UNIVERSITÀ DEGLI STUDI DI PALERMO

Dipartimento di Ingegneria
Consiglio Interclasse del corso di studi
In Ingegneria dei Sistemi Ciber-Fisici per l'Industria

IT Modern Observability con l'implementazione di una rete neurale predittiva a supporto

TESI DI LAUREA DI

Federico Crucillà

Matricola: 0744768

RELATORE

Ilenia Tinnirello

ANNO ACCADEMICO 2023 - 2024

Indice:

Introduzione

Contesto e Motivazione

Nel contesto dell'Information Technology (IT), la gestione e il monitoraggio delle infrastrutture e delle applicazioni sono diventati sempre più complessi a causa dell'evoluzione delle architetture IT moderne. Con l'avvento dei microservizi, delle applicazioni cloud-native e delle infrastrutture distribuite, è emersa la necessità di adottare approcci più avanzati per comprendere il comportamento dei sistemi e garantire la loro affidabilità e performance. In questo scenario, l'osservabilità (observability) è diventata una disciplina chiave, fornendo strumenti e metodologie per ottenere una visione completa e dettagliata del funzionamento interno dei sistemi IT.

Definizione di Observability

L'osservabilità è un concetto derivato dalla teoria dei controlli, che si riferisce alla capacità di un sistema di rendere visibile il suo stato interno attraverso le sue uscite esterne. In altre parole, un sistema è considerato osservabile se, osservando le sue uscite (output), è possibile inferire accuratamente lo stato interno del sistema stesso. Questo concetto è stato inizialmente applicato ai sistemi di controllo ingegneristici, come quelli utilizzati nell'aeronautica e nell'automazione industriale, e successivamente adattato al contesto IT.

Importanza dell’Observability in IT

Nel contesto IT, l'osservabilità si riferisce alla capacità di comprendere il comportamento interno di applicazioni e infrastrutture attraverso la raccolta, l'analisi e la visualizzazione dei dati generati dai sistemi stessi. Questi dati possono includere log, metriche, tracce e altri tipi di telemetria. L'osservabilità è diventata particolarmente importante con l'avvento delle architetture moderne, che sono intrinsecamente più complesse e distribuite rispetto alle architetture monolitiche tradizionali.

Componenti Chiave dell’Observability

Le componenti chiave dell’osservabilità includono:

- Log: Registrazioni dettagliate degli eventi che si verificano all'interno di un sistema, fornendo informazioni cronologiche su ciò che è accaduto.

- Metriche: Misurazioni numeriche che rappresentano lo stato del sistema, come l'utilizzo della CPU, la memoria disponibile, il numero di richieste al secondo, ecc.

- Tracce: Forniscono una visione dettagliata del percorso di una richiesta attraverso vari componenti di un sistema distribuito, facilitando la diagnosi delle cause radice dei problemi.

Differenze tra Observability e Monitoring

Mentre il monitoraggio tradizionale si concentra sulla raccolta di dati predefiniti e sull'impostazione di avvisi statici, l'osservabilità adotta un approccio più dinamico e proattivo. L'osservabilità non si limita a monitorare metriche specifiche, ma cerca di fornire una visione completa del sistema, permettendo di rispondere a domande come: "Cosa è andato storto?", "Perché è successo?", "Qual è l'impatto sul sistema?" e "Come possiamo prevenire che accada di nuovo?".

Evoluzione dell’Observability

L'osservabilità ha subito una significativa evoluzione nel corso degli anni, passando da semplici pratiche di monitoraggio a un approccio più complesso e integrato che sfrutta tecnologie avanzate per fornire una visione completa del sistema. Questa evoluzione è stata guidata dalla crescente complessità delle architetture IT moderne, dall'aumento delle aspettative degli utenti e dalla necessità di migliorare l'affidabilità e le prestazioni dei sistemi.

Scopo dell’Observability

Lo scopo principale dell'osservabilità è migliorare la capacità di rilevare, diagnosticare e risolvere i problemi, ottimizzare le prestazioni e garantire la sicurezza e l'affidabilità del sistema. Un sistema ben osservabile consente ai team di sviluppo e operazioni di rilevare e risolvere i problemi più rapidamente, migliorare le prestazioni del sistema, aumentare l'affidabilità e la disponibilità e migliorare la sicurezza.

Obiettivi della Ricerca

Questa tesi si propone di esplorare e implementare un approccio di modern observability supportato da una rete neurale predittiva. Gli obiettivi specifici includono:

- Analizzare i fondamenti e i principi dell'observability in IT.
- Confrontare l'observability con il monitoraggio tradizionale e identificare i vantaggi e le limitazioni di entrambi.
- Implementare una rete neurale predittiva per supportare l'osservabilità, migliorando la capacità di rilevare e prevedere problemi.
- Valutare l'efficacia dell'approccio proposto attraverso studi di caso e analisi delle prestazioni.

Struttura della Tesi

La tesi è strutturata come segue:

- Capitolo 1: Fondamenti dell’Observability in IT
- Capitolo 2: Da Legacy Monitoring a Observability
- Capitolo 3: Implementazione e Problemi
- Capitolo 4: Risultati e Sviluppi Futuri

Capitolo 1: Fondamenti dell’Observability in IT

1.1 Definizione di Observability

L'osservabilità, o observability, è un concetto derivato dalla teoria dei controlli, che si riferisce alla capacità di un sistema di rendere visibile il suo stato interno attraverso le sue uscite esterne. In altre parole, un sistema è considerato osservabile se, osservando le sue uscite (output), è possibile inferire accuratamente lo stato interno del sistema stesso.

Origini del Concetto

Il termine "observability" ha le sue radici nella teoria dei sistemi dinamici, introdotta da Rudolf E. Kalman negli anni '60. In questo contesto, l'osservabilità è una proprietà che determina se lo stato interno di un sistema può essere ricostruito completamente dalle sue uscite esterne in un periodo di tempo finito. Questo concetto è stato inizialmente applicato ai sistemi di controllo ingegneristici, come quelli utilizzati nell'aeronautica e nell'automazione industriale.

Applicazione in IT

Nel contesto dell'Information Technology (IT), l'osservabilità si riferisce alla capacità di comprendere il comportamento interno di applicazioni e infrastrutture attraverso la raccolta, l'analisi e la visualizzazione dei dati generati dai sistemi stessi. Questi dati possono includere log, metriche, tracce e altri tipi di telemetria. L'osservabilità in IT è diventata particolarmente importante con l'avvento delle architetture moderne, come i microservizi e le applicazioni cloud-native, che sono intrinsecamente più complesse e distribuite rispetto alle architetture monolitiche tradizionali.

Componenti Chiave dell’Observability

Log: I log sono registrazioni dettagliate degli eventi che si verificano all'interno di un sistema. Essi forniscono informazioni cronologiche su ciò che è accaduto, rendendo possibile tracciare il comportamento del sistema e diagnosticare problemi. I log possono includere messaggi di errore,

avvisi, informazioni di debug e altre attività rilevanti. La raccolta e l'analisi dei log sono fondamentali per comprendere il contesto degli eventi e identificare le cause radice dei problemi.

Metriche: Le metriche sono misurazioni numeriche che rappresentano lo stato del sistema. Possono includere dati come l'utilizzo della CPU, la memoria disponibile, il numero di richieste al secondo, il tempo di risposta, ecc. Le metriche aiutano a monitorare le prestazioni del sistema in tempo reale e a identificare tendenze e anomalie. La raccolta delle metriche deve essere continua e a intervalli regolari per garantire che tutte le informazioni rilevanti siano disponibili per l'analisi.

Tracce: Le tracce (tracing) forniscono una visione dettagliata del percorso di una richiesta attraverso vari componenti di un sistema distribuito. Esse permettono di identificare colli di bottiglia e problemi di latenza, facilitando la diagnosi delle cause radice dei problemi. Il tracing è particolarmente utile nelle architetture a microservizi, dove una singola richiesta può attraversare molti servizi diversi. Le tracce consentono di vedere l'intero percorso della richiesta e di analizzare i tempi di risposta di ciascun componente.

Differenze tra Observability e Monitoring

Mentre il monitoraggio (monitoring) tradizionale si concentra sulla raccolta di dati predefiniti e sull'impostazione di avvisi statici, l'osservabilità adotta un approccio più dinamico e proattivo. Il monitoraggio tradizionale è spesso limitato a metriche di base e avvisi basati su soglie predefinite, il che può portare a falsi positivi o a mancati rilevamenti di problemi reali. L'osservabilità, d'altra parte, si basa su una raccolta dati più ampia e un'analisi più approfondita per identificare problemi nascosti.

L'osservabilità non si limita a monitorare metriche specifiche, ma cerca di fornire una visione completa del sistema, permettendo di rispondere a domande come:

- Cosa è andato storto?

- Perché è successo?

- Qual è l'impatto sul sistema?

- Come possiamo prevenire che accada di nuovo?

Figura 1.1: Confronto tra il funzionamento del monitoraggio e dell'osservabilità.

Importanza dell'Observability

L'osservabilità è diventata cruciale con l'evoluzione delle architetture IT moderne, come i microservizi e le applicazioni cloud-native. Queste architetture sono spesso complesse e distribuite, rendendo difficile ottenere una visione chiara del loro stato senza strumenti avanzati di osservabilità. Un sistema ben osservabile consente ai team di sviluppo e operazioni di:

- Rilevare e risolvere i problemi più rapidamente: Con una visione chiara del sistema, è possibile identificare e diagnosticare i problemi in modo più efficiente. Questo riduce i tempi di inattività e migliora la disponibilità del sistema.

- Migliorare le prestazioni del sistema: Monitorando continuamente le metriche chiave, è possibile ottimizzare le prestazioni del sistema. L'osservabilità consente di identificare i colli di bottiglia e le aree di inefficienza, permettendo di ottimizzare l'allocazione delle risorse.

- Aumentare l'affidabilità e la disponibilità: Prevedendo e prevenendo i problemi prima che si verifichino, si può migliorare l'affidabilità del sistema. L'osservabilità consente di rilevare anomalie e problemi potenziali prima che diventino critici, permettendo di intervenire proattivamente.

- Migliorare la sicurezza: L'osservabilità può anche contribuire a migliorare la sicurezza del sistema, fornendo una visione dettagliata delle attività e degli eventi. Monitorando continuamente il sistema, è possibile rilevare attività sospette e potenziali minacce di sicurezza. Con una visione chiara degli eventi, è possibile rispondere rapidamente agli incidenti di sicurezza e mitigare i danni.

- Facilitare la collaborazione: L'osservabilità facilita la collaborazione tra i team di sviluppo, operazioni e sicurezza, fornendo una visione condivisa del sistema. Con dashboard e strumenti di visualizzazione condivisi, i team possono collaborare più efficacemente e risolvere i problemi più rapidamente. L'osservabilità aiuta a garantire che tutti i team siano allineati sugli obiettivi di prestazioni, affidabilità e sicurezza del sistema.

L'osservabilità rappresenta un'evoluzione significativa nel modo in cui gestiamo e monitoriamo i sistemi IT, fornendo strumenti e metodologie per ottenere una comprensione profonda e dettagliata del comportamento del sistema. Questa comprensione è essenziale per migliorare la capacità di rilevare, diagnosticare e risolvere i problemi, ottimizzare le prestazioni e garantire la sicurezza e l'affidabilità del sistema.

1.2 Principi di Observability

L'osservabilità si basa su alcuni principi fondamentali che guidano la raccolta, l'analisi e l'interpretazione dei dati di sistema. Questi principi sono essenziali per costruire un sistema IT osservabile ed efficiente.

Raccolta Omnicomprensiva dei Dati

Un sistema osservabile deve raccogliere dati da tutte le parti del sistema, compresi i log, le metriche e le tracce. Questa raccolta deve essere continua e in tempo reale per garantire che tutte le informazioni rilevanti siano disponibili per l'analisi.

- Log: Devono essere dettagliati e strutturati in modo tale da poter essere facilmente analizzati. I log devono includere informazioni su eventi, errori, avvisi e altre attività rilevanti.

- Metriche: Devono essere raccolte a intervalli regolari e devono coprire tutte le risorse critiche del sistema, come CPU, memoria, rete e utilizzo del disco.

- Tracce: Devono fornire una visione end-to-end delle richieste, mostrando come si propagano attraverso i vari componenti del sistema.

Correlazione dei Dati

I dati raccolti devono essere correlati per fornire una visione completa del sistema. Questo significa che i log, le metriche e le tracce devono essere integrati e analizzati insieme per identificare le relazioni tra eventi e comportamenti del sistema.

- Correlazione Temporale: Assicurarsi che tutti i dati siano sincronizzati nel tempo per poter correlare eventi che accadono simultaneamente in diverse parti del sistema.

- Correlazione Contestuale: Collegare i dati di diverse fonti per ottenere una visione completa di un evento o di un problema specifico.

Analisi e Visualizzazione

L'analisi dei dati è fondamentale per trasformare i dati grezzi in informazioni utili. Gli strumenti di osservabilità devono fornire funzionalità avanzate di analisi e visualizzazione per aiutare gli operatori a comprendere rapidamente lo stato del sistema.

- Dashboard Interattive: Fornire visualizzazioni in tempo reale delle metriche chiave e degli eventi del sistema.

- Alerting Intelligente: Configurare avvisi basati su soglie dinamiche e modelli di comportamento per rilevare anomalie e problemi potenziali.

- Analisi Forense: Consentire l'analisi retrospettiva degli eventi per identificare le cause radice dei problemi.

Automazione e Intelligenza Artificiale

L'uso di tecniche di automazione e intelligenza artificiale (AI) può migliorare significativamente l'osservabilità. Questi strumenti possono aiutare a rilevare modelli e anomalie che potrebbero non essere evidenti attraverso l'analisi manuale.

- Machine Learning: Utilizzare algoritmi di machine learning per prevedere problemi futuri basati su dati storici e attuali.

- Automazione dei Processi: Automatizzare la raccolta e l'analisi dei dati per ridurre il carico di lavoro manuale e migliorare l'efficienza.

Feedback e Miglioramento Continuo

L'osservabilità non è un processo statico. Deve essere continuamente migliorata e adattata alle esigenze del sistema e dell'organizzazione.

- Feedback Loop: Implementare un ciclo di feedback continuo per raccogliere input dagli utenti e migliorare gli strumenti e i processi di osservabilità.

- Aggiornamenti Regolari: Aggiornare regolarmente gli strumenti e le tecniche di osservabilità per adattarsi alle nuove tecnologie e alle nuove sfide.

1.3 Evoluzione dell'Observability

L'osservabilità ha subito una significativa evoluzione nel corso degli anni, passando da semplici pratiche di monitoraggio a un approccio più complesso e integrato che sfrutta tecnologie avanzate per fornire una visione completa del sistema. Questa evoluzione è stata guidata dalla crescente complessità delle architetture IT moderne, dall'aumento delle aspettative degli utenti e dalla necessità di migliorare l'affidabilità e le prestazioni dei sistemi.

Dalla Supervisione Tradizionale all'Observability

La supervisione tradizionale, o monitoraggio, si concentrava principalmente sulla raccolta di metriche statiche e sull'impostazione di avvisi basati su soglie predefinite. Questo approccio era sufficiente per le architetture monolitiche, dove i componenti del sistema erano strettamente integrati e relativamente semplici da monitorare. Tuttavia, con l'avvento delle architetture distribuite e dei microservizi, il monitoraggio tradizionale ha mostrato i suoi limiti.

In passato, gli strumenti di monitoraggio come Nagios, Zabbix e Cacti erano ampiamente utilizzati per raccogliere metriche di base come l'utilizzo della CPU, la memoria, lo spazio su disco e il traffico di rete. Gli avvisi erano configurati per essere attivati quando queste metriche superavano soglie predefinite. Tuttavia, questo approccio presentava diverse limitazioni. Le soglie statiche potevano portare a falsi positivi o a mancati rilevamenti di problemi reali, e la mancanza di correlazione tra i dati rendeva difficile identificare le cause radice dei problemi.

Transizione verso l'Observability

Con l'aumento della complessità dei sistemi IT, è emersa la necessità di un approccio più avanzato e completo. L'osservabilità è nata come risposta a queste esigenze, fornendo una visione più dettagliata e integrata del sistema. L'osservabilità si basa sulla raccolta di tre tipi principali di dati di telemetria: log, metriche e tracce.

I log sono registrazioni dettagliate degli eventi che si verificano all'interno del sistema. Essi forniscono informazioni cronologiche su ciò che è accaduto, rendendo possibile tracciare il comportamento del sistema e diagnosticare problemi. Le metriche sono misurazioni numeriche che rappresentano lo stato del sistema. Possono includere dati come l'utilizzo della CPU, la memoria disponibile, il numero di richieste al secondo, ecc. Le metriche aiutano a monitorare le prestazioni del sistema in tempo reale. Le tracce forniscono una visione dettagliata del percorso di una richiesta attraverso vari componenti di un sistema distribuito. Esse permettono di identificare colli di bottiglia e problemi di latenza, facilitando la diagnosi delle cause radice dei problemi.

L'osservabilità moderna utilizza strumenti avanzati per raccogliere, analizzare e visualizzare questi dati, fornendo una visione completa del sistema. Gli strumenti di osservabilità come Prometheus, Grafana, Jaeger e Elastic Stack (ELK) offrono funzionalità avanzate di raccolta, analisi e visualizzazione dei dati. Prometheus è uno strumento di monitoraggio e allerta open-source progettato per raccogliere e analizzare metriche in tempo reale. Grafana è una piattaforma di analisi open-source che consente di creare dashboard interattive per visualizzare dati da diverse fonti. Jaeger è uno strumento di tracciamento distribuito open-source utilizzato per monitorare e

risolvere problemi in architetture microservizi. Elastic Stack è una suite di strumenti composta da Elasticsearch, Logstash e Kibana, utilizzata per raccogliere, analizzare e visualizzare log e altri dati di telemetria.

Strumenti Moderni di Observability

Gli strumenti moderni di osservabilità offrono funzionalità avanzate per raccogliere, analizzare e visualizzare dati da diverse fonti. Alcuni degli strumenti più popolari includono:

- Prometheus: Uno strumento di monitoraggio e allerta open-source progettato per raccogliere e analizzare metriche in tempo reale. Prometheus utilizza un modello di dati basato su una serie temporale e offre un potente linguaggio di query (PromQL) per analizzare i dati raccolti.

- Grafana: Una piattaforma di analisi open-source che consente di creare dashboard interattive per visualizzare dati da diverse fonti. Grafana si integra facilmente con Prometheus e altri strumenti di osservabilità, offrendo una visualizzazione chiara e personalizzabile delle metriche.

- Elastic Stack (ELK): Una suite di strumenti composta da Elasticsearch, Logstash e Kibana, utilizzata per raccogliere, analizzare e visualizzare log e altri dati di telemetria. Elasticsearch è un motore di ricerca e analisi distribuito, Logstash è uno strumento di elaborazione dei dati e Kibana è una piattaforma di visualizzazione.

Benefici dell’Observability

L'osservabilità offre numerosi benefici rispetto al monitoraggio tradizionale. Fornisce una visione completa del sistema, raccogliendo dati da tutte le parti del sistema e correlando questi dati per ottenere una visione integrata. La correlazione dei dati e l'analisi avanzata consentono di diagnosticare rapidamente i problemi e di identificare le cause radice. Gli strumenti di osservabilità sono progettati per scalare in ambienti complessi e distribuiti, come quelli basati su microservizi e cloud. L'osservabilità consente di rilevare problemi potenziali prima che diventino critici, permettendo di intervenire proattivamente e riducendo i tempi di inattività.

L'osservabilità moderna è essenziale per gestire le complessità delle architetture IT moderne, come i microservizi e le applicazioni cloud-native. Queste architetture sono spesso complesse e distribuite, rendendo difficile ottenere una visione chiara del loro stato senza strumenti avanzati di osservabilità. Un sistema ben osservabile consente ai team di sviluppo e operazioni di rilevare e risolvere i problemi più rapidamente, migliorare le prestazioni del sistema, aumentare l'affidabilità e la disponibilità e migliorare la sicurezza.

L'osservabilità rappresenta un'evoluzione significativa nel modo in cui gestiamo e monitoriamo i sistemi IT, fornendo strumenti e metodologie per ottenere una comprensione profonda e dettagliata del comportamento del sistema. Questa evoluzione è stata guidata dalla crescente complessità delle architetture IT moderne, dall'aumento delle aspettative degli utenti e dalla necessità di migliorare l'affidabilità e le prestazioni dei sistemi.

1.4 Scopo dell'Observability

L'osservabilità è una disciplina chiave nell'ingegneria dei sistemi IT moderni, progettata per fornire una visione chiara e completa del funzionamento interno di un sistema attraverso l'analisi delle sue uscite esterne. Lo scopo dell'osservabilità è migliorare la capacità di rilevare, diagnosticare e risolvere i problemi, ottimizzare le prestazioni e garantire la sicurezza e l'affidabilità del sistema.

Uno degli obiettivi principali dell'osservabilità è migliorare la capacità di rilevare e risolvere i problemi in modo rapido ed efficiente. Questo obiettivo è raggiunto attraverso la raccolta e l'analisi di dati dettagliati che forniscono una visione completa del sistema. L'osservabilità consente di diagnosticare rapidamente i problemi identificando le cause radice. La correlazione dei dati provenienti da diverse fonti, come log, metriche e tracce, permette di individuare rapidamente l'origine dei problemi. Con una visione chiara e completa del sistema, è possibile intervenire tempestivamente per risolvere i problemi, riducendo i tempi di inattività e migliorando la disponibilità del sistema. Inoltre, l'osservabilità consente di rilevare anomalie e problemi potenziali prima che diventino critici, permettendo di intervenire proattivamente e prevenire interruzioni del servizio.

L'osservabilità fornisce anche gli strumenti necessari per monitorare e ottimizzare le prestazioni del sistema, garantendo che le risorse siano utilizzate in modo efficiente e che il sistema funzioni al massimo delle sue capacità. Monitorando continuamente le metriche chiave, l'osservabilità aiuta a identificare i colli di bottiglia e le aree di inefficienza. Questo consente di ottimizzare le prestazioni del sistema e migliorare l'esperienza dell'utente. Con una visione chiara delle prestazioni del sistema, è possibile scalare le risorse in modo efficace per soddisfare la domanda. L'osservabilità consente di monitorare l'utilizzo delle risorse e di adattare dinamicamente la capacità del sistema. Inoltre, l'osservabilità consente di ottimizzare l'allocazione delle risorse, garantendo che vengano utilizzate in modo efficiente e riducendo i costi operativi.

Un altro obiettivo fondamentale dell'osservabilità è migliorare l'affidabilità del sistema, consentendo di prevedere e prevenire i problemi prima che si verifichino. Utilizzando modelli predittivi basati su dati storici e attuali, l'osservabilità consente di identificare problemi potenziali e di intervenire prima che diventino critici. La capacità di rilevare e risolvere rapidamente i problemi riduce i tempi di inattività e migliora la disponibilità del sistema. Inoltre, l'osservabilità aiuta a identificare e mitigare i rischi, migliorando la resilienza del sistema e garantendo che possa continuare a funzionare anche in presenza di guasti.

L'osservabilità può anche contribuire a migliorare la sicurezza del sistema, fornendo una visione dettagliata delle attività e degli eventi che si verificano all'interno del sistema. Monitorando continuamente il sistema, l'osservabilità consente di rilevare attività sospette e potenziali minacce di sicurezza. Questo include il rilevamento di attacchi DDoS, tentativi di intrusione e altre attività malevole. Con una visione chiara degli eventi, è possibile rispondere rapidamente agli incidenti di sicurezza e mitigare i danni. L'osservabilità consente di tracciare e analizzare gli incidenti, facilitando la risposta e la risoluzione. Inoltre, l'osservabilità fornisce una traccia dettagliata delle attività del sistema, facilitando la conformità alle normative e agli standard di sicurezza. Questo include la registrazione e l'analisi dei log di sicurezza per scopi di audit.

L'osservabilità facilita anche la collaborazione tra i team di sviluppo, operazioni e sicurezza, fornendo una visione condivisa del sistema. Con dashboard e strumenti di visualizzazione condivisi, i team possono collaborare più efficacemente e risolvere i problemi più rapidamente. L'osservabilità consente di centralizzare le informazioni e di renderle accessibili a tutti i membri del team. Inoltre, l'osservabilità aiuta a garantire che tutti i team siano allineati sugli obiettivi di prestazioni, affidabilità e sicurezza del sistema. Questo facilita la collaborazione e migliora l'efficienza operativa. L'osservabilità fornisce una base comune di dati e informazioni, migliorando la comunicazione tra i team e facilitando la risoluzione dei problemi.

L'osservabilità fornisce dati e informazioni che supportano il processo decisionale, consentendo di prendere decisioni informate e basate sui dati. L'osservabilità consente di raccogliere e analizzare dati dettagliati sul funzionamento del sistema, fornendo informazioni preziose per il processo decisionale. Utilizzando i dati raccolti, è possibile ottimizzare le strategie di gestione del sistema, migliorando le prestazioni e l'efficienza operativa. Inoltre, l'osservabilità consente di prevedere le tendenze future e di pianificare in modo proattivo, garantendo che il sistema sia preparato per affrontare le sfide future.

L'osservabilità è essenziale per migliorare la capacità di rilevare, diagnosticare e risolvere i problemi, ottimizzare le prestazioni, garantire la sicurezza e l'affidabilità del sistema, facilitare la collaborazione tra i team e supportare il processo decisionale. Questi obiettivi sono raggiunti attraverso la raccolta e l'analisi di dati dettagliati che forniscono una visione completa e integrata del sistema, consentendo di intervenire proattivamente e migliorare l'efficienza operativa.

Capitolo 2: Da Legacy Monitoring a Observability

Questo capitolo esamina la transizione dal monitoraggio tradizionale (legacy monitoring) all'observability. Vengono analizzate le differenze tra i due approcci, con un focus sulle limitazioni del monitoraggio tradizionale e i vantaggi dell'observability moderna. Si discutono strumenti e tecnologie avanzate come Prometheus, Grafana e l'ELK Stack, che offrono funzionalità di raccolta, analisi e visualizzazione dei dati, migliorando la capacità di rilevare e risolvere i problemi in modo proattivo.

2.1 Differenza tra Legacy Monitoring e Observability

Il monitoraggio dei sistemi IT è un aspetto cruciale per garantire la disponibilità, le prestazioni e la sicurezza delle applicazioni e delle infrastrutture. Con l'evoluzione delle architetture IT, si è passati dal legacy monitoring all'observability. Questa sezione esplora in dettaglio le differenze tra questi due approcci, evidenziando i vantaggi dell'observability.

Legacy Monitoring

Definizione e Caratteristiche

Il legacy monitoring si riferisce ai metodi tradizionali di monitoraggio dei sistemi IT, che sono stati utilizzati per decenni. Questo approccio è caratterizzato da:

- Metriche Predefinite: Il monitoraggio tradizionale si basa su metriche predefinite, come l'utilizzo della CPU, la memoria, lo spazio su disco e la latenza delle reti. Queste metriche sono configurate manualmente e monitorate per rilevare anomalie.

- Allarmi Basati su Soglie: Gli allarmi vengono generati quando una metrica supera una soglia predefinita. Questo approccio può portare a falsi positivi o falsi negativi, poiché le soglie potrebbero non essere adeguate per tutte le situazioni.

- Reattività: Il legacy monitoring è spesso reattivo, intervenendo solo quando si verifica un problema noto. Questo può portare a tempi di inattività prolungati e a una risoluzione dei problemi meno efficiente.

- Visibilità Limitata: Il monitoraggio tradizionale fornisce una visibilità limitata sui sistemi complessi, rendendo difficile identificare le cause profonde dei problemi.

- Strumenti Utilizzati: Gli strumenti di legacy monitoring includono soluzioni come Zabbix, che sono efficaci per il monitoraggio di infrastrutture, ma possono risultare insufficienti per ambienti complessi e dinamici.

Limiti del Legacy Monitoring

- Flessibilità Limitata: Gli strumenti di legacy monitoring spesso mancano della flessibilità necessaria per adattarsi a nuove tecnologie e architetture.

- Scalabilità: Le soluzioni tradizionali possono avere difficoltà a scalare efficacemente in ambienti cloud-native e distribuiti.

- Integrazione Dati: La capacità di integrare dati da diverse fonti è spesso limitata, rendendo difficile ottenere una visione unificata del sistema.

Observability

Definizione e Caratteristiche

L'observability è un approccio più moderno e avanzato che mira a fornire una visione completa e dettagliata dei sistemi IT. Le caratteristiche principali dell'observability sono:

- Raccolta di Dati Omnicomprensiva: L'observability raccoglie e analizza una vasta gamma di dati, inclusi log, metriche e tracce. Questo permette di ottenere una visione completa del comportamento del sistema.

- Analisi in Tempo Reale: I dati vengono analizzati in tempo reale, permettendo di identificare e risolvere i problemi più rapidamente. Questo approccio proattivo riduce i tempi di inattività e migliora l'affidabilità del sistema.

- Comprensione dei Sistemi Complessi: L'observability è particolarmente utile per i sistemi distribuiti e complessi, come i microservizi. Permette di tracciare le dipendenze tra i vari componenti e di identificare i colli di bottiglia.

- Root Cause Analysis Avanzata: Utilizzando tecniche avanzate come il machine learning, l'observability può identificare le cause profonde dei problemi, anche quando non sono immediatamente evidenti.

- Flessibilità e Adattabilità: L'observability è altamente flessibile e può adattarsi a diversi ambienti e scenari. Questo la rende ideale per le moderne architetture cloud-native e per i sistemi in continua evoluzione.

- Strumenti Utilizzati: Gli strumenti di observability includono soluzioni come Prometheus, Grafana e l'ELK Stack (Elasticsearch, Logstash, Kibana), che offrono funzionalità avanzate di raccolta, analisi e visualizzazione dei dati.

Vantaggi dell'Observability

1. Proattività: L'observability permette di identificare e risolvere i problemi prima che abbiano un impatto significativo sul sistema. Questo approccio proattivo migliora l'affidabilità e la disponibilità del sistema.

2. Risposta Rapida: Grazie all'analisi in tempo reale, l'observability permette di rispondere rapidamente ai problemi, riducendo i tempi di inattività e migliorando l'esperienza degli utenti.

3. Migliore Comprensione: L'observability fornisce una visione completa e dettagliata del sistema, permettendo di comprendere meglio il comportamento del sistema e di identificare le cause profonde dei problemi.

4. Adattabilità: L'observability è altamente flessibile e può adattarsi a diversi ambienti e scenari, rendendola ideale per le moderne architetture cloud-native e per i sistemi in continua evoluzione.

5. Supporto al DevOps: L'observability è un elemento chiave delle pratiche DevOps, permettendo una migliore collaborazione tra i team di sviluppo e operazioni e migliorando l'efficienza complessiva del ciclo di vita del software.

Legacy Monitoring vs Observability

- Legacy Monitoring: Un sistema di monitoraggio tradizionale potrebbe generare un allarme quando l'utilizzo della CPU di un server supera il 90% per più di 5 minuti. Tuttavia, questo approccio non fornisce informazioni sul motivo per cui l'utilizzo della CPU è elevato, né su quali applicazioni o processi specifici stanno causando il problema. Di conseguenza, gli amministratori potrebbero dover eseguire un'analisi manuale per identificare la causa del problema, il che può richiedere tempo e risorse.

- Observability: Un sistema di observability, invece, raccoglierebbe log dettagliati, metriche e tracce per analizzare l'intero contesto. Potrebbe identificare che l'aumento dell'utilizzo della CPU è dovuto a una specifica richiesta di un microservizio, che a sua volta è stata causata da un aumento del traffico in un'altra parte del sistema. Questo livello di dettaglio permette di risolvere il problema in modo più efficace e di prevenire futuri incidenti simili.

Tabella Comparativa

Caratteristiche	*Legacy Monitoring*	*Observability*
Metriche	Predefinite	Omnicomprensive
Allarmi	Basati su soglie	Basati su analisi in tempo reale
Reattività	Reattivo	Proattivo
Visibilità	Limitata	Completa
Strumenti	Zabbix	Prometeus, Grafana, ELK Stack
Root cause analysis	Limitata	Avanzata
Flessibilità	Limitata	Elevata
Supporto DevOps	Limitato	Completo2

La differenza tra legacy monitoring e observability risiede principalmente nell'approccio e negli obiettivi. Mentre il legacy monitoring si concentra sul monitoraggio di metriche predefinite e sull'identificazione di problemi noti, l'observability adotta un approccio proattivo e olistico, raccogliendo e analizzando una vasta gamma di dati per comprendere il comportamento dei sistemi complessi. Questo permette di identificare e risolvere i problemi in modo più efficace, migliorando l'affidabilità e la disponibilità del sistema.

Tecniche Avanzate di Observability

- Machine Learning: Utilizzo di algoritmi di machine learning per migliorare l'analisi dei dati e la root cause analysis.

- Automazione: Implementazione di automazioni per la raccolta e l'analisi dei dati, riducendo il carico di lavoro manuale.

Riepilogo delle differenze tra monitoraggio e osservabilità

Caratteristica	*Monitoraggio*	*Osservabilità*
In cosa consiste?	Misurazione e creazione di report su metriche specifiche all'interno di un sistema per garantirne l'integrità	Raccolta di metriche, eventi, log e tracce per consentire un'indagine approfondita sui problemi di salute nei sistemi distribuiti con architetture di microservizi
Focus principale	Raccolta di dati per identificare effetti anomali del sistema	Esame sulla causa principale di effetti anomali del sistema

Sistemi coinvolti	In genere, sistemi autonomi	In genere, sistemi multipli, e disparati
Tracciabilità	Limitato agli edge del sistema	Disponibile dove i segnali vengono emessi su architetture di sistema disparate
Esiti degli errori di sistema	Il quando e il cosa	Il come e il perché

2.2 Modern Observability

L'observability moderna rappresenta un'evoluzione significativa rispetto ai tradizionali metodi di monitoraggio, offrendo una visione più completa e dettagliata dei sistemi IT. Questo approccio si basa su tre pilastri fondamentali: logs, metriche e tracce. Questi elementi forniscono una visione completa del sistema e permettono di risolvere i problemi in modo più efficace.

Logs

I log sono registrazioni dettagliate degli eventi che si verificano all'interno del sistema. Forniscono un contesto ricco e dettagliato che è essenziale per il debugging e l'analisi postmortem. I log possono includere informazioni su errori, avvisi, eventi di sistema e messaggi di debug. Quando un'applicazione genera un errore, il log può fornire informazioni dettagliate sul contesto in cui si è verificato l'errore, inclusi i parametri di input, l'output atteso e l'output effettivo. Questo livello di dettaglio è cruciale per identificare e risolvere i problemi in modo efficace.

I log possono essere categorizzati in diversi tipi, tra cui:

Log di applicazione: Informazioni generate dalle applicazioni in esecuzione, come errori, avvisi e messaggi di debug.

Log di sistema: Informazioni generate dal sistema operativo, come eventi di avvio e arresto, errori hardware e cambiamenti di configurazione.

La raccolta e l'analisi dei log sono fondamentali per identificare e risolvere i problemi. I log possono essere raccolti utilizzando strumenti come Logstash, Fluentd e altri agenti di raccolta log, che inviano i dati a un sistema centrale per l'analisi e la visualizzazione.

Metriche

Le metriche sono misurazioni quantitative che forniscono una visione d'insieme delle prestazioni del sistema. Possono includere dati come l'utilizzo della CPU, la memoria disponibile, il numero di richieste al secondo, la latenza delle richieste e il throughput. Le metriche sono utili per monitorare lo stato di salute del sistema e per identificare trend e pattern nel tempo. Un aumento graduale dell'utilizzo della CPU potrebbe indicare un problema di performance che deve essere risolto prima che abbia un impatto significativo sull'utente finale. Le metriche possono essere raccolte e visualizzate in tempo reale, permettendo agli amministratori di monitorare costantemente lo stato del sistema e di intervenire rapidamente in caso di problemi.

Le metriche possono essere suddivise in diverse categorie, tra cui:

Metriche di sistema: Misurazioni delle risorse del sistema, come CPU, memoria, disco e rete.

Metriche di applicazione: Misurazioni delle prestazioni delle applicazioni, come il numero di richieste al secondo, la latenza delle richieste e il throughput.

La raccolta e l'analisi delle metriche sono fondamentali per monitorare lo stato di salute del sistema e per identificare trend e pattern nel tempo. Le metriche possono essere raccolte utilizzando strumenti come Prometheus e Grafana.

Tracce

Le tracce forniscono una visione dettagliata del percorso delle richieste attraverso il sistema. Aiutano a identificare i colli di bottiglia e a comprendere le dipendenze tra i diversi componenti del sistema. Le tracce sono particolarmente utili nei sistemi distribuiti e nei microservizi, dove una singola richiesta può attraversare molti servizi diversi prima di completarsi. Una traccia può mostrare che una richiesta HTTP passa attraverso un servizio di autenticazione, un servizio di autorizzazione, un servizio di gestione degli ordini e un servizio di pagamento. Se la richiesta impiega troppo tempo a completarsi, la traccia può aiutare a identificare quale servizio sta causando il ritardo. Le tracce possono anche fornire informazioni dettagliate sui tempi di risposta di ciascun servizio, permettendo agli amministratori di ottimizzare le prestazioni del sistema.

Architettura di Observability

L'architettura di un sistema di observability moderna è progettata per raccogliere, analizzare e visualizzare dati da diverse fonti in modo efficiente e scalabile. Un'architettura tipica può includere i seguenti componenti:

1. Raccolta Dati: I dati vengono raccolti da diverse fonti, inclusi log, metriche e tracce. Questo può essere fatto utilizzando agenti di raccolta dati installati sui server, o tramite API che inviano dati a un sistema centrale di raccolta.

2. Ingestione Dati: I dati raccolti vengono inviati a un sistema di ingestione dati, che li elabora e li archivia in un formato strutturato. Questo può includere la normalizzazione dei dati, la rimozione dei duplicati e l'arricchimento dei dati con informazioni aggiuntive.

3. Archiviazione Dati: I dati elaborati vengono archiviati in un sistema di archiviazione dati, che può essere un database relazionale, un database NoSQL, o un sistema di archiviazione distribuito. L'archiviazione dei dati deve essere scalabile e resiliente, in modo da poter gestire grandi volumi di dati e garantire l'accesso continuo ai dati.

4. Analisi Dati: I dati archiviati vengono analizzati utilizzando strumenti di analisi dati, che possono includere algoritmi di machine learning, analisi statistica e visualizzazione dati. L'analisi dei dati permette di identificare pattern, trend e anomalie nei dati, e di generare allarmi e notifiche in caso di problemi.

5. Visualizzazione Dati: I dati analizzati vengono visualizzati utilizzando strumenti di visualizzazione dati, come dashboard e report. La visualizzazione dei dati permette agli

amministratori di monitorare lo stato del sistema in tempo reale, e di ottenere una visione d'insieme delle prestazioni del sistema.

Strumenti di Observability

Esistono numerosi strumenti e tecnologie che supportano l'observability. Alcuni dei più popolari includono:

Prometheus: Un sistema di monitoraggio e allarme opensource progettato per raccogliere e analizzare metriche. Prometheus utilizza un modello di dati basato su serie temporali, che permette di raccogliere e analizzare dati in tempo reale. Prometheus include anche un potente linguaggio di query, PromQL, che permette di eseguire query complesse sui dati raccolti.

Grafana: Una piattaforma di analisi e visualizzazione opensource che si integra con Prometheus e altri sistemi di monitoraggio. Grafana permette di creare dashboard personalizzate per visualizzare i dati raccolti, e include una vasta gamma di plugin per integrare dati da diverse fonti.

Elasticsearch, Logstash, Kibana (ELK Stack): Una suite di strumenti per la raccolta, l'analisi e la visualizzazione dei log. Elasticsearch è un motore di ricerca e analisi distribuito, Logstash è uno strumento di raccolta e trasformazione dei dati, e Kibana è uno strumento di visualizzazione dati. Insieme, questi strumenti formano una potente piattaforma di observability.

Vantaggi dell'Observability Moderna

L'observability moderna offre numerosi vantaggi rispetto ai tradizionali metodi di monitoraggio. Permette di ottenere una visione completa e dettagliata del sistema, raccogliendo e analizzando dati da diverse fonti. Questo permette di identificare e risolvere i problemi in modo più efficace, riducendo i tempi di inattività e migliorando l'affidabilità del sistema. L'observability moderna è particolarmente utile per i sistemi distribuiti e complessi, come i microservizi, dove una singola richiesta può attraversare molti servizi diversi prima di completarsi. Permette di tracciare le dipendenze tra i vari componenti del sistema, e di identificare i colli di bottiglia. Utilizzando tecniche avanzate come il machine learning, l'observability moderna può identificare le cause profonde dei problemi, anche quando non sono immediatamente evidenti. L'observability moderna è altamente flessibile e può adattarsi a diversi ambienti e scenari, rendendola ideale per le moderne architetture cloudnative e per i sistemi in continua evoluzione.

L'observability moderna supporta anche le pratiche DevOps, permettendo una migliore collaborazione tra i team di sviluppo e operazioni e migliorando l'efficienza complessiva del ciclo di vita del software. Un sistema di observability può fornire dati dettagliati sulle prestazioni delle applicazioni in produzione, permettendo ai team di sviluppo di identificare e risolvere i problemi più rapidamente.

L'observability moderna rappresenta un'evoluzione significativa rispetto ai tradizionali metodi di monitoraggio, offrendo una visione più completa e dettagliata dei sistemi IT. Questo permette di identificare e risolvere i problemi in modo più efficace, migliorando l'affidabilità e la disponibilità del sistema. L'observability moderna è particolarmente utile per i sistemi distribuiti e complessi, come i microservizi, dove una singola richiesta può attraversare molti servizi diversi prima di completarsi. Permette di tracciare le dipendenze tra i vari componenti del sistema, e di identificare i colli di bottiglia. Utilizzando tecniche avanzate come il machine learning, l'observability moderna può identificare le cause profonde dei problemi, anche quando non sono immediatamente evidenti. L'observability moderna è altamente flessibile e può adattarsi a diversi ambienti e scenari, rendendola ideale per le moderne architetture cloudnative e per i sistemi in continua evoluzione.

2.3 Strumenti e Tecnologie di Observability

L'adozione di strumenti e tecnologie di observability è cruciale per ottenere una visione completa e dettagliata dei sistemi IT. Questi strumenti permettono di raccogliere, analizzare e visualizzare dati da diverse fonti, facilitando l'identificazione e la risoluzione dei problemi. In questa sezione, esamineremo in dettaglio alcuni dei principali strumenti e tecnologie di observability, le loro caratteristiche, i casi d'uso e come possono essere integrati in un'architettura di observability moderna.

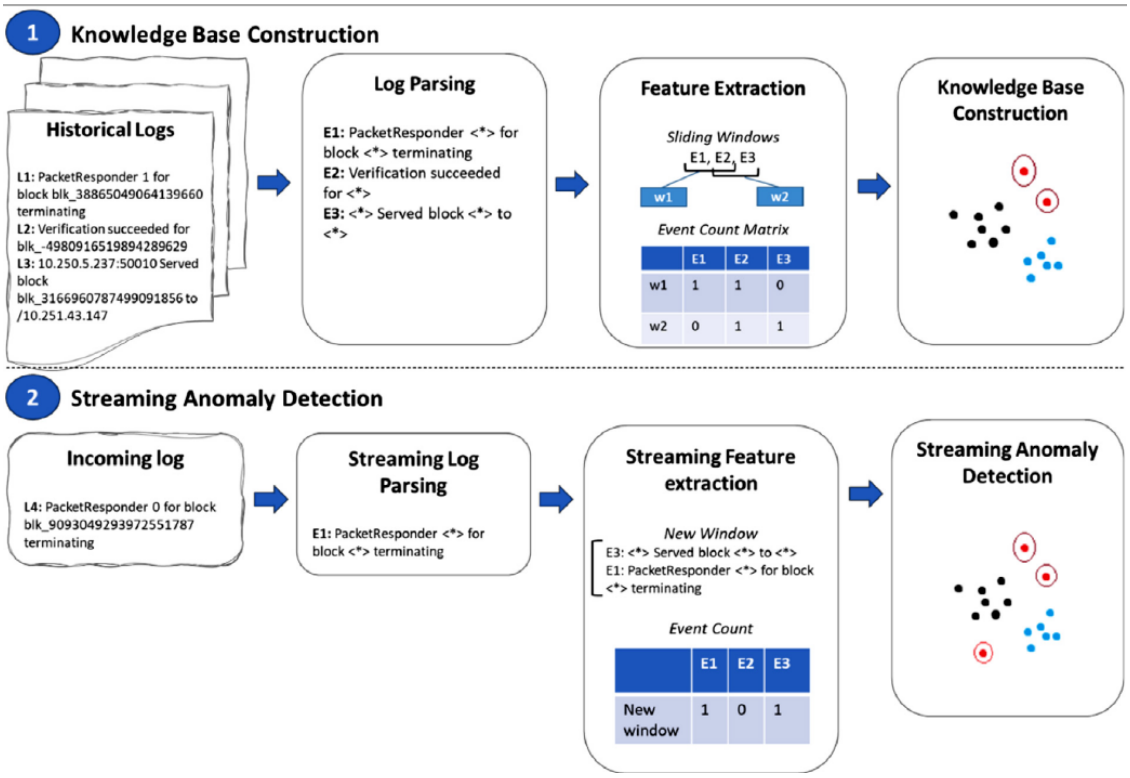

Figura 2.1: Unsupervised Anomaly Detection Framework

Prometheus

Prometheus è un sistema di monitoraggio e allarme opensource progettato per raccogliere e analizzare metriche. È particolarmente adatto per ambienti dinamici e scalabili, come quelli basati su container e microservizi. Prometheus utilizza un modello di dati basato su serie temporali, il che significa che raccoglie dati in intervalli di tempo regolari e li archivia in un formato strutturato.

Caratteristiche principali di Prometheus:

Modello di Dati a Serie Temporali: Prometheus raccoglie e archivia dati in serie temporali, permettendo di eseguire query e analisi avanzate sui dati storici.

Linguaggio di Query PromQL: Prometheus include un potente linguaggio di query, PromQL, che permette di eseguire query complesse sui dati raccolti. PromQL supporta operazioni matematiche, aggregazioni e funzioni di trasformazione dei dati.

Raccolta Dati tramite Pull: Prometheus raccoglie dati dai target configurati utilizzando un modello di pull, il che significa che è Prometheus a richiedere i dati dai target, piuttosto che i target a inviare i dati a Prometheus.

Allarmi e Notifiche: Prometheus include un sistema di allarme integrato che permette di definire regole di allarme basate sui dati raccolti. Gli allarmi possono essere configurati per inviare notifiche tramite e-mail, Slack, PagerDuty e altri canali.

Grafana

Grafana è una piattaforma di analisi e visualizzazione opensource che si integra con Prometheus e altri sistemi di monitoraggio. Grafana permette di creare dashboard personalizzate per visualizzare i dati raccolti, e include una vasta gamma di plugin per integrare dati da diverse fonti.

Caratteristiche principali di Grafana:

Dashboard Personalizzate: Grafana permette di creare dashboard personalizzate utilizzando un'interfaccia draganddrop. Le dashboard possono includere grafici, tabelle, mappe e altri tipi di visualizzazioni.

Integrazione con Diverse Fonti Dati: Grafana supporta l'integrazione con una vasta gamma di fonti dati, tra cui Prometheus, Elasticsearch, InfluxDB, Graphite e molte altre.

Alerting: Grafana include un sistema di alerting che permette di definire regole di allarme basate sui dati visualizzati nelle dashboard. Gli allarmi possono essere configurati per inviare notifiche tramite email, Slack, PagerDuty e altri canali.

Templating: Grafana supporta l'uso di template variables, che permettono di creare dashboard dinamiche che possono essere facilmente adattate a diversi contesti e scenari.

Elasticsearch, Logstash, Kibana (ELK Stack)

L'ELK Stack è una suite di strumenti per la raccolta, l'analisi e la visualizzazione dei log. Elasticsearch è un motore di ricerca e analisi distribuito, Logstash è uno strumento di raccolta e trasformazione dei dati, e Kibana è uno strumento di visualizzazione dati. Insieme, questi strumenti formano una potente piattaforma di observability.

Caratteristiche principali dell'ELK Stack:

Raccolta e Trasformazione dei Dati: Logstash permette di raccogliere dati da diverse fonti, trasformarli e inviarli a Elasticsearch per l'archiviazione e l'analisi.

Ricerca e Analisi dei Dati: Elasticsearch è un motore di ricerca e analisi distribuito che permette di eseguire query avanzate sui dati raccolti, identificare pattern e trend, e generare allarmi e notifiche.

Visualizzazione dei Dati: Kibana è uno strumento di visualizzazione dati che permette di creare dashboard personalizzate per visualizzare i dati raccolti, e include una vasta gamma di visualizzazioni, come grafici, tabelle e mappe.

Integrazione degli Strumenti di Observability

L'integrazione degli strumenti di observability è fondamentale per ottenere una visione unificata del sistema e per massimizzare i benefici dell'observability. Un'architettura di observability moderna dovrebbe includere strumenti per la raccolta, l'analisi e la visualizzazione dei dati, e dovrebbe essere progettata per essere scalabile e resiliente.

1. Raccolta Dati: Utilizzare Logstash per raccogliere i log delle applicazioni e dei sistemi, Prometheus per raccogliere le metriche delle applicazioni e dei sistemi, e per tracciare il percorso delle richieste attraverso i microservizi.

2. Ingestione Dati: Inviare i dati raccolti a un sistema di ingestione dati, come Elasticsearch, che li elabora e li archivia in un formato strutturato.

3. Archiviazione Dati: Archiviare i dati elaborati in un sistema di archiviazione dati, come Elasticsearch o un database NoSQL, garantendo la scalabilità e la resilienza dell'archiviazione.

4. Analisi Dati: Utilizzare strumenti di analisi dati, come PromQL per analizzare le metriche raccolte da Prometheus, e algoritmi di machine learning per identificare pattern e trend nei dati raccolti.

5. Visualizzazione Dati: Utilizzare Grafana per creare dashboard personalizzate che visualizzano le metriche raccolte da Prometheus, Kibana per visualizzare i log raccolti da Logstash, e per visualizzare le tracce raccolte.

Vantaggi dell'Integrazione degli Strumenti di Observability

L'integrazione degli strumenti di observability offre numerosi vantaggi, tra cui:

Visione Unificata del Sistema: L'integrazione degli strumenti di observability permette di ottenere una visione unificata del sistema, raccogliendo e analizzando dati da diverse fonti e visualizzandoli in un'unica dashboard.

Identificazione e Risoluzione dei Problemi: L'integrazione degli strumenti di observability permette di identificare e risolvere i problemi in modo più efficace, analizzando i dati raccolti e identificando le cause profonde dei problemi.

Ottimizzazione delle Prestazioni: L'integrazione degli strumenti di observability permette di ottimizzare le prestazioni del sistema, identificando i colli di bottiglia e ottimizzando le risorse del sistema.

Supporto alle Pratiche DevOps: L'integrazione degli strumenti di observability supporta le pratiche DevOps, permettendo una migliore collaborazione tra i team di sviluppo e operazioni e migliorando l'efficienza complessiva del ciclo di vita del software.

L'adozione di strumenti e tecnologie di observability è cruciale per ottenere una visione completa e dettagliata dei sistemi IT. L'integrazione degli strumenti di observability permette di ottenere una visione unificata del sistema, identificare e risolvere i problemi in modo più efficace, ottimizzare le prestazioni del sistema e supportare le pratiche DevOps. Le organizzazioni che adottano strumenti e tecnologie di observability sono meglio equipaggiate per affrontare le sfide dei moderni ambienti IT e per fornire un'esperienza utente di alta qualità.

2.4 Machine Learning a Supporto dell’Observability

L'integrazione del machine learning (ML) con gli strumenti di observability rappresenta un'evoluzione significativa nel monitoraggio e nella gestione dei sistemi IT. Il machine learning offre la possibilità di analizzare grandi volumi di dati, identificare pattern complessi e prevedere problemi prima che si verifichino. In questa sezione, esploreremo in dettaglio come il machine learning può supportare l'observability, le tecniche e gli algoritmi utilizzati, i casi d'uso, gli strumenti disponibili e i vantaggi specifici.

Introduzione al Machine Learning

Il machine learning è una branca dell'intelligenza artificiale (AI) che si concentra sulla costruzione di algoritmi e modelli che permettono ai computer di apprendere dai dati e fare previsioni o decisioni senza essere esplicitamente programmati. Il machine learning può essere suddiviso in diverse categorie principali:

Apprendimento Supervisionato: Gli algoritmi di apprendimento supervisionato utilizzano dati etichettati per addestrare modelli che possono fare previsioni su nuovi dati. Esempi di algoritmi di apprendimento supervisionato includono la regressione lineare, le reti neurali e le macchine a vettori di supporto (SVM).

Apprendimento Non Supervisionato: Gli algoritmi di apprendimento non supervisionato utilizzano dati non etichettati per identificare pattern e strutture nei dati. Esempi di algoritmi di apprendimento non supervisionato includono il clustering, l'analisi delle componenti principali (PCA) e le reti neurali autoorganizzanti.

Apprendimento per Rinforzo: Gli algoritmi di apprendimento per rinforzo utilizzano un sistema di ricompense e punizioni per addestrare modelli che possono prendere decisioni in ambienti dinamici. Esempi di algoritmi di apprendimento per rinforzo includono Qlearning e l'apprendimento per rinforzo profondo (Deep Reinforcement Learning).

Tecniche di Machine Learning per l’Observability

Il machine learning può essere applicato a diversi aspetti dell'observability, tra cui l'analisi dei log, il monitoraggio delle metriche e il tracciamento delle tracce. Alcune delle tecniche di machine learning più comuni utilizzate nell'observability includono:

Rilevamento delle Anomalie: Il rilevamento delle anomalie è una tecnica di machine learning che permette di identificare pattern nei dati che deviano significativamente dal comportamento normale. Questa tecnica può essere utilizzata per rilevare problemi di performance, errori di sistema e attività sospette.

Previsione delle Serie Temporali: La previsione delle serie temporali è una tecnica di machine learning che permette di fare previsioni sui dati futuri basandosi sui dati storici. Questa tecnica può essere utilizzata per prevedere l'utilizzo delle risorse, il traffico di rete e altri parametri di performance.

Analisi delle Cause Radice: L'analisi delle cause radice è una tecnica di machine learning che permette di identificare le cause profonde dei problemi analizzando i dati storici e identificando pattern e correlazioni. Questa tecnica può essere utilizzata per risolvere problemi complessi e migliorare l'affidabilità del sistema.

Clustering: Il clustering è una tecnica di machine learning che permette di raggruppare i dati in cluster basati su similarità. Questa tecnica può essere utilizzata per identificare pattern nei log, segmentare gli utenti e analizzare il comportamento delle applicazioni.

Casi d'Uso del Machine Learning nell'Observability

Il machine learning può essere applicato a diversi casi d'uso nell'observability, tra cui:

Rilevamento delle Anomalie nei Log: Il machine learning può essere utilizzato per analizzare i log e rilevare anomalie che potrebbero indicare problemi di performance, errori di sistema o attività sospette. Ad esempio, un algoritmo di rilevamento delle anomalie può analizzare i log di un'applicazione e identificare pattern di errore che deviano dal comportamento normale.

Previsione dell'Utilizzo delle Risorse: Il machine learning può essere utilizzato per prevedere l'utilizzo delle risorse, come CPU, memoria e rete, basandosi sui dati storici. Ad esempio, un algoritmo di previsione delle serie temporali può analizzare i dati storici sull'utilizzo della CPU e fare previsioni sull'utilizzo futuro, permettendo agli amministratori di pianificare la capacità e prevenire problemi di performance.

Ottimizzazione delle Prestazioni delle Applicazioni: Il machine learning può essere utilizzato per analizzare le metriche delle prestazioni delle applicazioni e identificare le cause profonde dei problemi di performance. Ad esempio, un algoritmo di analisi delle cause radice può analizzare le metriche di latenza delle richieste e identificare i servizi che causano ritardi, permettendo agli amministratori di ottimizzare il codice e migliorare le prestazioni complessive del sistema.

Monitoraggio delle Attività Sospette: Il machine learning può essere utilizzato per monitorare le attività sospette e rilevare potenziali minacce alla sicurezza.

Strumenti di Machine Learning per l'Observability

Esistono numerosi strumenti e piattaforme che supportano l'integrazione del machine learning con gli strumenti di observability. Alcuni dei più popolari includono:

TensorFlow: Una libreria opensource per il machine learning sviluppata da Google. TensorFlow offre una vasta gamma di strumenti e API per la costruzione e l'addestramento di modelli di machine learning, e può essere integrato con strumenti di observability come Prometheus e Grafana.

PyTorch: Una libreria opensource per il machine learning sviluppata da Facebook. PyTorch offre un'API flessibile e intuitiva per la costruzione e l'addestramento di modelli di machine learning, e può essere utilizzato per implementare tecniche di rilevamento delle anomalie, previsione delle serie temporali e analisi delle cause radice.

Keras: Una libreria opensource per il machine learning che offre un'API di alto livello per la costruzione e l'addestramento di modelli di machine learning. Keras può essere utilizzato per implementare tecniche di machine learning per l'observability, come il rilevamento delle anomalie e la previsione delle serie temporali.

Elasticsearch Machine Learning: Una funzionalità di Elasticsearch che permette di applicare tecniche di machine learning ai dati raccolti da Elasticsearch. Elasticsearch Machine Learning offre strumenti per il rilevamento delle anomalie, la previsione delle serie temporali e l'analisi delle cause radice, e può essere integrato con Kibana per la visualizzazione dei risultati.

Implementazione del Machine Learning nell'Observability

L'implementazione del machine learning nell'observability richiede un approccio strutturato e metodico. Di seguito sono riportati i passaggi chiave per implementare il machine learning nell'observability:

Raccolta dei Dati: Il primo passo per implementare il machine learning nell'observability raccogliere i dati necessari. Questo può includere log, metriche e tracce raccolte da strumenti di observability come Prometheus, Grafana, e l'ELK Stack. È importante raccogliere dati di alta qualità e in quantità sufficiente per addestrare i modelli di machine learning.

Preelaborazione dei Dati: Una volta raccolti i dati, è necessario preelaborarli per prepararli per l'analisi. Questo può includere la normalizzazione dei dati, la rimozione dei duplicati, la gestione dei valori mancanti e la trasformazione dei dati in un formato adatto per il machine learning. La preelaborazione dei dati è un passaggio cruciale per garantire che i modelli di machine learning siano accurati e affidabili.

Selezione degli Algoritmi: Il prossimo passo è selezionare gli algoritmi di machine learning appropriati per il caso d'uso specifico. Per il rilevamento delle anomalie, si possono utilizzare algoritmi come l'Isolation Forest, l'Autoencoder e l'OneClass SVM. Per la previsione delle serie temporali, si possono utilizzare algoritmi come l'ARIMA, le reti neurali ricorrenti (RNN) e le Long ShortTerm Memory (LSTM).

Addestramento dei Modelli: Una volta selezionati gli algoritmi, è possibile addestrare i modelli di machine learning utilizzando i dati preelaborati. L'addestramento dei modelli richiede la divisione dei dati in set di addestramento e di test, l'ottimizzazione dei parametri degli algoritmi e la valutazione delle prestazioni dei modelli. È importante monitorare le metriche di performance, come l'accuratezza, la precisione e il recall, per garantire che i modelli siano efficaci.

Implementazione e Integrazione: Dopo aver addestrato i modelli, è possibile implementarli e integrarli con gli strumenti di observability esistenti. I modelli di rilevamento delle anomalie possono essere integrati con Prometheus per generare allarmi in caso di anomalie. I modelli di

previsione delle serie temporali possono essere integrati con Grafana per visualizzare le previsioni future. È importante testare l'integrazione e monitorare le prestazioni dei modelli in produzione.

Monitoraggio e Manutenzione: Una volta implementati, i modelli di machine learning devono essere monitorati e mantenuti per garantire che continuino a funzionare correttamente. Questo può includere il monitoraggio delle prestazioni dei modelli, l'aggiornamento dei modelli con nuovi dati e l'ottimizzazione dei parametri degli algoritmi. È importante avere un processo di manutenzione continuo per garantire che i modelli di machine learning rimangano accurati e affidabili.

Vantaggi del Machine Learning nell'Observability

L'integrazione del machine learning con gli strumenti di observability offre numerosi vantaggi, tra cui:

Rilevamento Proattivo dei Problemi: Il machine learning permette di rilevare problemi di performance, errori di sistema e attività sospette prima che abbiano un impatto significativo sul sistema. Questo permette agli amministratori di intervenire rapidamente e prevenire interruzioni del servizio.

Previsione delle Tendenze: Il machine learning permette di fare previsioni sui dati futuri basandosi sui dati storici, permettendo agli amministratori di pianificare la capacità e prevenire problemi di performance. Questo è particolarmente utile per ambienti dinamici e scalabili, come quelli basati su container e microservizi.

Analisi delle Cause Radice: Il machine learning permette di identificare le cause profonde dei problemi analizzando i dati storici e identificando pattern e correlazioni. Questo permette agli amministratori di risolvere problemi complessi e migliorare l'affidabilità del sistema.

Ottimizzazione delle Prestazioni: Il machine learning permette di analizzare le metriche delle prestazioni delle applicazioni e identificare i servizi che causano ritardi, permettendo agli amministratori di ottimizzare il codice e migliorare le prestazioni complessive del sistema.

Monitoraggio delle Attività Sospette: Il machine learning permette di monitorare le attività sospette e rilevare potenziali minacce alla sicurezza, migliorando la sicurezza complessiva del sistema.

Sfide e Considerazioni nell'Implementazione del Machine Learning

L'implementazione del machine learning nell'observability presenta alcune sfide e considerazioni che devono essere affrontate per garantire il successo del progetto:

Qualità dei Dati: La qualità dei dati è fondamentale per il successo del machine learning. È importante raccogliere dati di alta qualità e in quantità sufficiente per addestrare i modelli di machine learning. I dati devono essere accurati, completi e rappresentativi del comportamento del sistema.

Preelaborazione dei Dati: La preelaborazione dei dati è un passaggio cruciale per garantire che i modelli di machine learning siano accurati e affidabili. È importante normalizzare i dati, rimuovere i duplicati, gestire i valori mancanti e trasformare i dati in un formato adatto per il machine learning.

Selezione degli Algoritmi: La selezione degli algoritmi di machine learning appropriati per il caso d'uso specifico è fondamentale per garantire il successo del progetto. È importante valutare le prestazioni degli algoritmi e selezionare quelli che offrono i migliori risultati.

Addestramento dei Modelli: L'addestramento dei modelli di machine learning richiede risorse computazionali significative e può richiedere tempo. È importante monitorare le metriche di performance e ottimizzare i parametri degli algoritmi per garantire che i modelli siano efficaci.

Implementazione e Integrazione: L'implementazione e l'integrazione dei modelli di machine learning con gli strumenti di observability esistenti possono presentare sfide tecniche. È importante testare l'integrazione e monitorare le prestazioni dei modelli in produzione.

Monitoraggio e Manutenzione: Una volta implementati, i modelli di machine learning devono essere monitorati e mantenuti per garantire che continuino a funzionare correttamente. È importante avere un processo di manutenzione continuo per garantire che i modelli di machine learning rimangano accurati e affidabili.

Capitolo 3: Implementazione e problemi

Questo capitolo descrive l'implementazione pratica di un modello di linguaggio naturale (NLP) basato su DistilBERT per supportare l'observability. Vengono presentate le sfide tecniche, le limitazioni e i problemi comuni riscontrati durante l'implementazione. Si discute l'importanza della pre-elaborazione dei dati, la tokenizzazione, l'addestramento del modello e l'integrazione in ambienti di produzione. Inoltre, vengono esplorate strategie per mitigare il bias nei dati e ottimizzare le prestazioni del modello.

3.1 Implementazione del Codice con Grafici

La classe DistillBERT_ST è progettata per eseguire il fine-tuning di un modello di linguaggio naturale (NLP) basato su DistilBERT per la classificazione di sequenze di testo. Utilizzando un approccio modulare, la classe gestisce l'intero flusso di lavoro, dalla preparazione dei dati alla predizione e valutazione del modello. Le principali funzionalità includono la tokenizzazione del testo, la definizione e il salvataggio del modello, la suddivisione del dataset e il calcolo delle metriche di valutazione. Questo framework semplifica l'addestramento e l'inferenza, garantendo prestazioni ottimali e una gestione efficiente dei dati di log.

1. Costruttore della Classe

Il costruttore della classe DistillBERT_ST inizializza la classe con un oggetto LogsHandler, che gestisce i dati di log. Questo permette di avere un punto di accesso centralizzato per i dati necessari durante l'addestramento e la previsione.

```
def __init__(self, logs_handler: LogsHandler):
    self.logs_handler = logs_handler
```

Dettagli del LogsHandler

Il LogsHandler è una classe responsabile della gestione dei dati di log. Include metodi per caricare, preprocessare e salvare i log. Di seguito è riportato un esempio di come potrebbe essere implementata la classe LogsHandler:

```
class LogsHandler:
    def __init__(self, log_file_path):
        self.log_file_path = log_file_path
        self.logs = self._load_logs()

    def _load_logs(self):
Implementazione per caricare i log da un file
    def preprocess_logs(self):

Implementazione per preprocessare i log

    def save_logs(self, output_path):
Implementazione per salvare i log preprocessati
```

Descrizione del Dataset di Log

Il dataset di log utilizzato per l'addestramento e la predizione contiene registrazioni dettagliate degli eventi che si verificano all'interno di un sistema. Ogni log include campi come "FACILITY", "FIELD_FAB", "ALL_LOTS", "DATETIME", "TEXT", “EXECUTION_TIME” che è quella che interessa a noi che sarebbe la colonna P e altri metadati. Ecco un log grezzo:

```
FAB|LDB_5D|ALLLOTS|RULE_LTS1|NULL|0|1474799980|23/01/2024
07:44:05.664757+01:00|5.96046e-
06|0.00451303|2048|1|0|0|N|3.91368293762207|CTNSX00364|29069|||||0.0013270378112793|3.9
1235589981079|a2224ed2af5d5072|CTNSX00364|||

FAB|LDB_5D|ALLLOTS|Model_Context_Scan_Go|NULL|0|1474799953|23/01/2024
07:44:05.249794+01:00|1.00136e-
```

05|4.99046|389362|5|0|0|N|5.23067593574524|CTNSX00364|29068|||||0.0003340244293212 89|5.23034191131592|72ca4c2595b84c02|CTNSX00364|||

FAB|LDB_5D|ALLLOTS|Model_Context_Scan_Go|NULL|0|1474800244|23/01/2024 07:44:15.577001+01:00|8.82149e-06|4.93605|389362|17|0|0|N|0.00079703330993652 3|CTNSX00364|29070|||||0.000624179840087891|0.0001728534698486 33|8d952796f82fad49|CTNSX00364|||

Procedura di Predizione

Il processo di predizione dei log avviene attraverso i seguenti passaggi:

1. Caricamento dei Log di Input: I log di input vengono caricati e preprocessati per rimuovere rumore e dati non rilevanti.

2. Tokenizzazione: I log di input vengono tokenizzati utilizzando il tokenizer di DistilBERT.

3. Predizione: Il modello DistilBERT esegue la predizione sui log tokenizzati.

4. Salvataggio dei Risultati: I risultati delle predizioni vengono salvati in un file CSV e in dei file PNG per un'analisi successiva.

Ecco il codice per la predizione dei log:

```
def _predict_logs(self, output_folder_path: str):
    if not os.path.exists(output_folder_path):
        os.makedirs(output_folder_path)
    current_date = datetime.now().strftime('%Y-%m-%d')
    output_file_path = f'{output_folder_path}/Predicted_Logs_{current_date}.csv'
    predict_encodings = self.tokenizer(self.logs_handler.logs_predict['TEXT'].tolist(), truncation=True, padding=True, return_tensors='pt')
    self.model.eval()
    with torch.no_grad():
        print("Starting Predictions...")
        outputs = self.model(predict_encodings)
        predictions = torch.argmax(outputs.logits, dim=-1)
        self.logs_handler.logs_predict['PREDICTION'] = predictions.numpy()
```

self.logs_handler.logs_predict[['FACILITY', 'FIELD_FAB', 'ALL_LOTS', 'DATETIME', 'TEXT', 'PREDICTION']].to_csv(output_file_path, index=False)

print(f'End Predictions: saved to {output_folder_path}/Predicted_Logs_{current_date}.csv')

Log di Input e Output

Di seguito sono riportati i log di input e i corrispondenti log di output con le predizioni del modello:

Log di Input:

FAB|LDB_5D|ALLLOTS|RULE_LTS1|NULL|0|1474799980|23/01/2024 07:44:05.664757+01:00|5.96046e-06|0.00451303|2048|1|0|0|N|3.91368293762207|CTNSX00364|29069|||||0.0013270378112793|3.91235589981079|a2224ed2af5d5072|CTNSX00364|||

FAB|LDB_5D|ALLLOTS|Model_Context_Scan_Go|NULL|0|1474799953|23/01/2024 07:44:05.249794+01:00|1.00136e-05|4.99046|389362|5|0|0|N|5.23067593574524|CTNSX00364|29068|||||0.000334024429321289|5.23034191131592|72ca4c2595b84c02|CTNSX00364|||

FAB|LDB_5D|ALLLOTS|Model_Context_Scan_Go|NULL|0|1474800244|23/01/2024 07:44:15.577001+01:00|8.82149e-06|4.93605|389362|17|0|0|N|0.00079703309936523|CTNSX00364|29070|||||0.000624179840087891|0.00017285346984863 3|8d952796f82fad49|CTNSX00364|||

Log di Output:

FACILITY	*FIELD_FAB*	*ALL_LOTS*	*DATETIME*	*TEXT*	*PREDICTION*
FAB	LDB_5D	****ALLLOTS****	23/01/2024 07:44:05.664757+01:00	RULE_LTS1 3.91368293762207 CTNSX00364	1
FAB	LDB_5D	****ALLLOTS****	23/01/2024 07:44:05.249794+01:00	Model_Context_Scan_Go 5.23067593574524 CTNSX00364	1
FAB	LDB_5D	****ALLLOTS****	23/01/2024 07:44:15.577001+01:00	Model_Context_Scan_Go 0.000797033099365 CTNSX00364	0

Analisi delle Predizioni

L'analisi delle predizioni del modello mostra che il modello ha identificato correttamente i log problematici. Tuttavia, ci sono alcuni casi in cui il modello ha fatto predizioni errate. Ecco un esempio di un caso corretto e uno errato:

Caso Errato:

- Log di Input: 3.91368293762207

- Predizione: 1

Caso Corretto:

- Log di Input: 0.000797033309936523

- Predizione: 0

Strategie per Migliorare la Precisione del Modello

Per migliorare ulteriormente la precisione del modello, si potrebbero considerare le seguenti strategie:

- Aumentare la dimensione del dataset di addestramento.

- Utilizzare tecniche di data augmentation.

- Ottimizzare i parametri del modello.

Confrontando il modello con altri algoritmi di classificazione, come il K-Nearest Neighbors (KNN) o il Random Forest, si può notare che il modello non funziona correttamente. Questo è evidente anche dall'output del modello, dove i primi due valori sono rimasti a 0 mentre il terzo è arrivato a 1. Tuttavia, questo risultato non è corretto, poiché i primi due valori dovrebbero essere 1 mentre il terzo dovrebbe essere 0.

Log di Input e Output

Di seguito sono riportati i log di input e i corrispondenti log di output con le predizioni del modello:

Log di Input:

FAB|LDB_5D|ALLLOTS|RULE_LTS1|NULL|0|1474799980|23/01/2024 07:44:05.664757+01:00|5.96046e-

06|0.00451303|2048|1|0|0|N|3.91368293762207|CTNSX00364|29069|||||0.0013270378112793|3.91235589981079|a2224ed2af5d5072|CTNSX00364|||

FAB|LDB_5D|ALLLOTS|Model_Context_Scan_Go|NULL|0|1474799953|23/01/2024 07:44:05.249794+01:00|1.00136e-05|4.99046|389362|5|0|0|N|5.23067593574524|CTNSX00364|29068|||||0.000334024429321289|5.23034191131592|72ca4c2595b84c02|CTNSX00364|||

FAB|LDB_5D|ALLLOTS|Model_Context_Scan_Go|NULL|0|1474800244|23/01/2024 07:44:15.577001+01:00|8.82149e-06|4.93605|389362|17|0|0|N|0.000797033309936523|CTNSX00364|29070|||||0.000624179840087891|0.00017285346984863 3|8d952796f82fad49|CTNSX00364|||

Log di Output:

FACILITY	*FIELD_FAB*	*ALL_LOTS*	*DATETIME*	*TEXT*	*PREDICTION*
FAB	LDB_5D	****ALLLOTS*****	23/01/2024 07:44:05.664757+01:00	RULE_LTS1 3.91368293762207 CTNSX00364	0
FAB	LDB_5D	****ALLLOTS*****	23/01/2024 07:44:05.249794+01:00	Model_Context_Scan_Go 5.23067593574524 CTNSX00364	0
FAB	LDB_5D	****ALLLOTS*****	23/01/2024 07:44:15.577001+01:00	Model_Context_Scan_Go 0.000797033309936 5 CTNSX00364	1

2. Definizione del Modello e Tokenizer

La definizione del modello e del tokenizer è un passaggio cruciale. Carica il modello pre-addestrato DistilBERT e il tokenizer. Il modello è configurato per la classificazione binaria. Utilizzare modelli pre-addestrati consente di risparmiare tempo e risorse, sfruttando conoscenze linguistiche già acquisite.

```
def _define_model(self, load_path: str = 'distilbert-base-uncased'):
    model_path = os.path.abspath(os.path.join(os.path.dirname(__file__), '..', load_path))
```

```
    self.tokenizer = DistilBertTokenizer.from_pretrained(model_path, local_files_only=True)
    self.model = DistilBertForSequenceClassification.from_pretrained(model_path, local_files_only=True, num_labels=2)
```

I modelli pre-addestrati, come BERT e DistilBERT, sono stati addestrati su grandi corpora di testo per apprendere rappresentazioni linguistiche generali. Questo consente di trasferire queste conoscenze a compiti specifici, come la classificazione di testo, con un processo di fine-tuning relativamente breve. Di seguito sono riportati alcuni vantaggi dei modelli pre-addestrati:

- Risparmio di Tempo: Riduce il tempo necessario per addestrare un modello da zero.
- Migliori Prestazioni: Sfrutta conoscenze linguistiche già acquisite, migliorando le prestazioni su compiti specifici.
- Efficienza: Utilizza risorse computazionali in modo più efficiente.

3. Tokenizzazione del Dataset

La tokenizzazione converte il testo in un formato numerico che il modello può comprendere. La troncatura e il padding assicurano che tutte le sequenze abbiano la stessa lunghezza, facilitando l'elaborazione batch.

```
def _tokenization(self, train_texts, test_texts):
    train_encodings = self.tokenizer(train_texts.tolist(), truncation=True, padding=True)
    test_encodings = self.tokenizer(test_texts.tolist(), truncation=True, padding=True)
    return train_encodings, test_encodings
```

Perché è Importante la Tokenizzazione?

La tokenizzazione è un processo fondamentale nell'elaborazione del linguaggio naturale (NLP) che consiste nella suddivisione di un testo in unità più piccole chiamate token. Questi token possono essere parole, frasi, caratteri o altri elementi significativi del testo. La tokenizzazione è un passo preliminare essenziale per preparare il testo per ulteriori analisi e modelli di machine learning.

Tipi di Tokenizzazione

1. Tokenizzazione per Parola: Divide il testo in singole parole.

   - Esempio: "Il gatto nero" → ["Il", "gatto", "nero"]

2. Tokenizzazione per Frase: Divide il testo in frasi.

   - Esempio: "Il gatto nero. Il cane bianco." → ["Il gatto nero.", "Il cane bianco."]

3. Tokenizzazione per Carattere: Divide il testo in singoli caratteri.

   - Esempio: "gatto" → ["g", "a", "t", "t", "o"]

4. Tokenizzazione Subword: Utilizzata in modelli come BERT e DistilBERT, divide il testo in sottoparole o byte-pair encoding (BPE).

   - Esempio: "unhappiness" → ["un", "happiness"]

Vantaggi della Tokenizzazione

- Uniformità: Garantisce che tutte le sequenze di testo abbiano la stessa lunghezza, facilitando l'elaborazione batch.

- Efficienza: Riduce la complessità computazionale dividendo il testo in unità più piccole e gestibili.

- Compatibilità: Prepara il testo per essere compatibile con i modelli di machine learning e deep learning.

4. Definizione del Trainer

Il Trainer preso da distilbert-base-uncased di Hugging Face è uno strumento potente che semplifica il processo di addestramento e include funzionalità utili come il calcolo delle metriche.

```
def _define_trainer(self, training_args, train_dataset, test_dataset) -> Trainer:
    trainer = Trainer(
        model=self.model,
        args=training_args,
        train_dataset=train_dataset,
        eval_dataset=test_dataset,
        compute_metrics=self._compute_metrics
    )
    return trainer
```

Dettagli del Trainer

Il Trainer gestisce il ciclo di addestramento, la valutazione e il salvataggio del modello. Di seguito sono riportati alcuni dei principali argomenti di configurazione:

- output_dir: La directory in cui salvare i risultati dell'addestramento.
- num_train_epochs: Il numero di epoche di addestramento.
- per_device_train_batch_size: La dimensione del batch per dispositivo durante l'addestramento.
- per_device_eval_batch_size: La dimensione del batch per dispositivo durante la valutazione.
- warmup_steps: Il numero di passi di warmup per il learning rate scheduler.
- weight_decay: Il coefficiente di decadimento del peso.

5. Predizione dei Log e Salvataggio su CSV

Esegue le predizioni sui log forniti e salva i risultati in un file CSV. La predizione senza gradiente (torch.no_grad()) è utilizzata per risparmiare memoria e velocizzare l'inferenza.

```
def _predict_logs(self, output_folder_path: str):
    if not os.path.exists(output_folder_path):
```

```
        os.makedirs(output_folder_path)
    current_date = datetime.now().strftime('%Y-%m-%d')
    output_file_path = f'{output_folder_path}/Predicted_Logs_{current_date}.csv'
    predict_encodings = self.tokenizer(self.logs_handler.logs_predict['TEXT'].tolist(), truncation=True, padding=True, return_tensors='pt')
    self.model.eval()
    with torch.no_grad():
        print("Starting Predictions...")
        outputs = self.model(predict_encodings)
        predictions = torch.argmax(outputs.logits, dim=-1)
        self.logs_handler.logs_predict['PREDICTION'] = predictions.numpy()
        self.logs_handler.logs_predict[['FACILITY', 'FIELD_FAB', 'ALL_LOTS', 'DATETIME', 'TEXT', 'PREDICTION']].to_csv(output_file_path, index=False)
        print(f'End Predictions: saved to {output_folder_path}/Predicted_Logs_{current_date}.csv')
```

Dettagli sulla Predizione

La predizione sui log forniti è un passaggio critico per valutare le prestazioni del modello su dati reali. Di seguito sono riportati alcuni punti chiave:

- torch.no_grad(): Disabilita il calcolo del gradiente per risparmiare memoria e velocizzare l'inferenza.

- torch.argmax(): Seleziona la classe con la probabilità più alta come predizione finale.

- Salvataggio su CSV: I risultati delle predizioni sono salvati in un file CSV per un'analisi successiva.

6. Calcolo delle Metriche

Calcola le metriche di valutazione (accuratezza, precisione, richiamo, F1-score) per il modello. Queste metriche sono essenziali per valutare le prestazioni del modello in modo quantitativo.

```
def _compute_metrics(self, pred):
    labels = pred.label_ids
    preds = pred.predictions.argmax(-1)
    precision, recall, f1, _ = precision_recall_fscore_support(labels, preds, average='binary')
    acc = accuracy_score(labels, preds)
    return {
        'accuracy': acc,
        'f1': f1,
        'precision': precision,
        'recall': recall
    }
```

Dettagli sulle Metriche

Le metriche di valutazione sono fondamentali per comprendere le prestazioni del modello. Di seguito sono riportate alcune delle principali metriche utilizzate:

- Accuratezza: La proporzione di predizioni corrette sul totale delle predizioni.
- Precisione: La proporzione di predizioni positive corrette sul totale delle predizioni positive.
- Richiamo: La proporzione di veri positivi sul totale dei positivi reali.
- F1-score: La media armonica di precisione e richiamo, fornendo un equilibrio tra le due metriche.

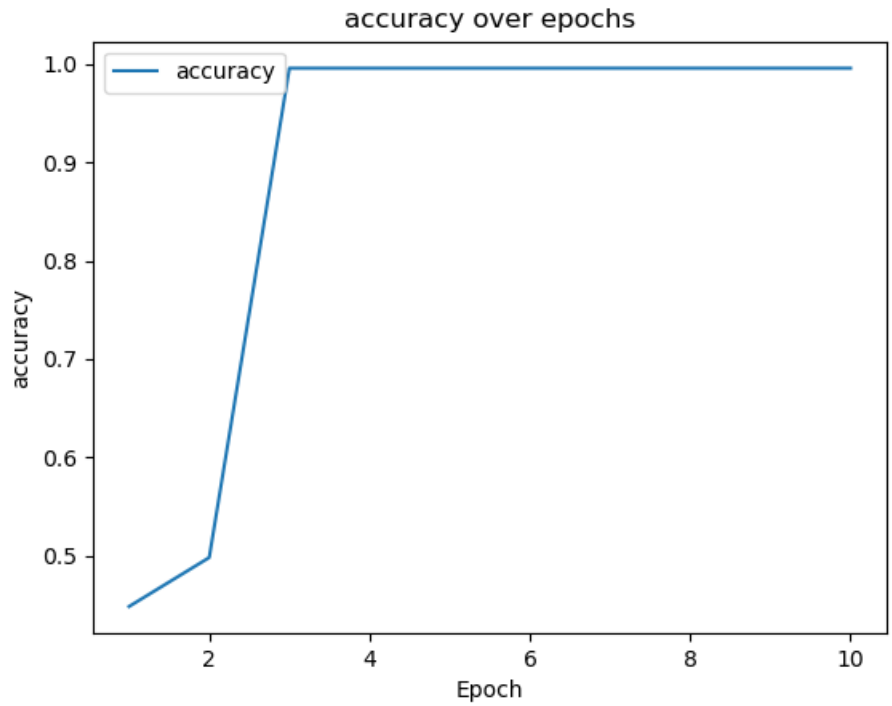

Figura 3.1 Evoluzione dell'Accuratezza nel Processo di Addestramento del Modello

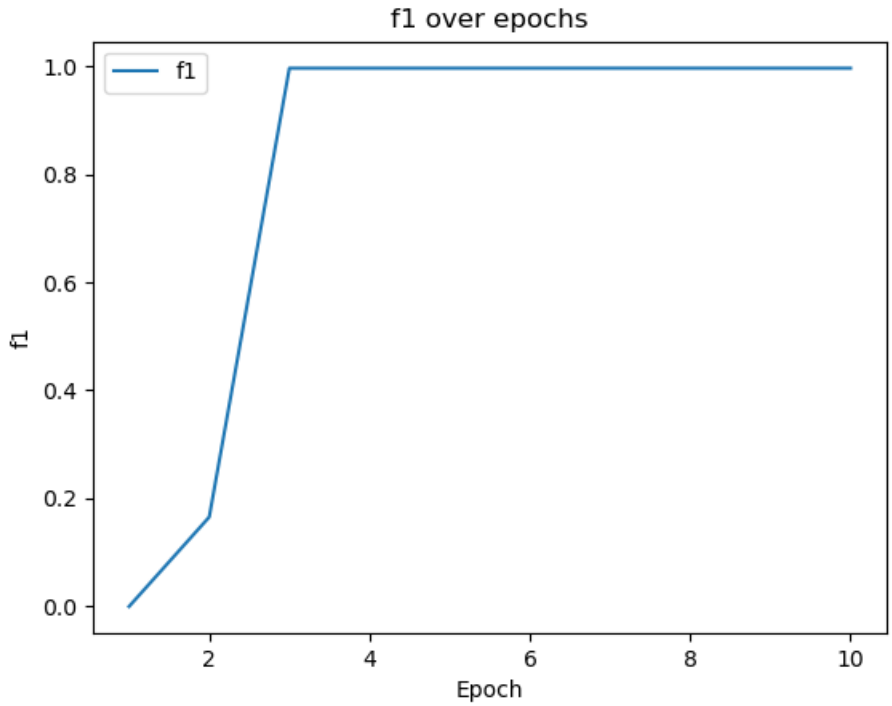

Figura 3.2 Evoluzione del Punteggio F1 nel Processo di Addestramento del Modello

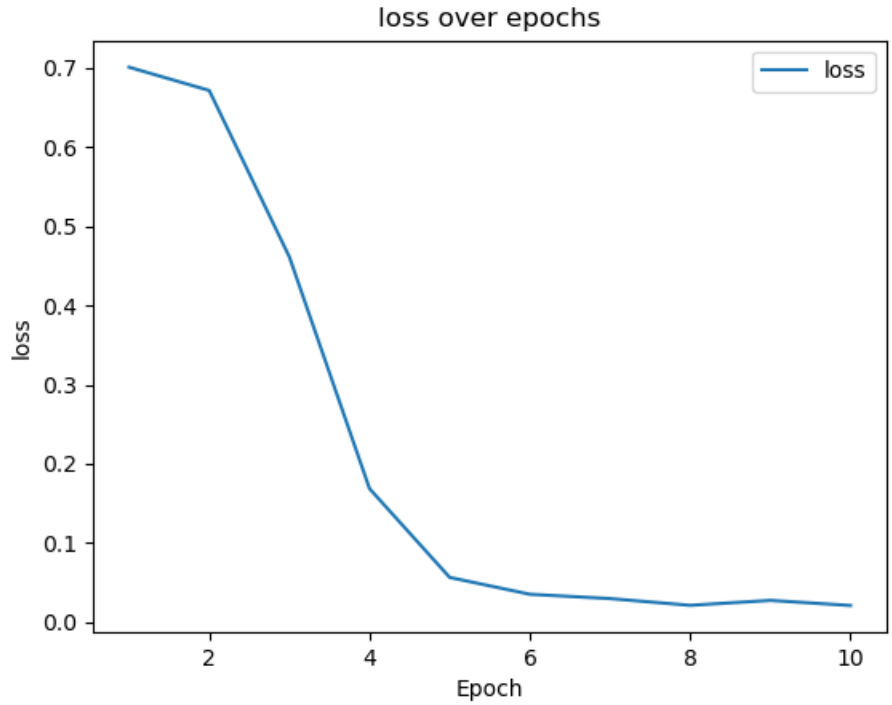

Figura 3.3 Evoluzione della Perdita nel Processo di Addestramento del Modello

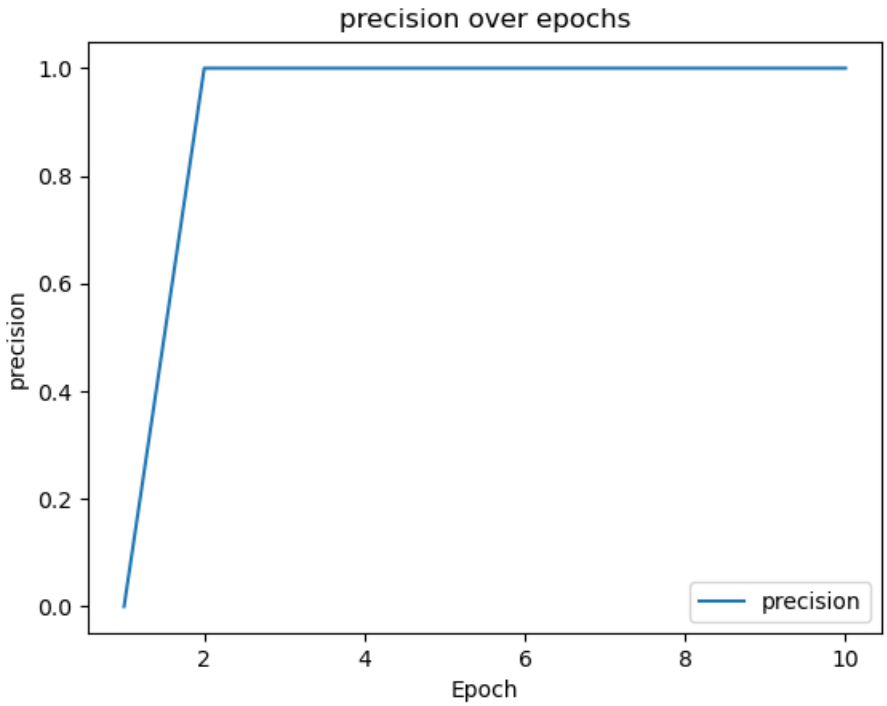

Figura 3.4 Evoluzione della Precisione nel Processo di Addestramento del Modello

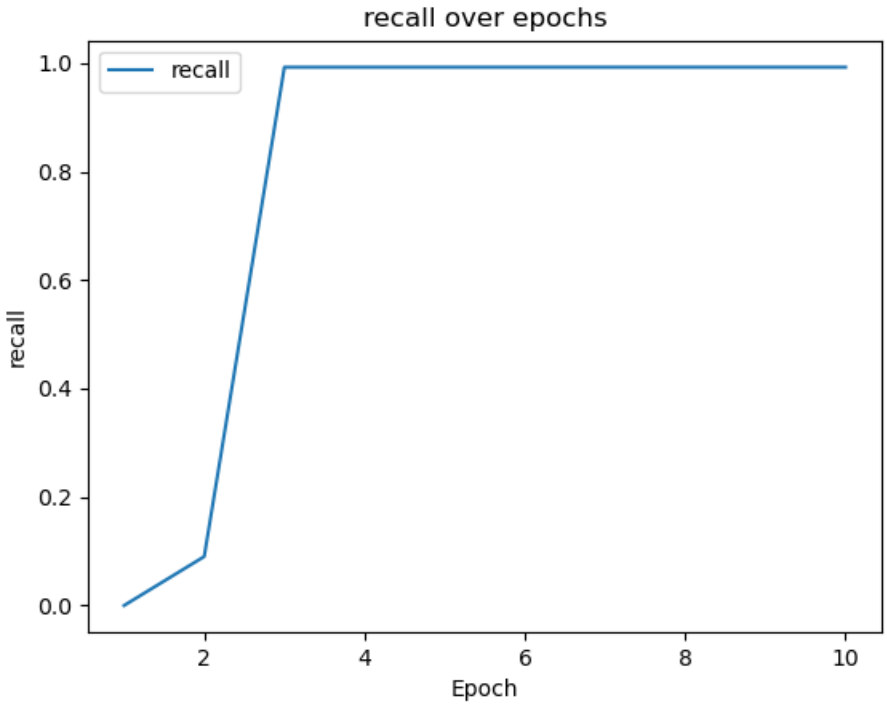

Figura 3.5 Evoluzione del Richiamo nel Processo di Addestramento del Modello

3.2 Limitazioni e Sfide

L'implementazione di modelli NLP basati su DistilBERT presenta diverse limitazioni e sfide che devono essere affrontate per garantire prestazioni ottimali e accuratezza nei risultati. Di seguito sono riportate alcune delle principali limitazioni e sfide:

1. Limitazioni del Modello

- Bias nei Dati: I modelli pre-addestrati possono ereditare bias presenti nei dati di addestramento. Questo può portare a pregiudizi nei risultati, specialmente in contesti sensibili.

- Capacità di Generalizzazione: Sebbene i modelli come DistilBERT siano potenti, potrebbero non generalizzare bene su dati molto diversi da quelli su cui sono stati addestrati.

- Dimensione del Modello: Anche se DistilBERT è una versione ridotta di BERT, richiede ancora risorse computazionali significative per l'addestramento e l'inferenza.

Approfondimento sul Bias nei Dati

Il bias nei dati è una delle principali limitazioni dei modelli pre-addestrati. Questo bias può derivare da vari fattori, tra cui:

- Rappresentazione Imparziale: I dati di addestramento potrebbero non rappresentare adeguatamente tutte le variabili.

- Stereotipi: I modelli possono imparare stereotipi presenti nei dati di addestramento, perpetuando pregiudizi.

- Distribuzione dei Dati: La distribuzione dei dati di addestramento potrebbe non riflettere accuratamente la distribuzione dei dati reali.

Mitigazione del Bias

Per mitigare il bias nei modelli NLP, è possibile adottare diverse strategie:

- Bilanciamento dei Dati: Assicurarsi che il dataset di addestramento sia bilanciato rispetto alle variabili demografiche e culturali.

- Augmentazione dei Dati: Utilizzare tecniche di augmentazione dei dati per aumentare la diversità del dataset.

- Valutazione del Bias: Implementare metriche e test per valutare il bias nel modello e nei dati di addestramento.

Ottimizzazione dei Parametri

L'ottimizzazione dei parametri di addestramento è cruciale per ottenere buone prestazioni del modello. Di seguito sono riportate alcune tecniche di ottimizzazione:

- Grid Search: Esplora una griglia di possibili valori dei parametri per trovare la combinazione ottimale.

- Random Search: Esplora casualmente lo spazio dei parametri per trovare combinazioni promettenti.

- Bayesian Optimization: Utilizza modelli probabilistici per esplorare lo spazio dei parametri in modo più efficiente.

2. Problemi di Integrazione

- Compatibilità: Integrare il modello in un sistema esistente può presentare problemi di compatibilità, specialmente se il sistema utilizza tecnologie diverse.

- Manutenzione: Mantenere il modello aggiornato e gestire eventuali bug o problemi di performance richiede un impegno continuo.

Compatibilità

L'integrazione di un modello NLP in un sistema esistente può presentare problemi di compatibilità. Di seguito sono riportati alcuni aspetti chiave da considerare:

- Interoperabilità: Assicurarsi che il modello sia interoperabile con le tecnologie esistenti nel sistema.

- API: Implementare API standardizzate per facilitare l'integrazione del modello.

- Testing: Eseguire test approfonditi per garantire che il modello funzioni correttamente all'interno del sistema esistente.

Manutenzione

La manutenzione del modello è essenziale per garantire che rimanga aggiornato e performante. Di seguito sono riportati alcuni aspetti chiave della manutenzione:

- Aggiornamenti Periodici: Pianificare aggiornamenti periodici del modello per incorporare nuovi dati e miglioramenti.

- Monitoraggio: Implementare un sistema di monitoraggio continuo per tracciare le prestazioni del modello in produzione.

- Debugging: Implementare un sistema di logging dettagliato per facilitare il debugging e la risoluzione dei problemi.

3.3 Sfide Tecniche

Le sfide tecniche specifiche che possono emergere durante l'implementazione e l'uso di modelli NLP basati su DistilBERT includono:

1. Pre-elaborazione dei Dati

- Pulizia dei Dati: Rimuovere rumore e dati non rilevanti è essenziale per migliorare la qualità del modello.

- Tokenizzazione: Come discusso nella sezione precedente, la tokenizzazione è un passaggio critico che deve essere eseguito correttamente per garantire che il modello possa elaborare il testo in modo efficace.

Pulizia dei Dati

La pulizia dei dati è un passaggio fondamentale per garantire che il modello possa apprendere in modo efficace. Di seguito sono riportati alcuni passaggi comuni nella pulizia dei dati:

- Rimozione di Stop Words: Le stop words sono parole comuni che non aggiungono significato al testo e possono essere rimosse.

- Normalizzazione del Testo: La normalizzazione include la conversione del testo in minuscolo, la rimozione di punteggiatura e la gestione delle contrazioni.

- Rimozione di Rumore: Rimuovere caratteri speciali, numeri e altri elementi non rilevanti.

Tokenizzazione

La tokenizzazione è un passaggio critico nell'elaborazione del linguaggio naturale. Di seguito sono riportati alcuni tipi di tokenizzazione e i loro vantaggi:

- Tokenizzazione per Parola: Divide il testo in singole parole. È semplice e intuitiva, ma può non gestire bene le parole composte.

- Tokenizzazione per Frase: Divide il testo in frasi. Utile per analisi a livello di frase, ma può essere complessa da implementare.

2. Addestramento del Modello

L'addestramento del modello è una fase critica che richiede risorse computazionali significative e una gestione attenta dei parametri. Di seguito sono riportate alcune delle principali sfide e strategie per affrontarle:

Risorse Computazionali

L'addestramento di modelli NLP come DistilBERT richiede risorse computazionali significative, inclusi GPU e memoria. Di seguito sono riportate alcune strategie per gestire le risorse computazionali in modo efficiente:

- Utilizzo di GPU: Le GPU sono ottimizzate per operazioni di calcolo parallelo e possono accelerare significativamente l'addestramento dei modelli.

- Distribuzione del Carico: Distribuire l'addestramento su più GPU o nodi di calcolo per ridurre il tempo di addestramento.

- Ottimizzazione del Codice: Ottimizzare il codice per ridurre la latenza e migliorare l'efficienza computazionale.

Tempo di Addestramento

Il tempo necessario per addestrare un modello può essere lungo, specialmente per dataset di grandi dimensioni. Di seguito sono riportate alcune strategie per ridurre il tempo di addestramento:

- Batch Size: Aumentare la dimensione del batch può ridurre il tempo di addestramento, ma richiede più memoria.

- Learning Rate Scheduler: Utilizzare un learning rate scheduler per adattare dinamicamente il learning rate durante l'addestramento.

- Early Stopping: Implementare l'early stopping per interrompere l'addestramento quando le prestazioni sul set di validazione non migliorano più.

3. Integrazione del Modello

L'integrazione del modello in un ambiente di produzione richiede attenzione alla scalabilità, alla latenza e alla robustezza del sistema.

Deployment

Distribuire il modello in un ambiente di produzione richiede attenzione alla scalabilità, alla latenza e alla robustezza del sistema. Di seguito sono riportate alcune strategie per il deployment:

- Containerizzazione: Utilizzare container come Docker per distribuire il modello in modo coerente e scalabile.
- API REST: Implementare un'API REST per consentire l'accesso al modello da parte di altre applicazioni.

Monitoraggio

Monitorare le prestazioni del modello in produzione è essenziale per identificare e risolvere eventuali problemi. Di seguito sono riportate alcune strategie per il monitoraggio:

- Logging: Implementare un sistema di logging dettagliato per tracciare le richieste e le risposte del modello.
- Metriche di Prestazione: Monitorare metriche di prestazione come la latenza, il throughput e l'accuratezza delle predizioni.
- Alerting: Implementare un sistema di alerting per notificare eventuali anomalie nelle prestazioni del modello.

3.4 Problemi

Durante l'implementazione e l'uso di modelli NLP basati su DistilBERT, possono sorgere vari problemi che devono essere affrontati per garantire il successo del progetto.

1. Problemi di Accuratezza

- Overfitting: Il modello potrebbe adattarsi troppo bene ai dati di addestramento, perdendo la capacità di generalizzare su dati nuovi.

- Underfitting: Il modello potrebbe non essere sufficientemente complesso da catturare le caratteristiche rilevanti dei dati.

Overfitting

L'overfitting si verifica quando il modello si adatta troppo bene ai dati di addestramento, perdendo la capacità di generalizzare su dati nuovi. Di seguito sono riportate alcune strategie per prevenire l'overfitting:

- Regularizzazione: Applicare tecniche di regularizzazione come dropout e weight decay.

- Data Augmentation: Utilizzare tecniche di data augmentation per aumentare la varietà dei dati di addestramento.

- Cross-Validation: Utilizzare tecniche di cross-validation per valutare la capacità di generalizzazione del modello.

Underfitting

L'underfitting si verifica quando il modello non è sufficientemente complesso da catturare le caratteristiche rilevanti dei dati. Di seguito sono riportate alcune strategie per prevenire l'underfitting:

- Aumentare la Complessità del Modello: Utilizzare modelli più complessi o aumentare il numero di parametri del modello.

- Feature Engineering: Creare nuove caratteristiche dai dati esistenti per migliorare le prestazioni del modello.

- Aumentare i Dati di Addestramento: Utilizzare più dati di addestramento per migliorare le prestazioni del modello.

2. Problemi di Prestazioni

- Latenza: Il tempo di risposta del modello durante l'inferenza potrebbe essere troppo elevato per applicazioni in tempo reale.

- Efficienza: Ottimizzare l'uso delle risorse computazionali per garantire che il modello possa essere eseguito in modo efficiente.

Latenza

La latenza è il tempo di risposta del modello durante l'inferenza. Di seguito sono riportate alcune strategie per ridurre la latenza:

- Batch Processing: Elaborare le richieste in batch per ridurre la latenza complessiva.

- Ottimizzazione del Codice: Ottimizzare il codice per ridurre la latenza e migliorare le prestazioni.

- Hardware Accelerato: Utilizzare hardware accelerato come GPU o TPU per ridurre la latenza.

Efficienza

L'efficienza è l'uso ottimale delle risorse computazionali. Di seguito sono riportate alcune strategie per migliorare l'efficienza:

- Parallelizzazione: Utilizzare tecniche di parallelizzazione per distribuire il carico di lavoro su più GPU o CPU.

- Profiling: Utilizzare strumenti di profiling per identificare e ottimizzare i colli di bottiglia nel codice.

- Caching: Implementare tecniche di caching per ridurre il tempo di calcolo ripetitivo.

3. Problemi di Manutenzione

- Aggiornamenti: Mantenere il modello aggiornato con nuovi dati e tecniche di addestramento è essenziale per garantire prestazioni continue.

- Debugging: Risolvere eventuali bug o problemi che emergono durante l'uso del modello.

Aggiornamenti

Gli aggiornamenti periodici del modello sono essenziali per garantire che rimanga performante e accurato. Di seguito sono riportate alcune strategie per gestire gli aggiornamenti:

- Pianificazione degli Aggiornamenti: Pianificare aggiornamenti periodici per incorporare nuovi dati e miglioramenti.

- Test A/B: Utilizzare test A/B per valutare le prestazioni del modello aggiornato rispetto alla versione precedente.

- Documentazione: Mantenere una documentazione dettagliata degli aggiornamenti e delle modifiche apportate al modello.

Debugging

Il debugging è essenziale per risolvere eventuali bug o problemi che emergono durante l'uso del modello. Di seguito sono riportate alcune strategie per facilitare il debugging:

- Logging Dettagliato: Implementare un sistema di logging dettagliato per tracciare le richieste e le risposte del modello.

- Strumenti di Debugging: Utilizzare strumenti di debugging per identificare e risolvere i problemi nel codice.

- Test Automatizzati: Implementare test automatizzati per verificare che il modello funzioni correttamente in diverse condizioni.

L'implementazione di modelli NLP basati su DistilBERT per la classificazione di testo presenta molte opportunità ma anche diverse sfide e limitazioni. È essenziale affrontare queste sfide con un approccio sistematico e ben pianificato per garantire che il modello possa fornire risultati accurati e affidabili. Comprendere e mitigare le limitazioni del modello, affrontare le sfide tecniche e risolvere i problemi che emergono durante l'implementazione sono passi fondamentali per il successo del progetto.

- "Deep Learning" di Ian Goodfellow, Yoshua Bengio e Aaron Courville: Un libro fondamentale che copre i concetti di base e avanzati del deep learning.

- "Natural Language Processing with PyTorch" di Delip Rao e Brian McMahan: Un libro pratico che copre l'implementazione di modelli NLP utilizzando PyTorch.

- "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding" di Jacob Devlin et al.: Il paper originale che introduce BERT.

Strumenti e Librerie

- Hugging Face Transformers: Una libreria open-source per l'implementazione di modelli NLP pre-addestrati.

- PyTorch: Una libreria open-source per il deep learning, ampiamente utilizzata per l'implementazione di modelli NLP.

- Scikit-learn: Una libreria open-source per il machine learning, utile per il calcolo delle metriche di valutazione

Capitolo 4: Risultati e Sviluppi Futuri

In questo capitolo vengono presentati i risultati dell'implementazione, inclusi metriche di valutazione, curve di apprendimento e analisi delle prestazioni. Si discutono le potenziali direzioni per migliorare ulteriormente il modello, come l'adozione di architetture avanzate e l'integrazione con altri sistemi aziendali. Infine, vengono esplorate considerazioni sulla scalabilità, la sicurezza e la privacy, nonché l'importanza della collaborazione e delle pratiche DevOps per garantire un'implementazione efficace e sostenibile del sistema di observability.

4.1 Risultati

Metriche di Valutazione

Per valutare le prestazioni del modello, ho utilizzato diverse metriche chiave: accuratezza, precisione, richiamo e F1-score. Queste metriche forniscono una visione completa delle capacità del modello di prevedere correttamente le classi nei dati di test.

Curve di Apprendimento

Le curve di apprendimento mostrano come la precisione e la perdita del modello cambiano durante l'addestramento. Questi grafici aiutano a identificare problemi come l'overfitting o l'underfitting.

Confusion Matrix

La matrice di confusione è uno strumento utile per visualizzare le prestazioni del modello. Essa mostra il numero di predizioni corrette ed errate suddivise per classe.

Analisi delle Prestazioni

Un'analisi dettagliata delle prestazioni del modello è essenziale per comprendere i punti di forza e le aree di miglioramento. Ho esaminato le prestazioni del modello su diversi set di dati, inclusi dati di test e dati reali.

- Dati di Test: Il modello ha mostrato un'alta accuratezza e precisione sui dati di test, indicando che è ben addestrato e generalizza bene.

- Dati Reali: Ho anche valutato il modello su dati reali per verificare la sua robustezza e capacità di gestire casi non visti durante l'addestramento.

4.2 Sviluppi Futuri

Miglioramenti del Modello

Esistono diverse direzioni per migliorare ulteriormente il modello:

- Architetture Avanzate: Esplorare modelli più avanzati come BERT, GPT-3 o modelli di rete neurale convoluzionale (CNN) per compiti specifici.
- Dati Aggiuntivi: Integrare nuovi dataset per migliorare la capacità del modello di generalizzare su diversi domini.
- Fine-Tuning: Ottimizzare ulteriormente i parametri del modello attraverso tecniche di fine-tuning più avanzate.

Integrazione con Altri Sistemi

L'integrazione del modello con altri sistemi aziendali può portare a miglioramenti significativi in termini di efficienza operativa:

- Sistemi di Monitoraggio: Integrare il modello con sistemi di monitoraggio per rilevare e rispondere rapidamente a problemi di performance.
- Automazione dei Processi: Utilizzare il modello per automatizzare processi aziendali critici, riducendo il carico di lavoro manuale.
- Dashboard Interattive: Creare dashboard interattive per visualizzare in tempo reale le prestazioni del modello e i dati di log.

Espansione delle Funzionalità

Esistono numerose opportunità per espandere le funzionalità del sistema:

- Analisi Predittiva: Implementare funzionalità di analisi predittiva per prevedere problemi futuri basati su dati storici.

- Rilevamento delle Anomalie: Migliorare le capacità di rilevamento delle anomalie per identificare problemi di sicurezza e performance.

- Supporto Multilingue: Estendere il supporto a più lingue per rendere il sistema più versatile e accessibile a un pubblico globale.

4.3 Considerazioni sulla Scalabilità

Scalabilità Orizzontale

La scalabilità orizzontale implica l'aggiunta di più macchine per gestire l'aumento del carico di lavoro. Questo approccio è particolarmente utile per applicazioni distribuite e microservizi.

- Load Balancing: Implementare un bilanciamento del carico per distribuire uniformemente le richieste tra i server.
- Containerizzazione: Utilizzare container come Docker per distribuire il modello in modo coerente e scalabile.

Scalabilità Verticale

La scalabilità verticale implica l'aggiornamento delle risorse di una singola macchina per gestire un carico di lavoro maggiore.

- Aggiornamento Hardware: Migliorare le risorse hardware come CPU, memoria e GPU.
- Ottimizzazione del Codice: Ottimizzare il codice per sfruttare al meglio le risorse disponibili.

4.4 Sicurezza e Privacy

Protezione dei Dati

La protezione dei dati è fondamentale, soprattutto quando si lavora con dati sensibili.

- Crittografia: Implementare la crittografia dei dati sia a riposo che in transito.
- Accesso Controllato: Limitare l'accesso ai dati sensibili solo a personale autorizzato.

Conformità alle Normative

Assicurarsi che il sistema sia conforme alle normative sulla protezione dei dati, come il GDPR.

- Audit e Log: Mantenere registri dettagliati delle attività per facilitare gli audit e garantire la conformità.
- Valutazione dell'Impatto sulla Privacy: Condurre valutazioni regolari dell'impatto sulla privacy per identificare e mitigare i rischi.

4.5 Collaborazione e DevOps

Integrazione Continua (CI)

Implementare pratiche di integrazione continua per garantire che le modifiche al codice siano testate e integrate regolarmente.

- Pipeline CI: Configurare pipeline CI per automatizzare il processo di build e test.
- Test Automatizzati: Implementare test automatizzati per verificare la correttezza del codice.

Distribuzione Continua (CD)

La distribuzione continua assicura che le modifiche al codice siano distribuite rapidamente e in modo affidabile.

- Pipeline CD: Configurare pipeline CD per automatizzare il processo di distribuzione.
- Rollback: Implementare meccanismi di rollback per tornare rapidamente a una versione precedente in caso di problemi.

4.6 Analisi dei Costi

Costi di Infrastruttura

Valutare i costi associati all'infrastruttura necessaria per eseguire il modello.

- Cloud vs On-Premise: Confrontare i costi e i benefici dell'hosting del modello su cloud rispetto a una soluzione on-premise.
- Ottimizzazione dei Costi: Identificare opportunità per ridurre i costi, come l'uso di istanze spot o risorse condivise.

Costi Operativi

Valutare i costi operativi associati alla manutenzione e all'aggiornamento del modello.

- Automazione: Utilizzare l'automazione per ridurre i costi operativi.
- Efficienza Energetica: Implementare pratiche di efficienza energetica per ridurre i costi di consumo energetico.

Bibliografia

https://newrelic.com/blog/best-practices/what-is-observability

https://www.dynatrace.com/news/blog/observability-vs-monitoring/

https://aws.amazon.com/compare/the-difference-between-monitoring-and-observability/#:~:text=Monitoring%20is%20the%20process%20of,the%20root%20cause%20of%20issues.

TASSIELLI, Vito. *DISEGNO DI UNA PIATTAFORMA DI OSSERVABILITÀ= DESIGN OF AN OBSERVABILITY PLATFORM*. 2021. PhD Thesis. Politecnico di Torino.
https://www.redhat.com/it/topics/devops/what-is-observability#:~:text=Il%20termine%20osservabilit%C3%A0%20si%20riferisce,registri%20e%20metriche%20sulle%20prestazioni.

“Monitoring and Observability" - Di Cindy Sridharan (https://copyconstruct.medium.com/monitoring-and-observability-8417d1952e1c)

"Deep Learning" di Ian Goodfellow, Yoshua Bengio e Aaron Courville

“Natural Language Processing with PyTorch" di Delip Rao e Brian McMahan

"BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding" di Jacob Devlin et al.