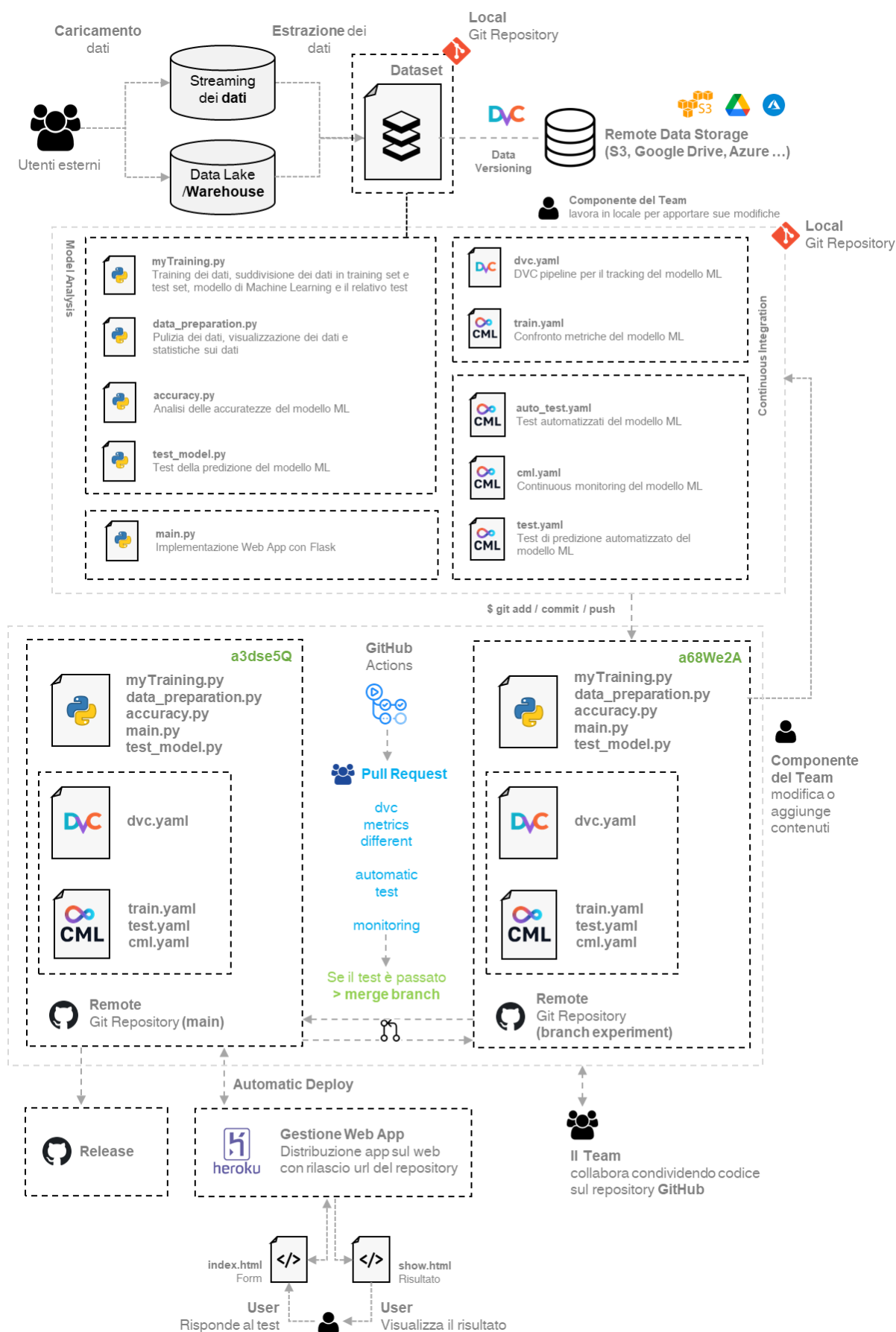


Figura 47. Flusso 'Proof of Concept' MLOps

Appendice: COVID Probability Infection Checker

Per rendere fruibile l'applicazione agli utenti, l'autore sceglie di realizzare una Web App. Un' applicazione accessibile via web per mezzo di un network, realizzando un'architettura di tipo Client-Server. Per la realizzazione della Web App sarà creato un 'index.html' per la creazione della Home Page e form per l'inserimento dei sintomi, malattie e altro che il paziente presenta e un 'show.html' per il risultato calcolato dal modello ML integrato nella Web App. Come già fatto presente nei paragrafi precedenti, per configurare il modello ML con le pagine HTML si è fatto uso del framework Flask e della creazione di un 'Procfile' utile ad Heroku per la distribuzione del file, il tutto verrà caricato sul repository GitHub. Heroku sarà connesso al *main branch* GitHub del repository di interesse ed ogni qualvolta che viene fatto un **push** verrà aggiornata l'applicazione. Il funzionamento:

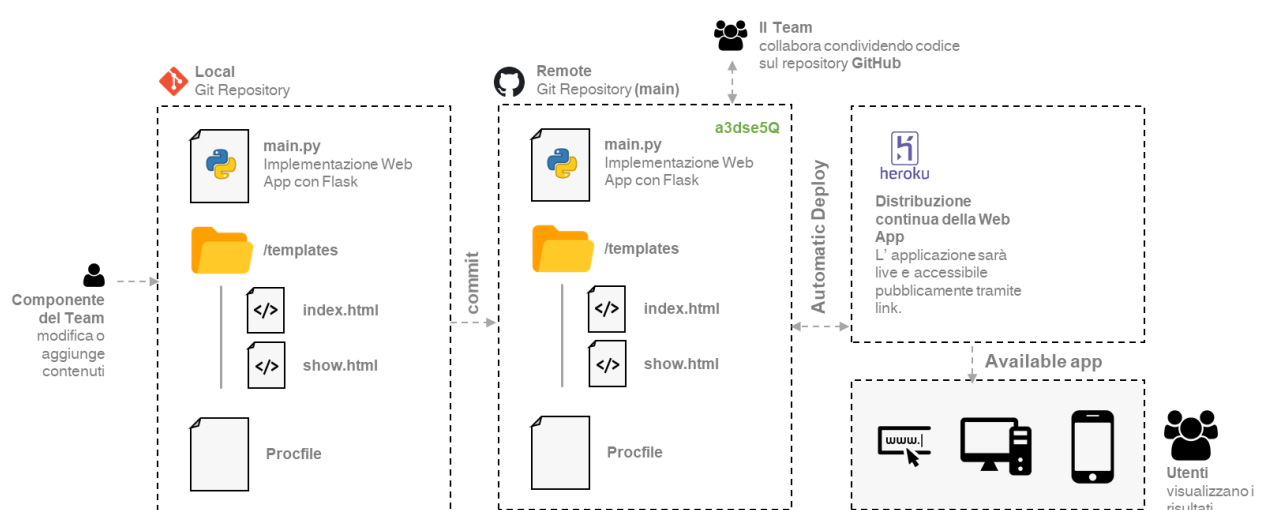


Figura 48. Flusso di funzionamento Web App e distribuzione con Heroku

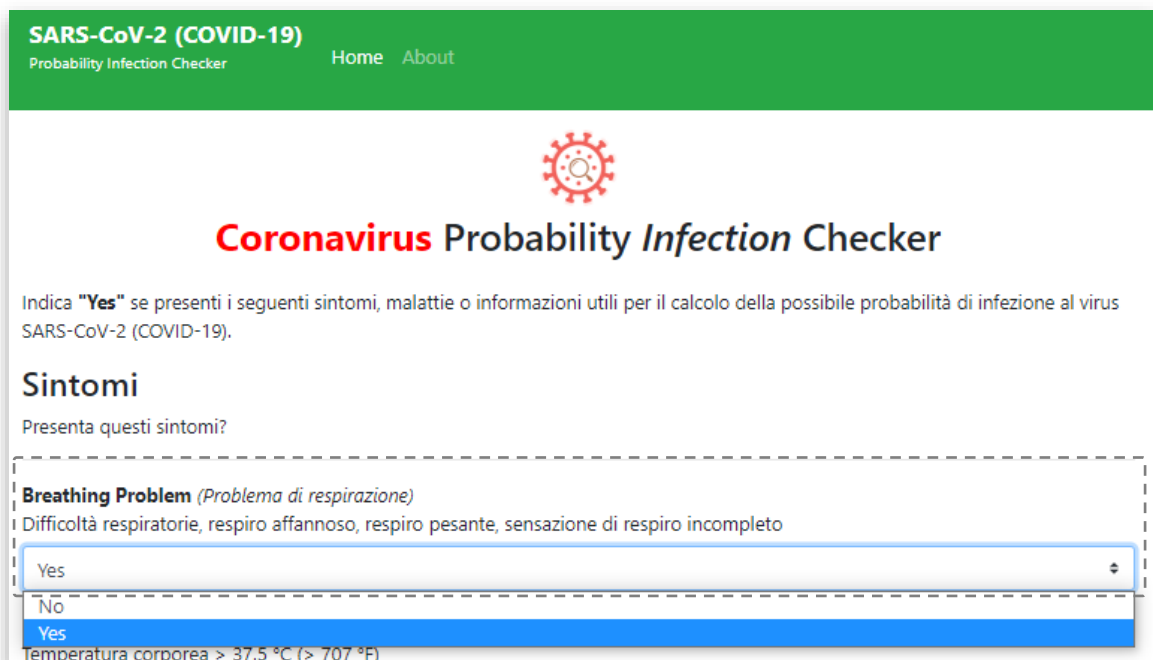
A.1 Index.html e Show.html

Si sono realizzate due pagine HTML per la realizzazione della Web App: *index.html*²⁴ in cui viene mostrato un esempio di form di risposta “Yes” e “No”:


```

1 <!--Form problema di respirazione-->
2 <div class="form-group">
3   <label for="Breathing Problem"><b>Breathing Problem</b><i>
  (Problema di respirazione)</i><br>Difficoltà respiratorie,
  respiro affannoso, respiro pesante, sensazione di respiro
  incompleto</label>
4 #Passaggio dei parametri inseriti nel form
5   <select class="custom-select" name="Breathing Problem"
6   id="Breathing Problem">
7     <option value="0">No</option>
8     <option value="1">Yes</option>
9   </select>
10 </div>

```



SARS-CoV-2 (COVID-19)
Probability Infection Checker Home About



Coronavirus Probability Infection Checker

Indica "Yes" se presenti i seguenti sintomi, malattie o informazioni utili per il calcolo della possibile probabilità di infezione al virus SARS-CoV-2 (COVID-19).

Sintomi
Presenta questi sintomi?

Breathing Problem (Problema di respirazione)
Difficoltà respiratorie, respiro affannoso, respiro pesante, sensazione di respiro incompleto

Yes

No

Yes

temperatura corporea > 37.5 °C (> 707 °F)

Figura 49. Form in *index.html*

²⁴ Il file *index.html* è disponibile al seguente link: https://github.com/LaErre9/POC_MLOps-Covid_Probability_Infection_Checker/blob/main/templates/index.html

Per la pagina *show.html*²⁵ viene mostrato come vengono trattati i parametri Python

```

1 <div class="container">
2 <center></center>
  <h2 class="text-center" style="margin: 25px;"><b><span
    style="color: red;">Coronavirus </span></b>Probability
    <i>Infection</i> Checker</h2>
5 <h3>Il risultato</h3>
  <hr>
6 <p> La probabilità di infezione del paziente è {{inf}}%.
  {{text}}</p>

```

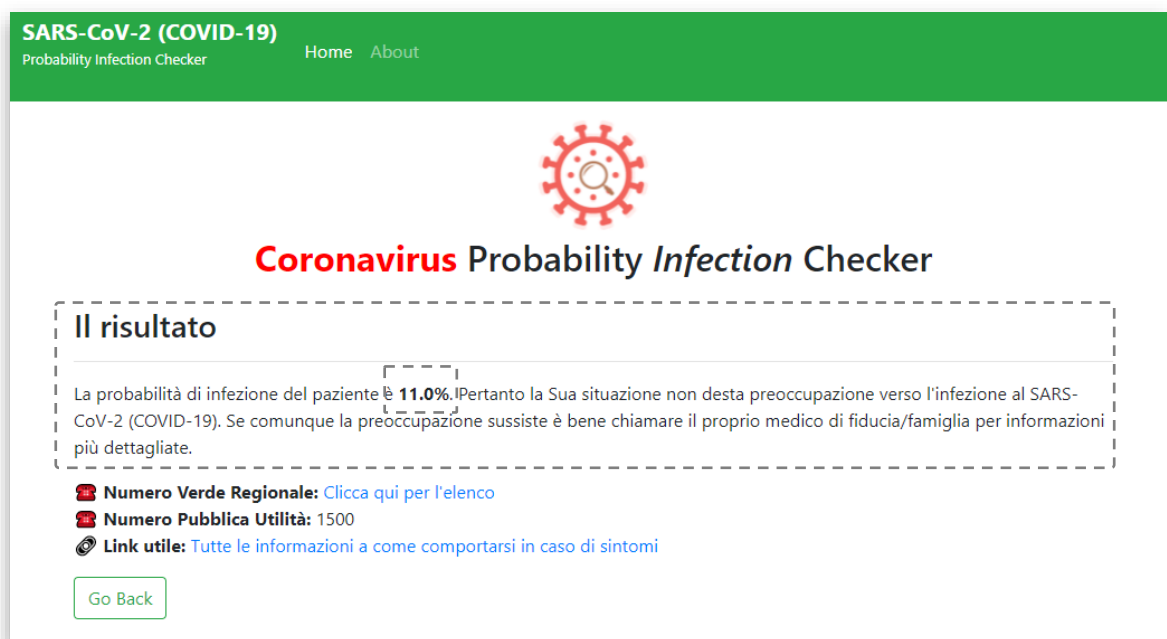


Figura 50. Risultato in *show.html*

Il funzionamento della Web App: l'utente che accede alla Web App risponde al test, quando clicca *'Calculate the probability of infection at COVID-19'*, i valori inseriti e passati da *main.py* a *myTraining.py* andranno in pasto al modello ML che a sua volta calcolerà la probabilità e invierà i dati di nuovo al *main.py* che a sua volta verranno inviati a *show.html* che mostrerà all'utente i risultati e il messaggio

²⁵ Il file *show.html* è disponibile al seguente link: https://github.com/LaErre9/POC_MLOps-Covid_Probability_Infection_Checker/blob/main/templates/show.html

testuale. Ecco uno schema grafico che spiega quanto spiegato:

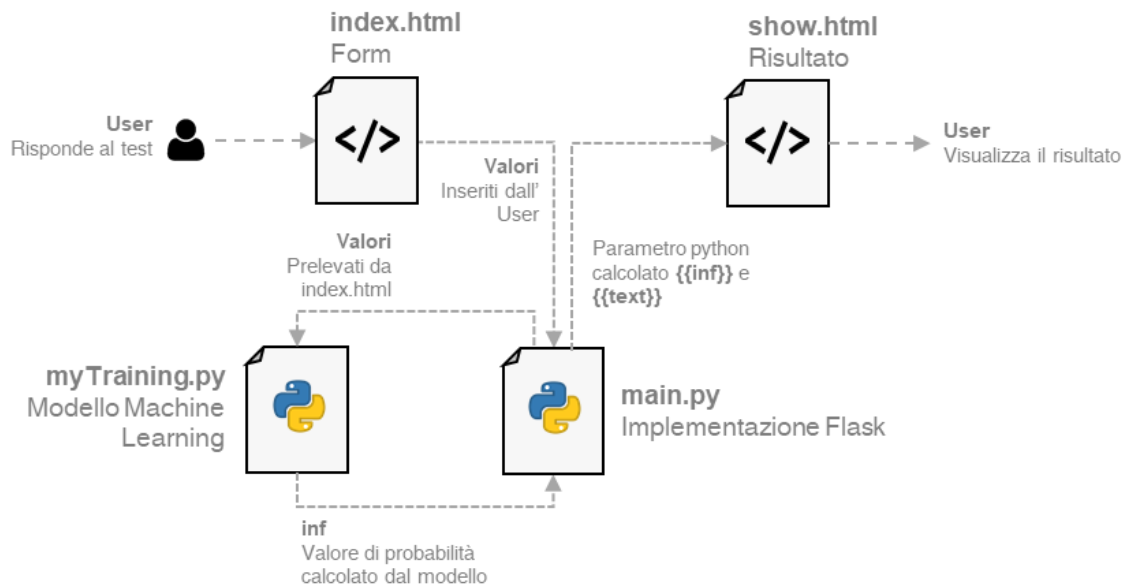


Figura 51. Funzionamento Web App e passaggio parametri tra index.html e show.html

A.2 Casi d'uso

Per dimostrare che la web app funzioni e il modello di Machine Learning fornisca una probabilità simile alla realtà, si prova a lavorare su due casi reali.

ATTENZIONE: L'applicazione realizzata dall'autore è solo a scopo dimostrativo, pertanto la probabilità calcolata può risultare incerta giacché, come già detto in precedenza, il dataset è datato e non aggiornato e i sintomi del virus mutano e variano ogni giorno.

Il primo caso è di un *paziente X* che presenta dei sintomi da COVID-19 ma dopo un tampone molecolare non è risultato positivo al virus.

Pertanto, il *paziente X* presenta: mal di gola (sore throat), febbre a 38° C (fever), naso che cola (running nose), mal di testa (headache), diarrea (gastrointestinal problem). Il paziente ha frequentato luoghi pubblici (visited public exposed places) ed indossato la mascherina (wearing masks). Inserendo i dati nel form, il risultato che calcolerà il modello ML sarà:

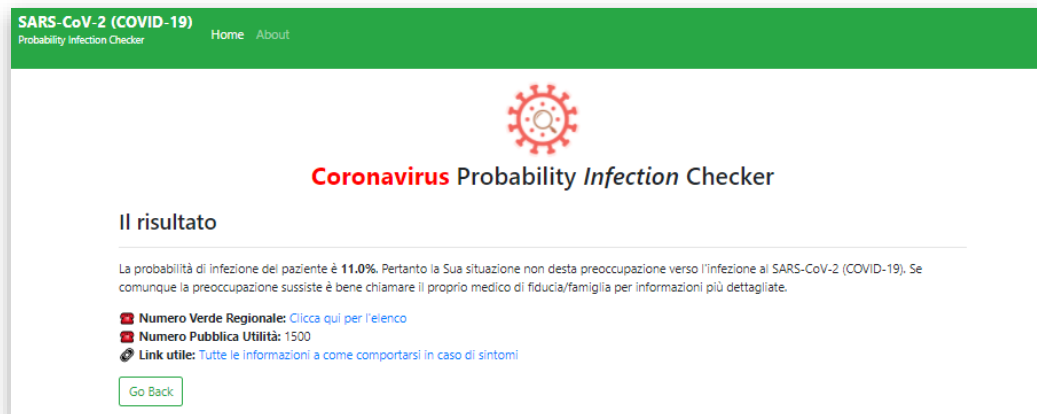


Figura 52. Risultato calcolato da un caso risultato negativo

Pertanto, anche per il 'Coronavirus Probabilty Infection Checker' il *paziente X* non è stato infetto dal virus. Il secondo caso è di un *paziente Y* che presenta sintomi da COVID-19 e dopo il tampone molecolare è risultato positivo al virus. Il seguente paziente presenta come sintomi: difficoltà respiratorie (breathing problem), febbre a 38,5 °C (fever), tosse secca (dry cough), naso che cola (running nose), mal di testa (headache) e spossatezza (fatigue). Inoltre, il paziente ha avuto contatti con una persona risultata positiva al tampone molecolare. Ha visitato luoghi pubblici e la famiglia lavora in luoghi pubblici esposti. Ha indossato la mascherina e ha frequentato luoghi sanificati. Inserendo i dati nel form, il risultato che calcolerà il modellino di ML sarà:

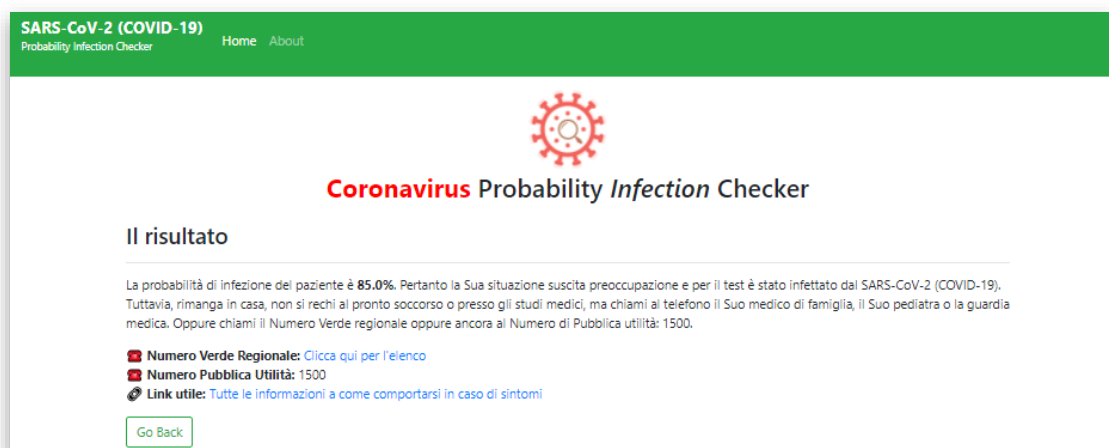


Figura 48. Risultato calcolato da un caso risultato positivo

Dunque anche per il test, il *paziente Y*, molto probabilmente è stato infetto dal virus.

A.3 Evoluzione futura

La Web App “Covid Probability Infection Checker” è un’applicazione che può essere usata dalle organizzazioni sanitarie, o anche dai pazienti interessati per capire, in base ai suoi malesseri o comportamenti di essere infetto al virus. Inoltre, importante sviluppo futuro, è provare ad evitare il tampone o test sierologico che a volte risultano fastidiosi ai pazienti stessi. L’ applicazione potrà essere aggiornata e supportata ad aggiunte di nuovi sintomi, come ad esempio, negli ultimi mesi è stato scoperto statisticamente che il sintomo che differisce maggiormente il COVID-19 alla classica influenza sia la perdita del gusto e dell’olfatto. Pertanto sarà necessario aggiornare il dataset, tutti i modelli e risultati che il modello ML calcola. Grazie alle pratiche MLOps sarà dunque possibile tenere sempre al passo il modello ML a nuove scoperte ed essere supportato per funzionare sempre ed in modo efficiente e sicuro.

Un ulteriore implementazione futura potrà essere la possibilità degli utenti di aggiungere i propri sintomi e definire se sono stati risultati positivi al tampone o al test sierologico cosicché il dataset diventerà più vasto e il modello di machine learning sarà più preciso a calcolare la predizione di un determinato paziente.

Questo esempio trattato, può essere generalizzato per altri virus, influenze, malattie e potrà essere di supporto ai dottori per automatizzare la diagnostica del paziente e riuscire ad individuare tempestivamente le problematiche che un paziente può manifestare personalizzando al meglio la ricetta medica.

Bibliografia

- [1] Wikipedia, “Modello di sviluppo del software”, https://it.wikipedia.org/wiki/Modello_di_sviluppo_del_software, consultazione il 2 agosto 2021
- [2] Wikipedia, “Metodologie Agili”, https://it.wikipedia.org/wiki/Metodologia_agile, consultazione il 2 agosto 2021
- [3] Agile Way, “Metodologia Scrum”, <https://www.agileway.it/scrum-metodologia-agile/>, consultazione il 3 agosto 2021
- [4] Amazon Web Services, “What is DevOps”, <https://aws.amazon.com/it/devops/what-is-devops/>, consultazione il 3 agosto 2021
- [5] ITPedia, “DevOps vs Scrum”, <https://it.itpedia.nl/2019/02/13/devops-en-scrum-een-winning-team/#:~:text=DevOps%20ha%20molti%20team%20che,minimo%20delle%20interuzioni%20dell'attivit%C3%A0>, consultazione il 4 agosto 2021
- [6] Andrea Minini, “Machine Learning”, <https://www.andreaminini.com/ai/machine-learning/>, consultazione il 5 agosto 2021
- [7] Blog Osservatori, “Modellazione predittiva”, https://blog.osservatori.net/it_it/modellazione-predittiva-come-funziona, consultazione il 5 agosto 2021
- [8] Freecodecamp, “MLOps Explained”, <https://www.freecodecamp.org/news/what-is-mlops-machine-learning-operations-explained>), consultazione il 6 agosto 2021
- [9] MLOps, “Principi MLOps”, <https://ml-ops.org/content/mlops-principles>, consultazione il 6 agosto 2021

- [10] Reply, “MLOps: Machine Learning Operations”, <https://www.reply.com/machine-learning-reply/it/mlops>, consultazione il 7 agosto 2021
- [11] Microsoft docs, “Guida DevOps ML”, <https://docs.microsoft.com/it-it/azure/cloud-adoption-framework/ready/azure-best-practices/ai-machine-learning-mlops>, consultazione il 7 agosto 2021
- [12] Google Cloud, “MLOps: Continuous delivery and automation pipelines in machine learning”, <https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning>), consultazione il 7 agosto 2021
- [13] Playlist MLOps Tutorial – Elle O’Brien, DVCorg, “CI for ML, Track ML models with Git & GitHub Actions, Automated Testing for Machine Learning” https://www.youtube.com/watch?v=9BgIDqAzfuA&list=PL7WG7YrwYcnDBDuCkFbcyjnZQrdskFsBz&ab_channel=DVCorg , consultazione il 23 agosto 2021
- [14] MLOps - guide, “MLOps guide”, <https://mlops-guide.github.io/Versionamento/>, consultazione il 26 agosto 2021
- [15] Neptune, “MLOps: What It Is, Why it Matters, and How To Implement It”, <https://neptune.ai/blog/mlops-what-it-is-why-it-matters-and-how-to-implement-it-from-a-data-scientist-perspective>, consultazione il 26 agosto 2021
- [16] Analyticsvidhya, “Use cases of MLOps”, <https://www.analyticsvidhya.com/blog/2021/06/will-mlops-change-the-future-of-the-healthcare-system-major-use-cases-of-mlops-in-health-care/>, consultazione il 26 agosto 2021
- [17] Ichi, “Metriche di valutazione per problemi di classificazione con l'implementazione in Python”, <https://ichi.pro/it/metriche-di-valutazione-per-problemi-di-classificazione-con-l-implementazione-in-python-178127656776387>, consultazione il 24 agosto 2021
- [18] Codemy, “Push Flask Apps To Heroku For Webhosting”, <https://youtu.be/Li0Abz-KT78>, consultazione il 25 agosto 2021
- [19] MLOps – guide, “Template”, <https://github.com/mlops-guide/mlops-template>, consultazione il 26 agosto 2021

Web App ispirata dalle seguenti idee:

[20] GitHub Repository, “*Coronavirus Probability Checker*”,
<https://github.com/Siddhant-K-code/Coronavirus-Probability-Checker>,

consultazione il 2 agosto 2021

[21] Kaggle, “*Dataset Symptoms COVID-19*”,
[https://www.kaggle.com/midouazerty/symptoms-covid-19-using-7-machine-](https://www.kaggle.com/midouazerty/symptoms-covid-19-using-7-machine-learning-98)

[learning-98](https://www.kaggle.com/midouazerty/symptoms-covid-19-using-7-machine-learning-98), consultazione il 2 agosto 2021

Ringraziamenti

A conclusione della tesi, mi è doveroso dedicare quest'ultima pagina alle persone che hanno contribuito, anche con la sola presenza, la stesura di questo lavoro.

In primis, ringrazio infinitamente i miei genitori e mio fratello, per avermi permesso di arrivare fin qui, con il supporto, i sacrifici e il lavoro. Ringrazio i miei parenti, vicini e lontani e ai miei amici di famiglia e soprattutto a chi non c'è più.

Ringrazio il Prof. Ing. Pietrantuono Roberto, per la disponibilità, per i suggerimenti e per avermi dato la possibilità di estendere una delle tante mie idee per questo lavoro.

Ringrazio tutti i miei colleghi di corso, in particolare ai miei compagni di università: Peppe, ai Salvatore, ai Francesco, Giovanni, Alberico e a tutti gli altri che hanno incrociato la loro vita con la mia lasciandomi qualcosa di buono. Non dimenticherò le risate, le ansie, la circumvesuviana e nel periodo "Covid" le videochiamate.

Voglio anche ricordare i miei "primi" colleghi dell'università, in particolare Carmine e Salvatore, tanti bei ricordi vissuti anche con loro.

Ringrazio tutti i miei amici con il quale, durante il corso della stesura della tesi e non solo, ho condiviso i momenti di spensieratezza: Roberto, Fabrizio, Chiara, Davide, Bruno, Antonella, Pasquale e a tutti gli altri ragazzi che in questi anni ho conosciuto e mi hanno lasciato un bel ricordo.

Ringrazio anche ai miei vecchi cari professori delle superiori, la loro professionalità la terrò sempre con me. Sono così tanti i ricordi e le persone che mi passano per la testa che è impossibile trovare spazio per ringraziarli, dico soltanto:

Grazie,
Antonio.