MLOps on AWS

24 Settembre 2021 - 18 min. read

Amazon SageMaker | *Continuous Deployment* | *Continuous Integration* | *Machine Learning* | *MLOps*

Quando affrontiamo i moderni problemi di Machine Learning in un ambiente AWS, c'è molto più che la tradizionale preparazione dei dati, l'addestramento del modello e le inferenze finali da considerare. Inoltre, la pura potenza di calcolo non è l'unica preoccupazione di cui dobbiamo occuparci nella creazione di una soluzione ML.

Esiste una **differenza sostanziale tra la creazione e il test di un modello di Machine Learning** all'interno di un notebook Jupyter in locale e il rilascio su un'infrastruttura di produzione in grado di generare valore aziendale.

Le complessità legate all'implementazione di un flusso di lavoro di Machine Learning nel Cloud sono chiamate gap di distribuzione e vedremo insieme in questo articolo come affrontarlo combinando velocità e agilità nella modellazione e formazione, con i criteri di solidità, scalabilità e resilienza richiesti da ambienti di produzione.

La procedura in cui ci addentreremo è simile per molti aspetti al modello DevOps per lo sviluppo software "tradizionale", e il paradigma MLOps, così chiamato, viene comunemente proposto come "un processo end-to-end per progettare, creare e gestire applicazioni di Machine Learning in modo riproducibile, testabile ed evolutivo".

Per questo motivo, man mano che ci addentreremo nei paragrafi seguenti, approfondiremo le ragioni e i principi alla base del paradigma MLOps e come si collega facilmente all'ecosistema AWS e alle migliori pratiche dell' AWS Well-Architected Framework.

Allora, iniziamo!

Perchè abbiamo bisogno del MLOps?

Come detto prima, i carichi di lavoro di Machine Learning possono essere visti essenzialmente come pezzi complessi di software, quindi possiamo ancora applicare pratiche software "tradizionali". Tuttavia, per la sua natura sperimentale, il Machine Learning mette in gioco alcune **differenze essenziali**, che richiedono un paradigma di gestione del ciclo di vita fatto su misura per le loro esigenze.

Queste differenze si presentano in tutte le varie fasi di un carico di lavoro e contribuiscono in modo significativo al divario di distribuzione di cui abbiamo parlato, quindi una descrizione generale è, quantomeno, d'obbligo:

Codice

Gestire codice nelle appliance di Machine Learning è una questione complessa. Vediamo perché!

La collaborazione sugli **esperimenti del modello tra i data scientist** non è facile come la condivisione di file di codice tradizionali: i notebook Jupyter consentono di scrivere ed eseguire codice, rendendo le operazioni git più complesse per mantenere il codice sincronizzato tra gli utenti, con **frequenti conflitti di merge**.

Gli sviluppatori devono scrivere codice per diversi sottoprogetti: **processi ETL**, **logica del modello**, **training e convalida**, **logica di inferenza** e **modelli di Infrastructure-as-Code**. Tutti questi progetti separati devono essere gestiti centralmente e adeguatamente versionati!

Per le moderne applicazioni software, esistono molte procedure **consolidate di controllo di versione** come il commit convenzionale, il branching delle funzionalità, lo squash e rebase e l'integrazione continua.

Queste tecniche, tuttavia, non sono sempre applicabili ai notebook Jupyter poiché, come affermato in precedenza, non sono semplici file di testo.

Sviluppo

I data scientists devono effettuare molte combinazioni di set di dati, funzionalità, tecniche di modellazione, algoritmi e configurazioni di parametri per trovare la **soluzione che estrae al meglio il valore aziendale**.

Il punto chiave è trovare un sistema per tenere traccia di esperimenti **riusciti** e **falliti** mantenendo la **riproducibilità** e la **riutilizzabilità del codice**. Perseguire questo obiettivo significa disporre di strumenti che consentano rapidi rollback e un monitoraggio efficiente dei risultati, meglio se con strumenti visivi.

Test

Testare un carico di lavoro di Machine Learning è **più complesso** rispetto al test di software tradizionali.

Il set di dati richiede una **convalida continua**. I modelli sviluppati dai data scientist **richiedono una valutazione continua della qualità**, la convalida del training e **controlli delle prestazioni**.

Tutti questi controlli si aggiungono ai tipici test di unità e integrazione, definendo il concetto di **Training Continuo**, necessario per evitare l'**invecchiamento del modello** e il concept drift.

Esclusivo dei flussi di lavoro di Machine Learning, il suo scopo è di attivare il retraining e servire automaticamente i modelli.

Rilascio in produzione

La distribuzione di modelli di Machine Learning nel cloud è un **compito impegnativo**. In genere richiede la creazione di varie **pipeline a più passaggi** che servono per riaddestrare e distribuire automaticamente i modelli.

Questo approccio aggiunge complessità alla soluzione e richiede l'**automazione di passaggi eseguiti manualmente dai data scientist** durante il training e la convalida di nuovi modelli nella fase sperimentale di un progetto.

È fondamentale creare procedure di retraining efficienti!

Monitoraggio in Produzione

I modelli di Machine Learning tendono a **decadere molto più velocemente rispetto al software "tradizionale"**. Possono avere prestazioni ridotte a causa di codice non

ottimale, **scelte hardware errate** nelle fasi di addestramento e inferenza e set di dati in evoluzione.

Una metodologia adeguata deve tenere conto di questo degrado; pertanto, abbiamo bisogno di un **meccanismo di tracciamento** per **riepilogare efficacemente le statistiche di un carico di lavoro**, monitorare le **prestazioni** e **inviare notifiche di allarme**.

Tutte queste procedure devono essere automatizzate e sono chiamate **Monitoraggio Continuo**, che ha anche l'ulteriore vantaggio di abilitare il Training Continuo, mediante la misurazione di soglie significative.

Vogliamo anche **applicare i rollback** quando un'inferenza del modello devia dalle soglie di punteggio selezionate il più rapidamente possibile per provare nuove combinazioni di funzionalità.

Continuous Integration e Continuous Deployment

Il Machine Learning condivide approcci simili alle pipeline di CI/CD standard delle moderne applicazioni software: controllo del codice sorgente, test di unità, test di integrazione, delivery continuo dei pacchetti applicativi.

Tuttavia, **modelli e data set richiedono interventi particolari**.

L'integrazione continua ora richiede, come detto prima, anche il test e la convalida dei dati, schemi di dati e modelli.

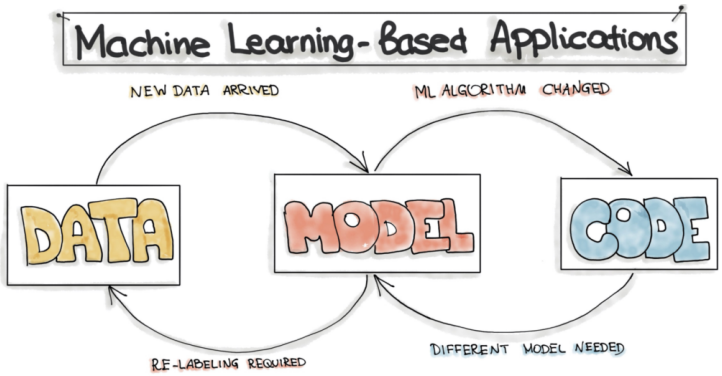
In questo contesto, la distribuzione continua deve essere progettata come una pipeline di training ML **in grado di distribuire automaticamente l'inferenza come servizio raggiungibile dal web**.

Come si può intuire, c'è molta carne al fuoco che rende la strutturazione di un progetto di Machine Learning un compito assai complesso.

Prima di introdurre il lettore alla metodologia MLOps, che pone sotto la sua ala tutti questi aspetti cruciali, vedremo come è strutturato un tipico workflow di Machine Learning, tenendo conto di quanto detto fino ad ora.

Proseguiamo insieme!

Un tipico flusso di Machine Learning in Cloud



Courtesy of https://ml-ops.org/content/end-to-end-ml-workflow

Un flusso di lavoro di Machine Learning non è pensato per essere lineare, proprio come il software tradizionale. È composto principalmente da tre livelli distinti: **dati**, **modello** e **codice** e ognuno **fornirà e recupererà continuamente feedback dagli altri**.

Quindi, mentre con il software tradizionale, possiamo dire che ogni passaggio che compone un flusso di lavoro può essere atomico e in qualche modo isolato, nel Machine Learning, questo non è del tutto vero **poiché i livelli sono profondamente interconnessi**.

Un tipico esempio è quando le modifiche al set di dati richiedono il retraining o il ripensamento di un modello. Anche un modello diverso, di solito, necessita di modifiche al codice che lo esegue.

Vediamo insieme da cosa è composto ogni Layer e come funziona.

Strato di dati

Il livello dati comprende tutte le attività necessarie per manipolare i dati e renderli disponibili per la progettazione e l'addestramento del modello: **acquisizione dei dati**, **ispezione**, **pulizia** e, infine, **preelaborazione dei dati**.

I set di Dati per problemi reali, o quantomeno realistici, possono essere nell'ordine di GB o addirittura TB, in continuo aumento, quindi abbiamo bisogno di uno spazio di archiviazione adeguato per gestire enormi data lake.

Lo storage deve essere robusto, consentire un'elaborazione parallela efficiente e integrarsi facilmente con gli strumenti per i lavori ETL.

Questo livello è il più cruciale, rappresentando l'80% del lavoro svolto in un flusso di lavoro di Machine Learning; due famose citazioni confermano questo fatto: "*garbage in, garbage out*" e "*il tuo modello è valido solo quanto i tuoi dati*".

La maggior parte di questi concetti sono prerogativa **delle buone pratiche di Data Analytics**, profondamente intrecciata con il Machine Learning, e li analizzeremo in dettaglio più avanti in questo articolo.

Strato di modello

Il livello di Modello contiene tutte le operazioni per **progettare**, **sperimentare**, **addestrare** e **convalidare** uno o più modelli di Machine Learning. I professionisti del machine learning conducono prove sui dati in questo livello, sperimentano algoritmi su **diverse soluzioni hardware** ed eseguono l'**ottimizzazione degli iperparametri**.

Questo livello è tipicamente **soggetto a frequenti modifiche dovute ad aggiornamenti sia di Dati che di Codice**, necessari per evitare il **concept drift**. Per gestire correttamente il suo ciclo di vita su larga scala, dobbiamo definire **procedure automatiche** per il retraining e la convalida.

Il livello di Modello è anche una fase in cui si verificano frequenti discussioni, tra Data Scientist e parti interessate, sulla convalida del modello, sulla solidità concettuale e sulle discordanze rispetto ai risultati attesi.

Strato di codice

Nel livello Codice, definiamo un insieme di procedure per mettere in produzione un modello, gestire le **richieste di inferenze**, archiviare i metadati di un modello, analizzare **le prestazioni complessive**, **monitorare il flusso di lavoro** (debug, logging, auditing) e **orchestrare automatismi di CI/CD/CT/CM**.

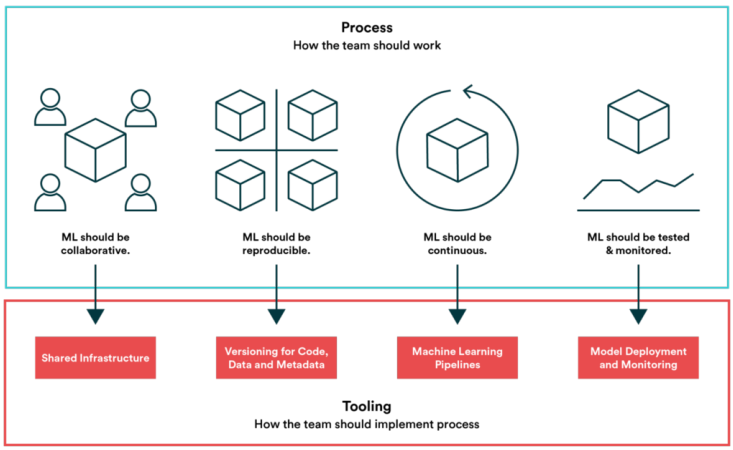
Un buon livello Codice consente un **modello di feedback continuo**, in cui il modello si evolve nel tempo, tenendo conto dei risultati delle inferenze in corso.

Tutti e tre questi livelli sono gestiti da "sub-pipeline", che si sommano tra loro per formare una "macro-pipeline" nota come Machine Learning Pipeline.

Progettare, costruire ed eseguire automaticamente questa pipeline, riducendo il deployment gap nel processo, è il nucleo del paradigma MLOps.

MLOps con AWS: i quattro pilastri

Il paradigma MLOps mira a rendere lo sviluppo e il mantenimento dei flussi di lavoro di Machine Learning **semplici** ed **efficienti**. La comunità di data science generalmente concorda sul fatto che non si tratta di un'unica soluzione tecnica, ma di una serie di best practice e principi guida sull'apprendimento automatico.



Courtesy of https://valohai.com/mlops/

Un approccio MLOps coinvolge operazioni, tecniche e strumenti, che possiamo raggruppare in **quattro pilastri principali**: **Collaborazione**, **Riproducibilità**, **Continuità** e **Monitoraggio**.

Ci concentreremo ora su ciascuno di essi, fornendo molteplici esempi pratici che mostrano come AWS, con molti dei suoi servizi, può essere uno strumento prezioso per sviluppare soluzioni che aderiscono alle best practice del paradigma.

Collaborazione

Un buon flusso di lavoro di Machine Learning dovrebbe essere **collaborativo** e la collaborazione si dovrebbe verificare su tutte le pipeline ML.

A partire dal Data Layer, abbiamo bisogno di **un'infrastruttura condivisa**, il che significa un **data lake distribuito**. AWS offre diverse soluzioni di storage per questo scopo, come **Amazon Redshift**, il più adatto per il Data Warehousing, o **Amazon FSx per Lustre**, perfetto come file system distribuito. Tuttavia, il servizio più comunemente utilizzato per la creazione di data lake è **Amazon S3**.

Per mantenere correttamente un data lake, dobbiamo effettuare regolarmente l'ingestion di dati da diverse fonti e gestire l'accesso condiviso tra i vari collaboratori, assicurandoci che i dati **siano sempre aggiornati**.

Sicuramente non un compito facile e per questo possiamo sfruttare **S3 LakeFormation**, un servizio gestito che aiuta a creare e mantenere un data lake, incapsulando **AWS Glue** e **Glue Studio**, in particolare semplificando il set-up dei Crawler di Glue e la loro manutenzione.

S3 LakeFormation può anche occuparsi dei dati e delle regole di autorizzazione dei collaboratori, gestendo utenti e ruoli nel **catalogo di AWS Glue**. Questa funzionalità è fondamentale, in quanto collaborazione significa anche mantenere la governance sul data lake, evitando manipolazioni involontarie dei dati, consentendo o negando l'accesso a risorse specifiche all'interno di un catalogo.

Per il livello del Modello, i Data Scientist hanno bisogno di uno **strumento per la progettazione collaborativa e la codifica dei modelli di Machine Learning**. Deve consentire a **più utenti di lavorare sullo stesso esperimento**, mostrare rapidamente i risultati di ciascun collaboratore, garantire la **programmazione in coppia in tempo reale** ed evitare il più possibile regressioni del codice e **conflitti di merge**.

SageMaker è il framework all-in-one opinionato per il Machine Learning su AWS e **Amazon SageMaker Studio** è un IDE specializzato sviluppato esplicitamente per lavorare con i notebook Jupyter pensando alla collaborazione.

SageMaker Studio permette di **condividere un'istanza EC2 dedicata** tra diversi utenti registrati, in cui è possibile salvare tutti gli esperimenti fatti durante lo sviluppo di un modello di Machine Learning. Questa istanza può ospitare direttamente i notebook Jupyter o ricevere risultati, allegati e grafica tramite API da altre istanze Notebook.

SageMaker Studio è inoltre direttamente integrato con **SageMaker Experiments** e **SageMaker Feature Store**.

Il primo è un set di API che consente ai Data Scientist di registrare e archiviare un esperimento sul modello, **dall'ottimizzazione** alla **convalida**, e riportare i risultati nella console dell'IDE. Il secondo è uno **store gestito da AWS, appositamente creato per la condivisione di parametri aggiornati su diverse prove dei modelli**.

SageMaker Feature Store rappresenta un notevole passo avanti nel mantenimento della governance sui parametri dei dati tra diversi team, principalmente perché evita un comportamento tipico, ma improprio, **di avere set di parametri diversi** per l'addestramento e l'inferenza. È anche una soluzione perfetta per garantire che ogni Data Scientist che lavora su un progetto abbia una **visibilità completa dell'etichettatura dei parametri**.

Riproducibilità

Per essere robusto, tollerante ai guasti e corretamente scalabile, proprio come le applicazioni software "tradizionali", un flusso di lavoro di Machine Learning deve essere **riproducibile**.

Un punto cruciale che dobbiamo affrontare con attenzione, come abbiamo detto prima, è il **Controllo di Versione**: dobbiamo garantire che codice, dati, metadati del modello e funzionalità siano adeguatamente versionati.

Per i notebook Jupyter, Git o **AWS CodeCommit** sono scelte naturali, ma la gestione delle informazioni di diversi esperimenti, in particolare dei metadati del modello, richiede alcune considerazioni particolari.

Possiamo utilizzare SageMaker Feature Store per metadati e funzionalità. Ci consente di archiviare i dati direttamente online in uno store gestito o di integrarci con **AWS Glue** (e S3 LakeFomation). Consente inoltre la crittografia dei dati utilizzando **AWS KMS** e può essere controllato tramite API o all'interno di SageMaker Studio.

Se si vuole che un flusso di lavoro sia riproducibile, si intende sperimentare **su larga scala**, anche in parallelo, in modo rapido, prevedibile e automatico.

SageMaker offre diversi modi per combinare e abbinare diversi algoritmi di Machine Learning e AWS permette tre possibili approcci per l'esecuzione di un modello.

Managed Algorithm: SageMaker offre fino a 13 algoritmi gestiti per scenari ML comuni e, per ognuno, una documentazione dettagliata descrive le specifiche software e hardware.

Bring your own algorithm: i Data Scientist possono introdurre rapidamente logica personalizzata sui notebook, a condizione che il modello rispetti i requisiti di **SageMaker fit()**.

Bring your own Container: modelli particolari come DBScan richiedono Kernel personalizzati per eseguire l'algoritmo, quindi SageMaker consente di registrare un container personalizzato con un Kernel speciale e il codice per l'esecuzione del modello.

I Data Scientist possono affrontare tutti questi approcci insieme.

SageMaker offre la possibilità di definire l'hardware su cui eseguire il training o la convalida del modello selezionando il **tipo di istanza** e la **dimensione di quest'ultima** nelle proprietà del modello, il che è estremamente importante in quanto algoritmi diversi richiedono macchine ottimizzate per CPU o GPU.

Per mettere a punto un modello, SageMaker può eseguire diverse strategie di **ottimizzazione degli iperparametri**: **Ricerca Casuale** e **Ricerca Bayesiana**. Queste due strategie sono completamente automatiche, garantendo un modo per testare un numero più significativo di combinazioni di prove in una frazione di tempo normalmente necessario.

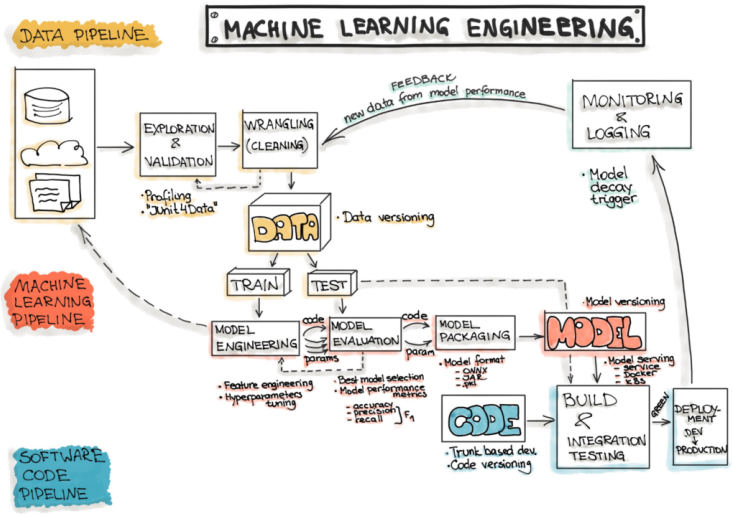
Per migliorare la ripetibilità degli esperimenti, dobbiamo anche gestire diversi modi di eseguire la preelaborazione dei dati (diversi set di dati applicati allo stesso modello). Per questo, abbiamo **AWS Data Wrangler**, che contiene oltre **300 trasformazioni di dati integrate** per normalizzare, trasformare e combinare rapidamente potenziali feature senza dover scrivere alcun codice.

AWS Data Wrangler può essere una buona scelta quando il problema di ML che si sta affrontando è in qualche modo standardizzato, ma nella maggior parte dei casi i set di dati sono estremamente diversi, il che significa affrontare i lavori di ETL personalmente.

Per i lavori ETL personalizzati, AWS Glue è ancora la scelta giusta, poiché consente anche di salvare i crawler di lavoro e i cataloghi di Glue (per la ripetibilità). Insieme ad AWS Glue e AWS Glue Studio, abbiamo anche provato **AWS Glue Elastic Views**, un nuovo servizio che aiuta a gestire diverse origini dati insieme.

Continuità

Per rendere **continuo** il nostro flusso di lavoro di Machine Learning, dobbiamo utilizzare il più possibile l'**automazione delle pipeline** per gestirne l'intero ciclo di vita.



Courtesy of https://ml-ops.org/content/three-levels-of-ml-software

Possiamo suddividere l'intero flusso di lavoro ML in tre pipeline significative, una per ogni livello di Machine Learning.

Pipeline di Data engineering

La pipeline dei dati è composta dalle fasi di **Acquisizione**, **Esplorazione**, **Convalida**, **Pulizia** e **Suddivisione**.

La fase di Acquisizione su AWS in genere significa portare i dati grezzi su S3, utilizzando qualsiasi strumento e tecnologia disponibile: accesso diretto all'API, crawler Lambda personalizzati, **S3 LakeFormation** o **Amazon Kinesis Firehose**.

Poi abbiamo una fase ETL di pre-elaborazione, che è **sempre richiesta**!

AWS Glue è il più versatile tra tutti gli strumenti disponibili per i processi di ETL, in quanto consente di leggere e aggregare informazioni da tutti i servizi precedenti utilizzando i **Glue Crawlers**. Queste routine possono eseguire il polling da divers datasource per ottenere nuovi dati.

Possiamo gestire le fasi di Esplorazione, Convalida e Pulizia creando script personalizzati in un linguaggio a scelta (ad es. Python) o utilizzando Jupyter Notebook, entrambi orchestrati tramite **AWS Step Functions**.

AWS Data Wrangler rappresenta un'altra soluzione praticabile, in quanto può occuparsi automaticamente di tutti i passaggi e connettersi direttamente a **Amazon SageMaker Pipelines**.

Pipeline di Modello

La pipeline del Modello è costituita da fasi di **Training**, **Valutazione**, **Test** e **Packaging**.

Queste fasi possono essere gestite direttamente dai file di Jupyter Notebook e integrate in una pipeline utilizzando **AWS StepFunctions SageMaker SDK**, che consente di chiamare le funzioni SageMaker all'interno di uno script StepFunction.

Questo exploit offre estrema flessibilità in quanto permette di:

1. **Avviare rapidamente i lavori di training di SageMaker** con tutti i parametri configurati.
2. **Valutare i modelli** utilizzando i **punteggi di valutazione precompilati** di SageMaker.
3. **Eseguire più test automatizzati** direttamente dal codice.
4. **Registrare** tutti i passaggi in Esperimenti di SageMaker.

Avere la logica di questa pipeline sui Notebook Jupyter ha l'ulteriore vantaggio di **avere tutto sotto controllo di versione** e **facilmente testabile**.

Il packaging può essere gestito tramite le **API di Elastic Container Registry**, direttamente da un Notebook Jupyter o da uno script esterno.

Pipeline di rilascio

La pipeline di distribuzione esegue la parte di **CI/CD** ed è responsabile della messa online dei modelli durante le fasi di **Training**, **Test** e **Produzione**. Un aspetto chiave durante questa pipeline è che la domanda di risorse computazionali è diversa per tutte e tre le fasi e cambia nel tempo.

Ad esempio, il training all'inizio richiederà più risorse rispetto ai test e alla produzione, ma in seguito, con l'aumentare della domanda di inferenze, i requisiti di produzione saranno più elevati (**Distribuzione Dinamica**).

Possiamo applicare strategie di deploy avanzate tipiche dello sviluppo software "tradizionale" per affrontare i flussi di lavoro ML, inclusi test A/B, implementazioni canary e implementazioni blue/green.

Ogni aspetto della distribuzione può trarre vantaggio dalle tecniche di Infrastructure as Code e da una combinazione di servizi AWS come AWS CodePipeline,

CloudFormation e AWS StepFunctions.

Monitoraggio

Infine, i flussi di lavoro di Machine Learning ben fatti devono essere **monitorabili** e il monitoraggio avviene in varie fasi.

Abbiamo il **monitoraggio delle prestazioni**, che permette di capire come si comporta un modello nel tempo. Avendo continuamente feedback basati su nuove inferenze, possiamo evitare l'invecchiamento del modello (**overfitting**) e il **concept drift**.

SageMaker Model Monitor fornisce un ottimo aiuto durante questa fase in quanto può eseguire il monitoraggio in tempo reale, rilevando bias e divergenze mediante tecniche di **Anomaly Detection** e inviando avvisi per applicare rimedi immediati.

Quando un modello inizia a performare al di sotto della soglia predefinita, la nostra pipeline inizierà un processo di riaddestramento con un set di dati aumentato, costituito da nuove informazioni provenienti da previsioni, diverse **combinazioni di iperparametri** o applicando il **re-labeling** sulle feature del set di dati.

SageMaker Clarify è un altro servizio che possiamo sfruttare nel processo di monitoraggio. Rileva potenziali bias durante la preparazione dei dati, l'addestramento del modello e la produzione per le feature più critiche selezionate nel set di dati.

Ad esempio, può verificare la presenza di divergenze legate all'età nel set di dati iniziale o in un modello addestrato e generare report dettagliati che quantificano diversi tipi di possibili bias. SageMaker Clarify include anche **grafici di importanza delle funzioni per spiegare le previsioni del modello**.

Il debug di un modello di Machine Learning, come possiamo vedere, è un processo lungo, complesso e costoso! C'è un altro utile servizio di AWS: **SageMaker Debugger**; esso **acquisisce le metriche di addestramento in tempo reale**, come la perdita di dati durante la regressione, e invia avvisi quando vengono rilevate anomalie.

SageMaker Debugger è ottimo per correggere immediatamente eventuali previsioni errate del modello.

Il logging su AWS può essere gestito sulla totalità della Pipeline utilizzando Amazon CloudWatch, disponibile con tutti i servizi presentati. Cloudwatch può essere

ulteriormente migliorato utilizzando **Kibana tramite ElasticSearch** per avere un modo semplice per esplorare i dati di log.

Possiamo anche utilizzare CloudWatch per **attivare procedure di rollback automatico** in caso di allarmi su alcune metriche chiave. Il rollback viene attivato anche da distribuzioni non riuscite.

Infine, la riproducibilità, la continuità e il monitoraggio di un carico di lavoro ML consentono il processo di messa a punto di costi/prestazioni, che avviene ciclicamente durante tutto il ciclo di vita del workflow.

Per riassumere

In questo articolo, abbiamo approfondito le caratteristiche del paradigma MLOps, mostrando come sono stati adottati concetti e pratiche della sua controparte DevOps per consentire al Machine Learning di adattarsi ai problemi del mondo reale e risolvere il cosiddetto deployment gap.

Abbiamo dimostrato che, mentre i carichi di lavoro del software tradizionale hanno cicli di vita più lineari, i problemi di Machine Learning si basano su tre macro-aree: Dati, Modello e Codice che sono profondamente interconnessi e forniscono un feedback continuo l'uno all'altro.

Abbiamo visto come affrontare questi particolari flussi di lavoro e come il paradigma MLOps permetta di gestire alcuni aspetti unici come le complessità nella gestione del codice del modello in Jupyter Notebook, esplorare i set di dati in modo efficiente con processi ETL corretti e fornire cicli di feedback rapidi e flessibili basati su metriche di produzione.

I modelli sono la seconda cosa più importante dopo i dati. Abbiamo appreso alcune strategie per evitare la deriva dei concetti e l'invecchiamento del modello nel tempo, come il training continuo, che richiede una soluzione di monitoraggio adeguata per fornire metriche di qualità sulle inferenze e una pipeline adeguata per invocare una nuova analisi del modello.

AWS fornisce alcuni servizi gestiti per aiutare con il training dei modelli e le pipeline in generale, come SageMaker AutoPilot e SageMaker Pipelines.

Abbiamo anche visto come AWS consenta diversi modi di creare e distribuire modelli per l'inferenza, come l'utilizzo di modelli precostruiti o l'utilizzo di un container con codice e algoritmi personalizzati. Tutte le immagini vengono salvate e recuperate da Elastic Container Registry.

Abbiamo parlato di come la collaborazione sia fondamentale a causa della natura sperimentale dei problemi di Machine Learning e di come AWS possa aiutare fornendo un IDE gestito all-in-one chiamato SageMaker Studio.

Abbiamo funzionalità come SageMaker Experiments per la gestione di più esperimenti, SageMaker Feature Store per raccogliere e trasformare in modo efficiente le etichette dei dati o SageMaker Model Monitoring e SageMaker Debugger per verificare la correttezza del modello e trovare eventuali bug.

Abbiamo anche discusso delle tecniche per rendere la nostra infrastruttura di Machine Learning solida, ripetibile e flessibile, facile da scalare su richieste, in base a requisiti che si evolvono nel tempo.

Tali metodi implicano l'utilizzo di template AWS Cloudformation per sfruttare l'Infrastructure as Code per la ripetibilità, AWS Step Functions per strutturare macchine a stati per gestire tutte le macro-aree e strumenti come AWS CodeBuild, CodeDeploy e CodePipeline per progettare adeguati flussi di CI/CD.

Ci auguriamo che ti sia piaciuto leggere questo articolo e che tu abbia imparato alcuni trucchi per gestire meglio i tuoi flussi di lavoro di Machine Learning.

Come già accennato, se il Machine Learning ti incuriosisce, ti invitiamo a dare un'occhiata ai nostri articoli con casi d'uso e analisi su ciò che AWS offre per affrontare i problemi di machine learning qui su Proud2beCloud!
Come sempre, sentiti libero di commentare nella sezione sottostante e contattaci per qualsiasi dubbio, domanda o idea!

Ci vediamo su **#Proud2beCloud** tra un paio di settimane per un'altra storia emozionante!

Alessandro Gaggia

Head of software development di beSharp, Full-Stack developer, mi occupo di garantire lo stato dell'arte di tutta la nostra codebase. Scrivo codice in quasi ogni linguaggio, ma prediligo Typescript. Respiro Informatica, Game design, Cinema, Fumetti e buona cucina. Disegno per passione!
