UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Dipartimento di Scienze Fisiche, Informatiche e Matematiche

Corso di Laurea in

Informatica

Titolo Tesi

Progettazione e sviluppo di un'applicazione Big Data per l'analisi e l'elaborazione di tweet in real-time

RELATORE
Chiar.mo Professore
Riccardo Martoglia

CANDIDATO
Alessandro Pillo
MATR. 111759

Anno Accademico
2018/2019

Indice

Parte I

Studio delle tecnologie utili per l'analisi, l'elaborazione e l'interrogazione di Big Data

Parte II

Studio di un’applicazione reale

Parte III

Appendice

Elenco delle figure

Elenco delle tabelle

Elenco dei grafici

Introduzione

L'oggetto della trattazione affrontata nella seguente tesi può essere designato con la delicata problematica inerente al tema della gestione dei Big Data, argomento articolato, che presenta mille sfaccettature e che ancora si trova in una fase di continua evoluzione e sviluppo.

Si potrebbe con ragione sostenere, dunque, l'esistenza di un problema reale, che riguarda diversi ambiti e che sta iniziando ad essere preso in considerazione in un modo tutt'altro che trascurabile. A tale proposito, la complessità e la gestione di tale problema si presta ad essere analizzata sotto diversi punti di vista (analisi, interrogazione, sicurezza, privacy, etc.), ognuno dei quali presenta una propria complessità.

Anche se, nel lavoro qui seguentemente descritto, è stata posta particolare attenzione all'analisi, all'elaborazione e alla costruzione di opportuni modelli previsionali.

La struttura di questa trattazione è organizzata in un modello esplicativo costituito da cinque capitoli, il cui contenuto verrà di seguito illustrato. Si presti particolare attenzione a come il nesso logico perseguito sia stato, innanzitutto, finalizzato a offrire una panoramica riguardo i principali tool utili ai fini di tale trattazione, per poi applicare quanto analizzato mediante la progettazione di un'applicazione reale che sia in grado di elaborare stream di dati, in real-time, e su di essi costruire un modello previsionale che risulti essere in grado di evolvere nel tempo.

Parte I

Studio delle tecnologie utili per l'analisi, l'elaborazione e l'interrogazione di Big Data

Capitolo I

“Ecosistema Hadoop"

1.1 Big Data

Negli ultimi anni, la quantità di dati generati sta via via crescendo in maniera esponenziale e, sotto vari aspetti, sta generando sempre più complicazioni: riguardo al volume (ad esempio, ogni minuto vengono pubblicati oltre 500.000 tweet), riguardo alla varietà (dati strutturati, non strutturati o semistrutturati) e riguardo alla velocità di gestione (necessità di raccolta, elaborazione, archiviazione ed analisi in tempi estremamente rapidi). Tutto ciò è più opportunamente riassumibile con il termine “Big Data”, e ha favorito, in maniera incalzante, la necessità di avere nuovi sistemi e tecnologie per poter gestire, analizzare ed interrogare tale moltitudine di dati in maniera più efficiente. Infatti, è ormai constatabile che i classici sistemi utilizzati per la gestione dei dati, quali per esempio il tradizionale “Relational Database”, non sono propriamente adatti a gestire dati non strutturati e record di grandi dimensioni; inoltre presentano una natura statica degli schemi, limitando di conseguenza la flessibilità del sistema stesso.

Dato ciò, per affrontare la sempre più crescente mole di dati semistrutturati e non strutturati (es.: social media, foto, audio, video, etc.) sono nate apposite tecnologie e framework di gestione. Tra tutte le piattaforme e gli approcci per archiviare e analizzare i Big Data si ricorda innanzitutto Hadoop, il quale nacque come implementazione del modello di programmazione del MapReduce [1] e con la finalità di essere un framework scalabile, flessibile e affidabile nella gestione di volumi elevati di dati potenzialmente distribuiti tra cluster paralleli di server. Con l’avvento di tale progetto innovativo, numerosi altri framework, linguaggi e librerie si sono sviluppati attorno ad esso - e in parte anche sulla base di esso - fornendo servizi complementari e talvolta sostitutivi. Per tali ragioni è più opportuno parlare del cosiddetto “Ecosistema Hadoop”, la cui architettura può essere descritta dalla figura 1.1.

---

[1] *Gestisce dati basandosi su un algoritmo parallelo e distribuito che agisce su cluster di server.*

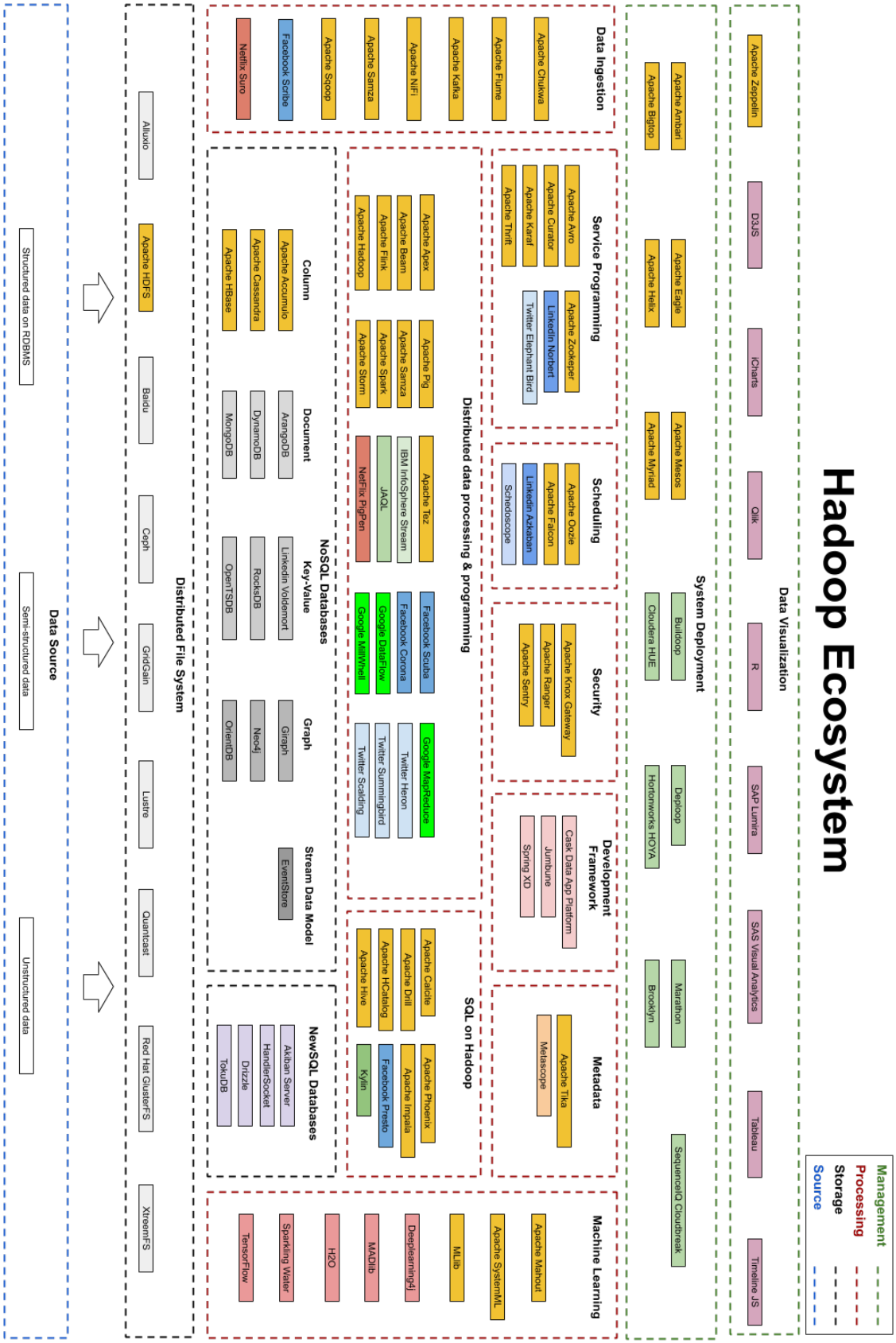

Figura 1.1 - Ecosistema Hadoop.

1.2 Analisi derivate dalla figura

La figura mostra solamente alcuni dei principali strumenti presenti. Infatti, esaminando i tool open source sviluppati dall'Apache Foundation e quelli presenti su GitHub oltre a quelli proprietari si supera la quota di 250 diverse soluzioni.

E' osservabile come siano presenti numerose soluzioni, alcune delle quali supportate dalle principali aziende che operano su Internet quali Google, Twitter, LinkedIn, Netflix etc., in quanto le prime ad aver avuto tali necessità. Inoltre, esse continuano ad avere il bisogno di gestire sempre più efficientemente i propri dati e quelli prodotti dai propri utenti. Data la moltitudine di alternative per processare, memorizzare e analizzare i Big Data e, in aggiunta, l'assenza di uno stack standard di tool, al fine di ottenere il massimo beneficio da questa architettura, una generica azienda, per poter scegliere il giusto set di tool, deve considerare, come suggerito da Boris Lublinsky [(2)] nel suo libro "Professional Hadoop Solutions", i seguenti parametri:

- le caratteristiche tecniche, quali la versione di Hadoop, i componenti disponibili, le funzionalità e le feature offerte (scalabilità, elaborazione parallela, performance, connettività con le applicazioni esistenti);
- il livello di praticità dei tool (facilità di installazione, configurazione, interfaccia di gestione intuitiva, possibilità di aggiornamenti);
- le esigenze di manutenzione e il supporto al ripristino in caso di fallimenti;
- il budget a disposizione: bisogna, infatti, considerare gli investimenti relativi all'installazione, alla manutenzione, agli aggiornamenti futuri e alle licenze;
- il supporto offerto per semplificare l'integrazione della soluzione con l'infrastruttura esistente.

A tutto ciò si aggiunge la necessità di considerare l'applicazione che utilizzerà tale piattaforma (batch, streaming, interactive) e le caratteristiche dei dati a disposizione.

L'analisi sopra descritta in modo sommario è di cruciale importanza per l'organizzazione stessa; infatti una scelta di strumenti non propriamente adeguati può avere ripercussioni economiche sull'azienda stessa.

[(2)] *Esperto e autore di numerosi articoli inerenti il panorama dei Big Data.*

1.3 Processing layer

Come descritto nel paper *"Big-Data Analytic Architecture for Businesses: a comprehensive review on new open-source big-data tools"* realizzato dall'Università di Cambridge, il livello che assume particolare rilievo in questa architettura è il cosiddetto "processing layer".

Questo è tale dal momento che, riuscire ad estrarre informazioni altamente rilevanti da un vasto insieme di dati, è di fondamentale importanza al fine di ottenere un processo decisionale valido su cui poter basare la propria applicazione. Per tali motivi, la gran parte dei tool sviluppati, e che saranno analizzati, sono all'interno della sottocategoria "distributed data processing & programming" nonché il core di questo livello (figura 1.1).

1.4 Distributed data processing & programming

Feature / Sistemi	Apache Apex	Apache Beam	Apache Hadoop (MapReduce)	Apache Flink	Apache Samza	Apache Spark	Apache Storm	Apache Tez	Google Cloud DataFlow	Google MillWheel	IBM InfoSphere Streams	Twitter Heron
Costi in memoria (RAM)	Elevato			Elevato	Elevato	Elevato	Elevato					
Data processing	Batch + native stream	Batch + stream	Batch	Batch + native stream	Batch + stream	Batch + stream (micro-batching/ fast batching)	Native stream	Batch	Batch + stream	Native stream	Native stream	Native stream
Descrizione	Piattaforma nativa Hadoop YARN che unifica stream e batch processing	Implementazione di un modello unificato per elaborare dati batch e streaming	Framework per l'elaborazione distribuita di dati tra cluster di computer usando il modello di programmazione del MapReduce	Framework ed engine di elaborazione distribuito per calcoli stateful sui flussi di dati illimitati e limitati. Funziona correttamente in ambienti cluster con prestazioni di calcolo elevate e su qualsiasi scala	E' un engine per elaborare e analizzare dati in tempo reale	E' un engine veloce e generale	E' un sistema di calcolo distribuito in tempo reale e in grado di elaborare grandi volumi di dati ad elevate velocità	Libreria per creare engine basati su flussi di dati	E' un servizio cloud che gestisce in modo completo la creazione e l'esecuzione di pipeline di dati in modalità batch e streaming	Framework per creare applicazioni in grado di elaborare stream di dati con bassa latenza	Piattaforma software per lo sviluppo e l'esecuzione di applicazioni che elaborano flussi di dati	Evoluzione di Apache Storm. Presenta performance migliori nel gestire milioni di tuple al minuto
Latenza	Molto bassa		Alta (secondi)	Molto bassa	Bassa	Medio/Bassa (pochi secondi)	Bassa (millisecondi)		Medio/Bassa (pochi secondi)	Bassa	Molto bassa	Molto bassa
Linguaggi Supportati	Java, Scala	Java, Python, Go, Scala	Java C C++, Ruby, Groovy, Perl, Python	Java, Scala, Python, R	Java, Scala	Java, Scala, Python, R	Multi-linguaggio		Java, Python (in sviluppo)	Principali linguaggi	Java, Python, Scala, C++, SPL	Java, Python, C++
Scalabilità	Alta e in modo dinamico	Alta	Media	Alta	Alta	Alta	Alta e in modo dinamico	Alta	Alta e in modo automatico (in orizzontale)	Alta	Medio/Alta	Alta e in modo automatico (in orizzontale)
Sicurezza	Kerberos, RBAC, LDAP		Kerberos	Kerberos, User-authentication	Apache Hadoop YARN il quale utilizzata Kerberos come meccanismo di autenticazione e autorizzazione	Kerberos, HDFS ACLS, file level permission	Kerberos	Kerberos + enctryption per i dati letti dalla rete	Sono presenti meccanismi di sicurezza e autenticazione		Sono presenti meccanismi di sicurezza al fine di determinare i permessi di accesso	
Supported streaming query	Sì	Beam SQL : Apache Calcite (subset)+ estensione affinché sia possibile gestire il modello unificato batch & stream. Supporta Join, Windowing & Triggering		API Table e SQL (basate su Apache Calcite). Le query presentano la stessa semantica sia per dati batch che per dati stream.	Samza SQL API (si basa su Apache Calcite estendendolo per il supporto allo stream). Versione non completamente stabile	SparkSQL (API DataSet e SQL)	Storm SQL				BigSQL (hybrid SQL engine per la sottomissione di query)	In fase di sviluppo
SQL	Funzionalità di base grazie ad Apex Malhar basato su Apache Calcite		Subset grazie ad Apache Hive (basato su Apache Calcite) ma non SQL-92									
Supporti di memorizzazione	RDBMS, NoSQL database e HDFS	File system e NoSQL database	HDFS	HDFS, RDBMS e NoSQL database	RDBMS, HDFS e NoSql database	HDFS e NoSql database	RDBMS, NoSql database e HDFS		RDBMS, NoSql database e HDFS		RDBMS, NoSql database e HDFS	RDBMS e HDFS
Tecniche che permettono di tollerare eventuali guasti	Checkpointing (recupero automatico senza dover ricominciare la computazione dall'inizio)		Built-in failure tolerance grazie ai HDFS	State (tramite distributed snapshot) & Checkpointing	Migration dei task in caso di fallimenti. Checkpointing incrementali	Recupero del lavoro perso	Re-start del "job" su un nodo diverso	Re-execution del task fallito	Re-execution del task fallito. Utilizza un meccanismo di ack	Single checkpoint per record multipli basati su un meccanismo di ack	Utilizzo di snapshot (algoritmo di Chandy-Lamport)	
Tipologia dati	Dati relazionali, avro, csv, xml, json, etc.	Dati relazionali, avro, parquet, xml, txt, etc.	Qualunque formato oltre a quelli ottimizzati (avro, orc, parquet)	Dati relazionali, avro, parquet, gz, xz, etc.	Dati relazionali, avro, parquet, orc, csv, json, etc.	Dati strutturati, orc, parquet, avro, json, etc.	Dati relazionali, orc, avro, parquet, etc.		Dati relazionali, xml, json, gcs, etc.		Dati relazionali, unicode text e binario	Dati relazionali, dati non relazionali, etc.
Throughput	Alto		No	Molto alto	Alto	Alto	Medio/Alto		Medio/Alto		Alto	Molto alto
Windowing event-time	Sì	Sì		Sì	Sì	Sì	Sì		Sì		Sì	

Tabella 1.1 - Confronto sistemi.

Le feature analizzate in tabella 1.1 sono:

- costi in memoria (RAM): la tecnologia alla base di sistemi real-time è denominata "in-memory" e permette di incrementare in maniera significativa le performance utilizzando le informazioni caricate in memoria centrale riducendo, o addirittura eliminando, il cosiddetto "collo di bottiglia" introdotto da operazioni di I/O da disco. Infatti, sistemi basati sull'uso del disco non possono offrire tali performance a causa delle latenze che l'uso del disco stesso introduce. Di conseguenza, tool che effettuano analisi in real-time hanno un'elevata occupazione di memoria principale;

- data processing: con il termine "batch processing" si intende una procedura in cui i dati sono in un primo momento raccolti, inseriti ed elaborati e solo successivamente si producono risultati batch (es.: elaborare tutte le transazioni eseguite da una società finanziaria in una settimana). Mentre con il termine "stream processing" ci si riferisce ad una procedura in cui i dati sono processati in real-time al loro arrivo. Tuttavia, è opportuno ricordare che esistono due tipi di "stream processing"; uno noto come "native streaming" e l'altro come "micro-batching". Nel primo caso ogni record è processato appena arriva senza aspettare altri record, mentre nel secondo caso, anche noto come "fast batching", i record sono aggregati in batch e poi elaborati in un unico "mini batch" introducendo di conseguenza piccoli ritardi;

- latenza: velocità con cui un utente può ottenere una risposta dopo aver eseguito una richiesta. In alcuni settori è di fondamentale importanza scegliere sistemi che presentano latenze molto basse (es.: trading finanziario, social media, giochi online);

- scalabilità: tool che scalano in modo elevato sono in grado di gestire l'aumento del carico di lavoro mantenendo le proprie performance invariate, grazie solitamente ad una ridistribuzione del carico a run-time;

- sicurezza: meccanismi necessari per l'autenticazione di chi (utenti, server e applicazioni) vuole e può accedere ai dati presenti nell'ecosistema Hadoop. Inoltre il

sistema, interfacciandosi il più delle volte anche con Internet, è necessario che presenti un sistema di protezione dei dati sia interno che esterno. A tale proposito, si ricordano i seguenti protocolli: Kerberos, LDAP, RBAC;

- SQL: è possibile notare come quasi tutti i tool utilizzino in modo diretto o indiretto Apache Calcite [3] il quale include un parser SQL, un'API per costruire espressioni utilizzando l'algebra relazionale e un query optimizer;

- supporti di memorizzazione: grazie alla presenza di appositi framework di data ingestion, i dati provenienti dalle varie sorgenti (es.: RDBMS, social media, etc.) possono essere memorizzati tramite tecnologie diverse quali NoSql database, NewSql database e HDFS. Mentre i Nosql e i NewSql database presentano soluzioni diverse per dati diversi, il cosiddetto Hadoop Distributed File System può memorizzare qualunque tipo di dato in ingresso (.xml, .csv, .jpg, .png, dati relazionali, etc.). Tuttavia, uno dei pilastri su cui si basa Hadoop è proprio la memorizzazione efficiente e la presenza di un elevato grado di parallelismo e ciò non è sempre ottenibile memorizzando i dati nel loro formato originale (es.: .json, .xml, etc.);

- tecniche che permettono di tollerare eventuali guasti: soluzioni adottate dai vari tool per consentire ad un sistema di tollerare guasti software che si possono verificare per esempio quando, date più istanze di un'applicazione in esecuzione su macchine diverse, un server subisce un guasto;

- tipologia dati:

1. Dati strutturati: seguono uno specifico schema e tipicamente sono memorizzati in tabelle all'interno di un apposito database relazionale.

---

[3] *Framework per la gestione dinamica dei dati tramite una pianificazione e ottimizzazione delle query in SQL.*

2. Dati semi-strutturati: sono una forma di dati strutturati ma non conformi con la struttura formale dei database relazionali. Solitamente contengono tag o altri marker per separare gli elementi semantici che li caratterizzano. Sono anche conosciuti come dati senza schema o dati con struttura autodescritta. Con alcuni processi è possibile memorizzarli in database relazionali ma ciò potrebbe essere molto costoso computazionalmente.
   Esempi: .csv, .tsv, .xml, .json, etc.

3. Dati non strutturati: sono dati che non sono organizzati in modo predefinito e pertanto non sono adatti ad un database relazionale. Quindi, per i dati non strutturati, esistono piattaforme alternative per l'archiviazione e la gestione.
   Esempi: “plain text file” (log file), media (.png, .gif, .jpeg, .mp3), etc.

- throughput: misura della quantità di informazione che un sistema può processare in un certo intervallo di tempo;

- windowing event-time: concetto correlato al processing di stream di dati e alla nozione di tempo e come essa sia modellata e integrata dal sistema.

Infine si noti come la maggior parte dei tool descritti siano progetti open source gestiti dall’Apache Foundation, la quale ricordiamo essere un’organizzazione no-profit.

Proprio per questo motivo, tali strumenti rappresentano valide soluzioni, ma a volte “troppo” generiche per le aziende che operano su Internet.

Un esempio a dimostrazione di ciò è fornito dal framework “Twitter Heron” [(4)], evoluzione migliorata del framework Apache Storm, nato per gestire il carico di lavoro sempre maggiore della piattaforma Twitter.

[(4)] *Framework sviluppato da Twitter.*

1.5 Sistemi analizzati in tabella

1.5.1 Apache Hadoop

E' opportuno innanzitutto ricordare le caratteristiche del progetto originale di Hadoop, il quale nacque come framework open source per l'elaborazione distribuita di dati tra cluster di computer, utilizzando il modello di programmazione del MapReduce.
Hadoop (versione 1.x) è composto dalle seguenti due parti:

- un file system, denominato "Hadoop Distributed File System" (HDFS), utilizzato per memorizzare i dati. Esso costituisce un sistema fault-tolerant ed è distribuito tra un "Name Node" e più "Data Node". Con il primo termine ci si riferisce al nodo master, il quale è utilizzato per memorizzare metadati (quanti blocchi sono memorizzati nei Data Node, quale Data Node ha i dati, i timestamp, etc.) sui Data Node. Mentre, con il secondo termine ci si riferisce ai nodi slave utilizzati per memorizzare effettivamente i dati dell'applicazione;

- il MapReduce, un modello di programmazione parallela per l'elaborazione distribuita di dati. E' basato sui demoni denominati "Job Tracker" e "Task Tracker". Il primo offre un servizio di assegnamento di appositi "MapReduce Task" ai Task Tracker (le cui istanze sono in esecuzione sui nodi slave, differenti, del cluster) e in scenari di arresto assegna nuovamente le stesse attività ad altri Task Tracker. Infine il Job Tracker mantiene tutti gli stati (attivo, non riuscito, ripristinato, etc.) dei Task Tracker. Invece, ogni istanza del demone Task Tracker esegue le attività assegnate dal Job Tracker e invia lo stato di tali attività al Job Tracker stesso.

Questa versione di Hadoop presenta i seguenti limiti: è adatta solo al processing di dati batch, supporta al più 4000 nodi per cluster, il JobTracker è un "single point of failure", non supporta la possibilità di scalare in orizzontale ed è in grado di assegnare le risorse solo in modo statico.

Con la versione 2.x di Hadoop si ha in aggiunta, innanzitutto, il supporto per l'esecuzione di applicazioni non batch attraverso l'introduzione di YARN, un gestore di risorse cluster che elimina la dipendenza di Hadoop dal modello di programmazione del MapReduce. In particolare, YARN elimina i concetti di JobTracker e TaskTracker introducendo le seguenti tre componenti: il ResourceManager (5), il NodeManager (6) e l'ApplicationMaster (7). Inoltre, grazie all'introduzione del gestore YARN si hanno benefici anche in termini di scalabilità (anche in orizzontale).

Grazie alla capacità di supportare modelli di programmazione diversi dal MapReduce, Hadoop (>= 2.x) ha assunto il ruolo di piattaforma per una più ampia varietà di applicazioni (streaming, real-time, etc.). Ora, quindi, il MapReduce è solo uno dei tanti "processing engine" che possono eseguire applicazioni nell'ecosistema Hadoop.

Inoltre, la versione 2.x di Hadoop ha introdotto anche dei miglioramenti nell'HDFS offrendo la possibilità di configurare cluster con "Name Node" ridondanti, eliminando la possibilità che il singolo "Name Node" diventi un "single point of failure" all'interno di un cluster, e offrendo in questo modo anche una migliore ridistribuzione del carico di lavoro rispetto ad Hadoop 1.x. Infine, questa versione ha introdotto anche tecniche di protezione contro errori di utilizzo e il concetto di istantanee per il ripristino dei sistemi in presenza di errori.

Hadoop utilizza Kerberos come protocollo di base per l'autenticazione e la propagazione dell'identità per utenti e servizi. Mentre per quanto riguarda l'autorizzazione, memorizza il file dei permessi nel file system presente (HDFS).

SQL:

"Apache Hadoop" - inteso come implementazione del MapReduce - permette la formulazione di query grazie all'utilizzo del framework Apache Hive.

---

(5) *Schedulatore che alloca le risorse disponibili nel cluster.*

(6) *In esecuzione su ciascun nodo nel cluster. Esegue i compiti forniti dal ResourceManager ed è responsabile della gestione delle risorse disponibili su un singolo nodo.*

(7) *Esegue un lavoro specifico ed è responsabile della negoziazione delle risorse, etc.*

“Apache Hadoop” - inteso come ecosistema - presenta numerosi framework per la formulazione di query, alcuni dei quali sono stati sviluppati per supportare la formulazione di query anche su dati stream oltre che batch.

Formati specifici supportati dall’HDFS:

1. SequenceFile: fornisce una struttura dati persistente, rappresentando una soluzione efficiente dal punto di vista dell’occupazione di memoria centrale, per quanto riguarda la memorizzazione di un numero elevato di file di piccole dimensioni. Infatti, l’idea su cui si basa è di memorizzare il numero elevato di file di piccole dimensioni in un unico file più grande, caratterizzato da coppie “chiave (nome del file) - valore (contenuto del file)” binarie. Risulta essere un formato più compatto rispetto al file di testo, supporta la sua suddivisione quando i dati al suo interno sono compressi, usa una memorizzazione basata sulle righe ed è disponibile in tre diverse varianti a seconda del tipo di compressione (uncompressed, record compressed e block compressed). Grazie a questo formato si gestiscono in modo corretto situazioni in cui si hanno sovraccarichi di memoria dovuti appunto alla presenza di un numero elevato di piccoli file.

2. Apache Avro: offre un formato veloce, compatto e supporta schemi dinamici. Inoltre, presenta due codifiche: binaria e .json. La prima può essere utilizzata solo per la codifica dei dati mentre la seconda anche per la codifica dello schema che caratterizza i dati. Inoltre, utilizza una memorizzazione basata sulle righe, risulta essere un formato altamente suddivisibile e particolarmente adatto a carichi di lavoro “write-heavy”. Infine, questo formato permette di minimizzare la dimensione dei dati in ingresso e di massimizzarne l’efficienza.

3. Apache Parquet: fornisce un formato binario, risulta essere altamente efficiente per query su larga scala, in termini di I/O da disco e di utilizzo di memoria. Inoltre, utilizza una memorizzazione basata sulle colonne, risulta essere un formato suddivisibile e adatto a carichi di lavoro “read-heavy” (usato con Apache Impala e progettato per query che necessitano di bassa latenza e alta concorrenza).

4. Apache ORC: formato sviluppato con l'obiettivo di migliorare la velocità delle query e l'efficienza della memorizzazione, la quale è basata sulle colonne. Inoltre, risulta essere un formato suddivisibile e adatto a carichi di lavoro "read-heavy" grazie alla presenza di un opportuno sistema di indicizzazione.

Formato dati	**Performance**	**Velocità**	**Piattaforme supportate**
CSV	Migliori rispetto al formato JSON	Minori rispetto al formato Sequence file	MapReduce / Hive
Text	Minori rispetto al formato CSV	Minori rispetto al formato Sequence file	MapReduce / Hive
Sequence File	Migliori per dataset piccoli	Maggiori per dataset piccoli ma non per grandi dataset	MapReduce / Hive
JSON	Minori rispetto al formato CSV	Minori rispetto al formato CSV	MapReduce / Hive
Avro	Minori rispetto Parquet. Ottimo in caso di molte righe	Minori rispetto Parquet. Scala meno per molte colonne	MapReduce, HBase, Hive, Pig e Spark
Parquet	Migliore di Avro, ottimo in caso di più colonne ma non supporta l'indicizzazione	Migliore di Avro e scala meno per molte righe	MapReduce, HBase, Hive, Pig e Spark
ORC	Migliore di Avro, supporta l'indicizzazione ed è ottimo con Hive	Migliore di Avro e ottimo per la formulazione di query	MapReduce, Hive, Pig e Spark. Non supporta HBase

Tabella 1.2 - Caratteristiche formato dati.

La scelta del formato ha un impatto sulle performance e sullo spazio occupato. A tale proposito sono stati sviluppati appositi formati ottimizzati adatti all'uso di cluster Hadoop. Tuttavia, nel caso in cui non sia possibile decidere il formato di dati con cui poter lavorare, e i dati arrivino in formati quali per esempio .csv, .json, .xml, etc., è comunque possibile utilizzare appositi strumenti (es.: Nexla, piattaforma scalabile e in real-time di "Data Operation") per convertire questi formati in formati più adatti (avro, parquet, orc).

Invece, nel caso in cui sia possibile scegliere il formato dei dati è opportuno prendere in considerazione i seguenti aspetti:

- specifiche del sistema, ovvero i tool utilizzati. Infatti, non tutti i tool supportano i vari formati presenti. Per esempio, Apache Impala [9] non supporta il formato ORC e quindi sarà più opportuno utilizzare Parquet;

- risorse a disposizione. E' opportuno ricordare come alcuni formati comprimono molto più di altri, incrementando di conseguenza la velocità delle operazioni di I/O e riducendo lo spazio occupato sul disco. Tuttavia, ciò introduce un uso maggiore della CPU per la successiva decompressione dei dati;

- suddivisibilità: determina l'abilità di processare parti di file in maniera indipendente abilitando pienamente il parallelismo dell'architettura alla base del sistema in uso. Formati compressi dovrebbero essere sempre suddivisibili, mentre è noto come formati quali .xml e .json non sono suddivisibili.

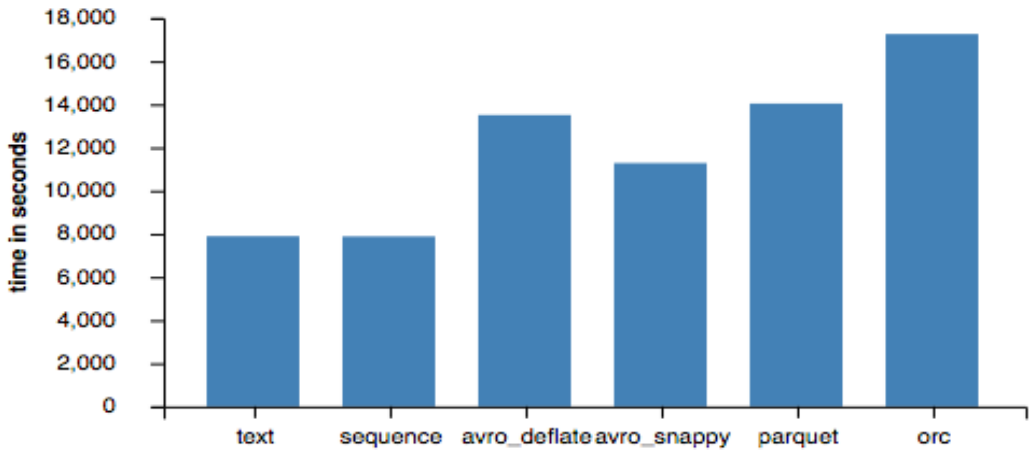

Grafico 1.1 - Confronto delle prestazioni in scrittura.

Il grafico 1.1 mostra quanto tempo impiega Apache Hive [10] per scrivere una nuova tabella di 1000 colonne, 302.924.000 righe e 1 TB di dati nei vari formati specificati.

---

[9] *Engine per l'esecuzione di query SQL in un cluster Hadoop.*

[10] *Framework per formulare query ed effettuare analisi.*

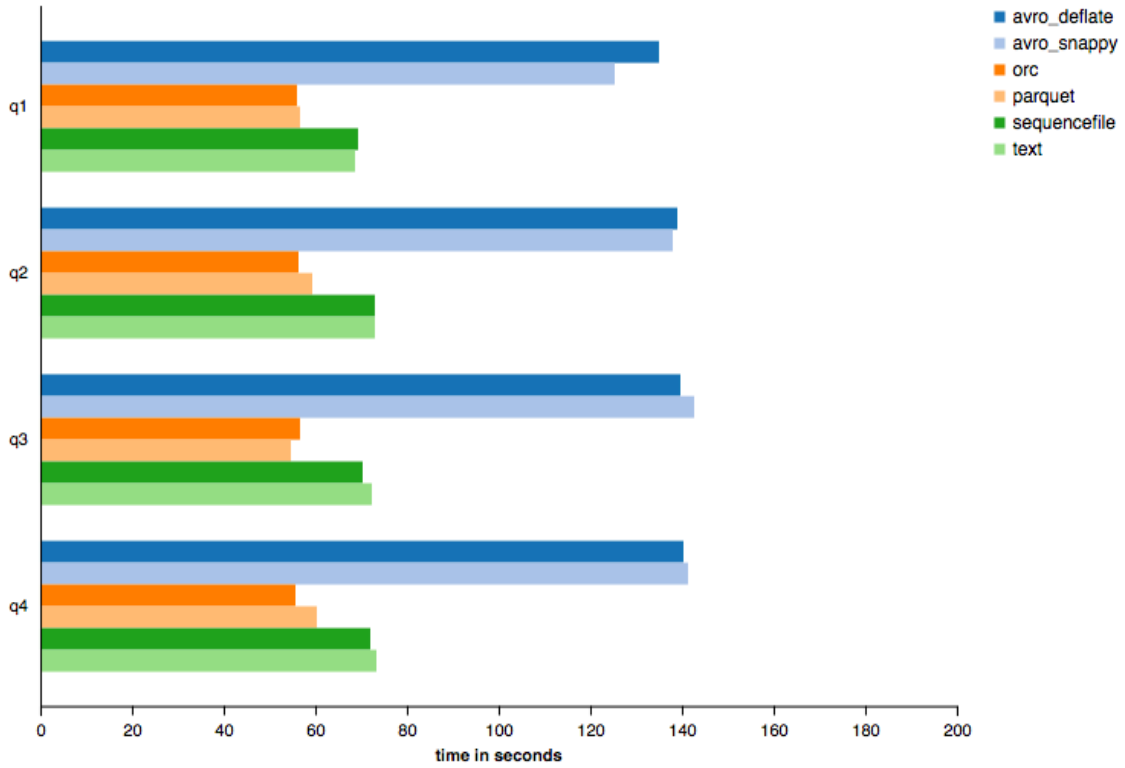

Grafico 1.2 - Confronto delle prestazioni per l'esecuzione delle query q_n.

Il grafico 1.2 mostra quanto tempo impiega Apache Hive per eseguire, nei vari formati specificati, le seguenti query :

Select count (*) from table where...

- q1: non presenta condizioni;
- q2: presenta 5 condizioni;
- q3: presenta 10 condizioni;
- q4: presenta 20 condizioni.

E' stata applicata la compressione snappy per la maggior parte dei formati, ad eccezione del formato Apache Avro per il quale è stato utilizzato anche il formato compresso deflate [11].

---

[11] *I dati mostrati sono il risultato di uno studio condotto dalla Silicon Valley Data Science, azienda di consulenza specializzata nei Big Data.*

1.5.2 Apache Apex

Apache Apex è una piattaforma che implementa la versione nativa del gestore "Hadoop YARN" e che unifica il processo di elaborazione di dati in stream e dati batch.

Pertanto rappresenta un engine per l'elaborazione di dati di grandi dimensioni e presenta le seguenti caratteristiche:

- altamente scalabile: in presenza di miliardi di eventi al secondo è in grado di scalare linearmente grazie ad un partizionamento, dinamico o talvolta definito staticamente, del carico di lavoro. Inoltre, aggiungendo ulteriori risorse è possibile processare una quantità di dati sempre maggiore senza alcun limite imposto dall'architettura stessa;

- altamente performante: grazie alla possibilità di elaborare anche dati presenti in memoria centrale. Così facendo è possibile raggiungere una latenza end-to-end di pochi millisecondi;

- tollerante ai guasti e mantenimento dello stato: innanzitutto, si utilizza un apposito buffer per garantire la replicazione dei dati dal punto di ripristino in modo che non vi siano perdite e si abbiano ripristini rapidi. In caso di guasti, anche lo stato degli operatori utilizzati, il quale è memorizzato di default nell'HDFS, viene ripristinato al valore assunto al momento del checkpoint. Inoltre, il rilevamento di guasti e il ripristino automatico da situazioni che presentano guasti non richiedono la necessità di un intervento manuale. Ciò è reso possibile grazie al processo di notifica offerto da YARN e all'uso del meccanismo di heartbeat. Quest'ultimo, ricordiamo essere un segnale inviato tra le macchine a intervalli di tempo regolari e nel caso in cui l'endpoint non riceva tale segnale per un certo tempo, si suppone che la macchina che avrebbe dovuto inviare il segnale abbia avuto un guasto;

- utilizzo della versione nativa di YARN per la negoziazione delle risorse;

- sicura: si utilizza il meccanismo di autenticazione presente in Hadoop e noto come Kerberos. Esso rappresenta un sistema di autenticazione “ticked based” - basato cioè sulla presenza di ticket validi per un intervallo di tempo limitato che possono essere utilizzati per stabilire connessioni - in un ambiente distribuito dove l’autenticazione è necessaria tra utenti multipli, host e servizi;

- developer friendly: presenta una API di alto livello che maschera i dettagli riguardanti la gestione dei vari problemi operativi (gestione dello stato, tolleranza agli errori, scalabilità, sicurezza, etc.) e permette lo sviluppo di operatori. La piattaforma principale Apex è completata dalla libreria open source Malhar, la quale è particolarmente utile per creare applicazioni di streaming real-time grazie ad un ampio insieme di operatori. Le due macro categorie di operatori sono note come “input/ output operator” (es.: operatori per interfacciarsi con file system, RDBMS, NoSQL, messaging, notification, protocol read/write, social media, in-memory database, etc.) e “compute operator” (es.: operatori per il pattern matching, machine learning, parser, query & scripting, etc.).

Directed Acyclic Graph (DAG):

Un’applicazione che elabora dati in streaming in Apex è rappresentata da un grafo diretto aciclico (DAG), il quale esprime la logica di elaborazione tramite vertici che rappresentano gli operatori utilizzati e archi che rappresentano gli stream di dati. Un operatore prende uno o più stream in input e ne esegue una computazione emettendo uno stream di output. Gli operatori possono essere appositamente realizzati o selezionati dalla libreria Malhar. Con il termine stream, invece, ci si riferisce a sequenze di dati anche chiamati eventi o tuple. Il grafo deve essere aciclico, nel senso che ogni tupla è processata solo da un operatore (garanzia di esecuzione: exactly-once). Un’eccezione a questo modello iterativo, comunque supportato da Apache Apex, presenta dei cicli e tale costrutto è richiesto per l’utilizzo di algoritmi di machine learning. Infine, dati batch possono essere elaboratori nello stesso modo in cui sono processati dati stream.

Meccanismi di windowing:

L'elaborazione di dati stream, solitamente "illimitati", richiede l'introduzione del concetto di "finestra" per poter stabilire un limite alla quantità di dati da processare e per semplificarne la gestione. Ricordiamo a tale proposito i seguenti due meccanismi:

- meccanismo di windowing introdotto dalla versione nativa dell'engine Apex: la finestra di streaming è un intervallo di elaborazione che può essere applicato in casi che non richiedono la gestione di input fuori ordine. Esso assume che il flusso di dati possa essere suddiviso in intervalli di tempo fissi (default 500 ms), in cui l'engine possa eseguire operazioni di aggregazione su più record arrivati in quell'intervallo. Tuttavia, questi record non supportano trasformazioni o altre elaborazioni basate sul tempo in quanto i dati potrebbero non arrivare nella sequenza corretta;

- meccanismo di windowing introdotto dalla libreria Malhar: introduce un concetto di "finestra" il quale risulta essere più flessibile e ampiamente applicabile, includendo la possibilità di elaborare eventi basati sul tempo e fuori ordine;

Esempi di casi di utilizzo:

- pubblicità online: in seguito alle ricerche online condotte da un utente, riguardanti l'acquisto di un prodotto di un particolare brand, il suo account Facebook mostrerà le migliori offerte riguardanti proprio il brand in questione;
- wireless sensor networks: soluzioni distribuite di sensori autonomi utilizzati per monitorare condizioni ambientali (temperatura, pressione, etc.) i cui dati necessitano di essere analizzati in real-time;
- real-time reporting (guadagni, click info, impressioni), real-time monitoring (avvisi su affari, parametri salute), real-time learning (raccomandazione di articoli correlati), etc.

Situazioni non propriamente adatte:

- machine learning: Apache Apex non offre alcun tipo di libreria che implementi algoritmi di machine learning. Può, tuttavia, essere utilizzato come "engine" per framework di machine learning (es.: Apache SAMOA);

- query sql interattive: nonostante supporti la possibilità di formulare query per dati in streaming non è confrontabile con framework quali Hive o simili;
- non supporta la versione nativa di Python, ma solamente la versione riscritta in Java nota come Jython.

SQL:

Apache Apex utilizza Apache Calcite per l'analisi, l'ottimizzazione e l'esecuzione di query anche su dati in streaming. Data una query in ingresso, Apache Calcite ritorna l'albero che ne rappresenta la struttura dal punto di vista relazionale, la quale poi viene convertita in un DAG grazie all'utilizzo di appositi operatori presenti nella libreria Apache Malhar.

Storage e formato:

Grazie alla libreria Malhar, Apache Apex supporta i principali sistemi di memorizzazione (es.: RDBMS, NoSQL database, HDFS, etc.), alcuni framework di "data ingestion" (es.: Apache Kafka, etc.) e alcuni dei principali formati di dati (es.: .csv, .xml, .json, file di log, avro).

1.5.3 Apache Beam

Apache Beam fu inizialmente sviluppato da Google che unì più progetti (MapReduce, MillWheel, Flume) per fornire un modello di programmazione unificato per la gestione di dati batch e stream. Successivamente Google decise di donare questo progetto all'Apache Foundation. Quindi, l'utilizzo di Apache Beam permette di sviluppare codice indipendente dall'ambiente di esecuzione (Google Cloud Dataflow, Apache Apex, Apache Spark, etc.). Grazie a ciò risulta essere molto portabile. Infine, è possibile estenderne le funzionalità di base grazie all'utilizzo di apposite "transformation libraries" e "IO connector".

Apache Beam pipeline:

Con il termine pipeline si intende l'intera attività di elaborazione dati dall'inizio alla fine. Ciò include la lettura dei dati in input, la trasformazione di tali dati e la scrittura dei dati di output. In particolare, la pipeline opera su un insieme di dati distribuiti limitati o illimitati noti come PCollection. Si hanno PCollection limitati nei casi in cui le sorgenti siano per esempio database o file, mentre si hanno PCollection illimitati nei casi in cui si effettuino letture di dati in streaming tramite appositi framework di "data ingestion" (es.: Apache Kafka).

La natura limitata o illimitata dei PCollection influisce sul modo in cui Apache Beam processa i dati. Infatti, nel primo caso si utilizza un "batch job" che legge l'intero insieme di dati e ne esegue un'elaborazione. Nel secondo caso, invece, si esegue uno "streaming job" la cui esecuzione avviene in modo continuo poiché l'intera collezione non sarà mai disponibile.

Infine, i PCollection sono soggetti ad operazioni di elaborazione note come PTransform. Per progettare correttamente una pipeline è opportuno considerare i seguenti aspetti:

- dove i dati sono memorizzati e quanti insiemi differenti di dati si hanno;
- il formato in input e in output. Per esempio: plain text, file di log, tuple di un database, etc.;
- quali tipi di operazioni saranno eseguite sui dati, al fine di applicare eventuali manipolazioni sui dati stessi.

Pipeline complesse possono essere rappresentate tramite appositi DAG che rappresentano in modo schematico la presenza di sorgenti multiple di dati, le operazioni di trasformazione e i dispositivi di memorizzazione in uscita.

Meccanismi di windowing:

In Apache Beam qualsiasi PCollection può essere suddivisa in finestre logiche caratterizzate da timestamp. Ciò assume particolare importanza per la gestione dei PCollection illimitati in finestre logiche di dimensioni finite. Ogni elemento in un PCollection ha un timestamp intrinseco assegnato inizialmente dalla sorgente che crea il

PCollection. Le sorgenti che creano una raccolta di dati illimitata, spesso assegnano a ogni nuovo elemento un timestamp che corrisponde a quando l'elemento è stato letto o aggiunto. Di contro, le sorgenti che creano una raccolta di dati limitata assegnano automaticamente data e ora (comportamento più comune: si assegna ad ogni elemento lo stesso timestamp). Inoltre, per dividere gli elementi di un PCollection, esistono diversi tipi di "finestre" quali:

- "Fixed Time Window": dato un PCollection illimitato e quindi con data e ora aggiornati continuamente, una tale finestra considera tutti gli elementi appartenenti ad uno stesso intervallo temporale fissato. Esempio: finestra con un intervallo pari a 5 minuti. Tutti gli elementi con timestamp da 0:00:00 a 0:05:00 (escluso) appartengono alla prima finestra, gli elementi con timestamp da 0:05:00 a 0:10:00 appartengono alla seconda finestra e via dicendo;

- "Sliding Time Window": rappresenta una finestra scorrevole, la quale è caratterizzata da una durata (es.: cinque minuti) e da un periodo che indica ogni quanto una nuova finestra viene generata. Queste finestre possono anche sovrapporsi. Esempio: calcolare il valor medio dei dati degli ultimi cinque minuti con un aggiornamento di tale valore ogni dieci secondi;

- "Session Window": definisce una finestra che contiene elementi che arrivano in modo asincrono ma che sono all'interno di un certo gap temporale l'uno dall'altro;

- "Single Global Window": i dati, di default, sono assegnati a questa finestra. Nel caso di insiemi di dati a dimensioni fisse si può utilizzare questa finestra.

Infine, esiste anche la finestra denominata "Calendar-based Window" ed eventualmente, in caso di necessità più complesse, è possibile definire anche una propria finestra.

Un concetto correlato, chiamato trigger, determina quando emettere i risultati ottenuti da dati non limitati ma che sono aggregati tra loro. Pertanto è possibile utilizzare i trigger per perfezionare la strategia di windowing.

Casi di utilizzo:

Apache Beam è particolarmente utile per l'elaborazione nota come "embarrassingly parallel", in cui il problema può essere scomposto in molti e più piccoli problemi che possono essere elaborati in maniera indipendente e in parallelo. Inoltre, può essere usato anche in processi di estrazione, trasformazione e caricamento di task e di dati (ETL).

Trova, infine, un'applicazione nel contesto dell'analisi (computazioni batch e stream) e dell'orchestrazione (composizione, simulazione, etc.) dei task.

Beam SQL:

Permette di formulare query su PCollection limitati e illimitati (attualmente disponibile solo in Java). La query viene tradotta in una PTransform, la quale può essere combinata con altre PTransform caratterizzanti la pipeline. Per una corretta formulazione di query nella pipeline è opportuno ricordare i seguenti tre aspetti:

- il Beam SQL si basa sulle funzionalità offerte da Apache Calcite e su un'apposita estensione che rende più semplice l'interazione con il modello unificato per l'elaborazione batch e streaming;

- SqlTransform: interfaccia utilizzata per trasformare query formulate in SQL in apposite operazioni di elaborazione (PTransform);

- righe: il tipo (PCollection <Row>) di elemento su cui il Beam SQL opera. Un elemento in formato Row rappresenta un singolo e immutabile record in un PCollection.

Storage e formato:

Apache Beam supporta i principali sistemi di memorizzazione (file system e NoSql database), alcuni framework di "data ingestion" (es.: Apache Kafka, etc.) e alcuni dei principali formati di dati (es.: .xml, avro, parquet e .txt).

In aggiunta, il supporto effettivo verso questi sistemi dipende anche dal linguaggio di programmazione utilizzato.

1.5.4 Apache Flink

Apache Flink è un framework open source che implementa un engine di elaborazione distribuito per computazioni stateful su flussi di dati limitati e illimitati ed è completamente indipendente da Apache Hadoop. Tuttavia, si integra molto bene con alcuni dei componenti che caratterizzano Hadoop, quali per esempio l'HDFS (usato per la lettura dei dati o la scrittura di snapshot/checkpoint) e YARN (dal quale sfrutta il modulo Kerberos). Inoltre, visto la possibilità di gestire applicazioni di streaming che funzionano ininterrottamente, presenta eccellenti meccanismi e strumenti per il monitoraggio e ripristino da situazioni di errore grazie ad appositi checkpoint e sistemi che garantiscono l'esecuzione denominata "exactly-once". Infine è opportuno ricordare anche il concetto di "savepoint" [12], il quale permette di effettuare migrazioni di applicazioni su cluster diversi, di aumentare o diminuire il parallelismo e di fermare temporaneamente l'esecuzione di un'applicazione.

Un'applicazione basata su questo framework può utilizzare quantità "illimitate" di cpu, ram e disco per la propria esecuzione. Apache Flink, infatti, è in grado di gestire miliardi di eventi al giorno (alto throughput), terabyte di informazioni sugli stati (in memoria o sul disco) e l'esecuzione su migliaia di core, il tutto con tempi di latenza molto bassi.

---

[12] *Snapshot consistente dello stato di un'applicazione. A differenza di un checkpoint deve essere attivato manualmente e non viene rimosso automaticamente all'arresto di un'applicazione.*

Concetti importanti:

- stream: aspetto fondamentale alla base di applicazioni di “stream processing”. Gli stream possono essere limitati (insiemi di dimensioni fissate) o illimitati. Inoltre, tali stream possono essere elaborati in real-time (direttamente al momento della generazione) o essere memorizzati su un sistema di archiviazione ed essere elaborati successivamente;

- stato: ogni applicazione di streaming non banale memorizza risultati intermedi o eventi per accedervi in un momento successivo. A tal riguardo, Flink offre un vasto insieme di funzionalità per una corretta e completa gestione dello stato (es.: esecuzione exactly-once, “multiple state primitive”, etc.);

- tempo: è un aspetto importante delle applicazioni di streaming. Flink offre un ampio insieme di feature a tale proposito tra cui: “event-time mode” (13), “watermark support” (14), “late data handling” (15) e “processing-time mode” (16). Il tipo di applicazioni che possono essere costruite dipende da questi concetti temporali.

API:

Flink fornisce tre differenti livelli di API, ognuno dei quali è rivolto a differenti casi di utilizzo e presenta un diverso livello di astrazione. E’ possibile descrivere questa suddivisione nel seguente modo:

- High-level Analytics API: presenta due API relazionali, API Table (per la composizione di query in Java o Scala. Segue il modello relazionale e quindi le tabelle

---

(13) *Le applicazioni calcolano i risultati in base ai timestamp degli eventi, così facendo si hanno risultati accurati e coerenti indipendentemente dal fatto che vengano elaborati eventi registrati o in tempo reale.*

(14) *Flink offre un meccanismo flessibile per la gestione a grana fine del tempo, permettendo di bilanciare la latenza e la completezza dei risultati.*

(15) *Meccanismo per gestire eventi in ritardo e aggiornare i risultati completati in precedenza.*

(16) *Modalità in cui Flink esegue la computazione in base al tempo di clock della macchina. Ciò può essere utile per applicazioni a bassa latenza e che possono tollerare risultati approssimativi*

presentano uno schema) e SQL, entrambe API unificate per l'elaborazione di dati batch e stream. Ciò significa che le query vengono espresse con la stessa semantica sia su flussi illimitati in tempo reale che su flussi registrati e limitati. Entrambe sfruttano Apache Calcite per l'analisi, la convalida e l'ottimizzazione delle query e possono essere perfettamente integrate con le API di DataStream e DataSet che supportano funzioni scalari e aggregate.

Operatore / Dati	**Batch (Table / SQL)**		**Streaming (Table / SQL)**	
Scan (From)				
Select				
As				
Where/Filter				
GroupBy			*update dei risultati*	*update dei risultati*
GroupBy Window				
Over				
Distinct			*update dei risultati*	*update dei risultati*
Having				
Inner Join				
Outer Join			*update dei risultati*	*update dei risultati*
Union				
Union All				
Intersect				
Minus (Except)				
Order By				
Offset & Fetch				
Insert into				

Tabella 1.3 - Principali operazioni supportate dalle API: Table e SQL.

Si noti come, una query su uno stream di dati viene eseguita in continuazione, aggiornando (quando possibile) i risultati in base ai nuovi record ricevuti. E' importante osservare come i risultati ottenuti da query eseguite in continuazione siano equivalenti ai risultati ottenuti dalla stessa query eseguita su dati batch ottenuti da uno snapshot della tabella dei dati stream;

- Stream & Batch Data Processing: presenta l'API DataStream e l'API DataSet. La prima fornisce primitive per operazioni comuni di elaborazione dello stream come il windowing, le trasformazioni record-at-time e l'arricchimento degli eventi mediante la formulazione di query su un sistema di memorizzazione di dati esterno. Mentre la seconda permette di eseguire operazioni di trasformazione su dati batch;

Trasformazione / API	**Descrizione**	**DataStream**	**DataSet**
Map	Dato un elemento in ingresso produce esattamente un elemento in uscita		
FlatMap	Dato un elemento in ingresso produce zero, uno o più elementi in uscita		
Filter	Per ogni elemento esegue una funzione booleana. Ritorna solo gli elementi per cui si ha True come valore di ritorno		
KeyBy	Partiziona un flusso in partizioni disgiunte. Tutti i record con la stessa chiave sono assegnati alla stessa partizione		
Reduce	Combina due elementi ed emette un singolo valore	Rolling reduce: riduce l'ultimo valore reduced con quello nuovo, sul valore della chiave	Supporta anche gruppi di elementi
ReduceGroup	Combina un gruppo di elementi ed emette uno o più elementi		
Aggregations / Aggregate	Esempi: min, max, sum		

Distinct	Rimuove i duplicati		
Join	Unisce due elementi in base ad un campo in comune		
OuterJoin	Può eseguire un left, right o full outer join		
Union	Unisce due data stream/set	Anche più di due	
Window	Supporta trasformazioni di aggregazione, join, reduce, etc.		
Select		Seleziona uno o più stream da uno "split stream"	Seleziona un sottoinsieme di campi dalle tuple

Tabella 1.4 - Principali trasformazioni supportate dalle API: DataStream e DataSet.

- Stateful Event-Driven Applications: presenta funzioni ("ProcessFunction") di basso livello che forniscono un controllo a grana fine sul tempo, sullo stato e sugli elementi dello stream.

Infine è presente un insieme di librerie, ognuna delle quali è integrata in una specifica API (es.: "Complex Event Processing", integrata nella DataStream API, per il rilevamento di pattern su flussi di eventi).

Meccanismi di windowing:

Il concetto di finestra è al centro dell'elaborazione di stream illimitati. Essa permette di suddividere il flusso in "bucket" di dimensioni finite su cui è possibile applicare delle computazioni (window function). La prima cosa da specificare è se lo stream sia "keyed" o meno e ciò deve essere fatto prima di definire la finestra. Usando il metodo keyBy (...) si divide il flusso infinito in flussi logici con chiave. Avere un flusso con chiave (può essere un qualsiasi attributo) consente di eseguire il calcolo delle window in parallelo tramite più task, poiché ogni flusso logico con chiave può essere elaborato

indipendentemente dal resto. Quindi, tutti gli elementi che fanno riferimento alla stessa chiave verranno inviati allo stesso task parallelo. In caso di flussi senza chiavi, il flusso originale non verrà suddiviso in più flussi logici e tutta la logica di windowing verrà eseguita da un singolo task, ovvero con parallelismo di 1.

Sono presenti le seguenti finestre:

- Tumbling window: presenta una dimensione fissa e le finestre non si sovrappongono. Esempio: data una finestra tumbling con una dimensione di 5 minuti, ogni 5 minuti, la finestra corrente verrà valutata e successivamente verrà avviata una nuova finestra;

- Sliding window: presenta una finestra di lunghezza fissa e una frequenza con cui vengono avviate nuove finestre scorrevoli. Di conseguenza le finestre scorrevoli possono essere sovrapposte. Esempio: finestre di dimensioni 10 minuti che scorrono ogni 5 minuti. Dato ciò si ottiene, ogni 5 minuti, una finestra che contiene gli eventi che sono arrivati negli ultimi 10 minuti;

- Session window: raggruppa gli elementi in base alle sessioni di attività. Queste finestre non si sovrappongono e non hanno un orario di inizio e di fine fisso ma si chiudono quando non si ricevono elementi per un certo periodo di tempo (session gap dinamico o statico);

- Global window: assegna tutti gli elementi con la medesima chiave alla stessa finestra globale. Questo tipo di finestra è utile solo se si specifica anche un trigger personalizzato, poiché in assenza di tale specificazione non verrà eseguito alcun calcolo, dal momento che la finestra globale non ha una fine naturale durante la quale elaborare gli elementi aggregati.

Casi di utilizzo:

- Applicazioni "event-driven" [17]: rilevamento di frodi/anomalie, social network, monitoraggio di processi, etc.
- Applicazioni "data analytic" [18]: monitoraggio della qualità di reti (es.: Telco), analisi ad-hoc di dati in tempo reale, analisi su grafi di larga-scala, etc.;
- Applicazioni "data pipeline" [19]: creazione e perfezionamento in modo incrementale di un indice di ricerca in applicazioni real-time di e-commerce, etc.;
- Machine learning, continuo ETL, etc.

Storage e formato:

Apache Flink non è accoppiato con nessun dispositivo di archiviazione o formato specifico. Può interagire con i vari file system esistenti (es.: HDFS e i formati che esso supporta), framework di "data ingestion" (es.: Apache Kafka) e database (relazionali, NoSQL). Supporta, infine, anche la decompressione (ma non la lettura in parallelo), di file in formati quali: .deflate, gz, .gzip, .bz2, .xz.

1.5.5 Apache Samza

Apache Samza è un engine di elaborazione dati che consente di analizzare dati in real-time provenienti da fonti diverse. Esso offre le seguenti funzionalità per semplificare la creazione di applicazioni:

- integrabilità ad ogni livello: è in grado di elaborare e trasformare dati provenienti da qualsiasi fonte (es.: Apache Kafka, AWS Kinesis, Azure EventHubs, Hadoop, etc.);

- tolleranza ai guasti: nel caso in cui si verifichino guasti migra in modo trasparente le attività insieme al loro stato. Inoltre, supporta l'host-affinity (tenta il riavvio prima

---

[17] *Applicazioni stateful che acquisiscono eventi da stream in input reagendo opportunamente.*

[18] *Applicazioni utili per estrarre informazioni da dati grezzi.*

[19] *Applicazioni per trasformare, arricchire e spostare dati tra sistemi di archiviazione.*

sulla stessa macchina per sfruttare il valore dello stato locale), il checkpoint incrementale per consentire il recupero veloce dai guasti ed offre la garanzia di esecuzione “at-least once” (garantisce che nessun dato sia perso anche in caso di errori);

- scalabilità elevata in orizzontale: applicazioni che utilizzano diversi terabyte di stato ed eseguono su migliaia di core utilizzano Apache Samza. (es.: LinkedIn, Uber, TripAdvisor, Slack etc.);

- latenze estremamente basse e alto throughput per analizzare i dati istantaneamente;

- sicurezza: è possibile eseguire i “job Samza” su un cluster YARN sicuro, che utilizza Kerberos come meccanismo di autenticazione e autorizzazione.

Concetti importanti:

- stream: Apache Samza elabora i dati sotto forma di stream. Con il termine stream si intende una raccolta di messaggi caratterizzati da una coppia chiave-valore, immutabili e solitamente dello stesso tipo. Uno stream può avere più produttori che scrivono e più utenti che leggono. I dati di uno stream possono essere illimitati (es.: Apache Kafka) o limitati (es.: un insieme di file su HDFS). Uno stream viene suddiviso in più partizioni per scalare il modo in cui i dati vengono elaborati. Ogni partizione è una sequenza di record ordinata e riproducibile, in cui ogni messaggio è identificato in modo univoco da un offset;

- stato: Apache Samza supporta l'elaborazione di stream sia stateless che stateful. L'elaborazione stateless, come suggerisce il nome, non mantiene alcun stato associato al messaggio corrente dopo che è stato elaborato. Al contrario, l'elaborazione con stato richiede di registrare qualche stato su un messaggio anche dopo averlo elaborato (es.: contare il numero di utenti unici in un sito Web ogni cinque minuti. Ciò richiede di memorizzare le informazioni su ciascun utente visto fino ad ora per la

deduplicazione). In aggiunta, Samza offre la memorizzazione dello stato anche per motivi di scalabilità e tolleranza d'errore;

- il tempo è un concetto fondamentale nell'elaborazione di stream, in particolare assume importanza il modo in cui è modellato e interpretato dal sistema. Apache Samza supporta due nozioni di tempo: tempo di elaborazione (default) e tempo dell'evento. Nel primo caso il timestamp di un messaggio è determinato dal momento in cui viene effettivamente elaborato dal sistema (es.: un evento generato da un sensore potrebbe essere elaborato da Samza diversi millisecondi dopo). Di contro, nel secondo caso il timestamp di un evento è determinato a partire da quando si è effettivamente verificato. Samza fornisce l'elaborazione basata su eventi grazie alla sua integrazione con Apache Beam;

API:

- Java API: sono presenti due API per sviluppare applicazioni di elaborazione di stream. La prima è di alto livello e consente di descrivere la pipeline di elaborazione del flusso sotto forma di un DAG (Directional Acyclic Graph) delle operazioni sui flussi di messaggi. Inoltre, fornisce un ricco insieme di operatori che semplificano operazioni quali il filtraggio, la proiezione, il join, la gestione delle finestre, etc. Mentre, la seconda API è di basso livello e consente di scrivere l'applicazione in termini di logica di elaborazione per ciascun messaggio in arrivo;

- Samza SQL: fornisce un linguaggio di query dichiarativo per descrivere la logica di elaborazione dello stream utilizzando i predicati SQL. Tale API utilizza internamente Apache Calcite, che fornisce il supporto per l'analisi e la pianificazione delle query;

- Apache Beam API: è considerata un'estensione per interagire con applicazioni scritte utilizzando l'API di Apache Beam.

Meccanismi di windowing:

Il concetto di finestra serve per raggruppare i messaggi in arrivo in un insieme finito.

Ogni finestra, a seconda della modalità di accumulo di dati prende il nome di:

- "finestra di scartamento": dopo l'emissione dei risultati la finestra cancella tutti i dati;
- "finestra di accumulo": mantiene i risultati delle finestre delle emissioni precedenti.

Inoltre, le finestre possono essere definite anche secondo un punto di vista temporale come:

- "finestra di tumbling": definisce una serie di intervalli di tempo contigui e fissi in cui suddividere lo stream;
- "finestra di sessione": introduce il concetto di sessione, la quale è definita da un intervallo. I risultati calcolati su questa finestra vengono emessi se non arrivano nuovi messaggi per la durata di un intervallo definito.

Infine, è possibile utilizzare i trigger per perfezionare la strategia di windowing. Infatti, un segnale di trigger determina quando emettere i risultati di elaborazione effettuati sulla finestra. I trigger possono essere: trigger iniziali che consentono di emettere risultati prima che tutti i dati della finestra siano arrivati o trigger ritardati che consentono di gestire i messaggi in ritardo per la finestra.

Casi di utilizzo:

- Prevenzioni da frodi (es.: Ebay);
- Elaborazione stateful per ottimizzare le comunicazioni via email (es.: Linkedin);
- Elaborazioni di stream in real-time (es.: monitoraggio di infrastrutture).

Storage e formato:

Apache Samza supporta i seguenti "produttori" di stream: Apache Kafka, Microsoft Azure Eventhubs ed Elasticsearch. Inoltre, può interagire con database relazionali (es.: Google Cloud Spanner), NoSQL database (es.: Amazon DynamoDB) e con il file system offerto da Hadoop e quindi di conseguenza supporta i formati ottimizzati (avro, parquet, orc) e non (plain text, csv, json, etc.) ammessi dall'HDFS.

1.5.6 Apache Spark

Apache Spark è un engine per l'elaborazione di dati batch e stream su larga scala.

Permette di raggiungere alte prestazioni per entrambi i tipi di dati utilizzando un apposito scheduler DAG e un ottimizzatore di query. Oltre a ciò, ricordiamo che quando è possibile fornisce una computazione in memoria che permette di avere performance fino a 100 volte migliori rispetto alla versione di Hadoop che implementa il MapReduce.Anche nel caso in cui la computazione in memoria non fosse possibile, per motivi di occupazione, si ha comunque un incremento delle performance di un fattore 10. In aggiunta, presenta elevate prestazioni anche per quanto riguarda la scalabilità.

A dimostrazione di ciò ricordiamo il cluster più grande, e al momento esistente, il quale presenta 8000 nodi. Inoltre, anche in termini di dimensioni di dati, Spark ha dimostrato di funzionare bene fino all'ordine dei petabyte. Infine offre piena compatibilità con Hadoop ma può, tuttavia, essere eseguito anche in totale autonomia.

Bisogna però tener presente, in quest'ultimo caso, che nell'eventualità di esecuzione su un cluster è necessario che ci sia un qualche tipo di file system condiviso.

Librerie:

- Spark SQL: è un modulo Spark per l'elaborazione di dati strutturati, in grado di scalare anche tra migliaia di nodi. Esistono diversi modi per interagire con Spark SQL, tra cui l'API Dataset [20] e l'esecuzione di query SQL. Entrambe le modalità forniscono un metodo unico per accedere ad un'ampia varietà di sorgenti differenti (Hive, Avro, Parquet, Orc, Json e JDBC) ed eventualmente unire i dati prodotti da sorgenti diverse. L'API Dataset è disponibile in Scala e Java mentre il Python non la supporta, tuttavia, grazie alla sua natura dinamica molti dei vantaggi sono già intrinsecamente disponibili. Spark SQL supporta anche la sintassi del linguaggio HiveQL [21], il quale, in questo caso, offre performance migliori per l'esecuzione della

---

[20] *Opera su una raccolta distribuita di dati detta Dataset, la quale può essere organizzata in colonne denominate. In quest'ultimo caso si parla di DataFrame.*

[21] *Linguaggio con cui formulare le query eseguite da Apache Hive.*

query rispetto alle performance offerte dall'esecuzione della stessa query su Hadoop (inteso come implementazione del MapReduce). Dalla versione Spark 2.x è presente un engine di elaborazione di stream, detto Structured Streaming, che offre un'elaborazione del flusso veloce, esattamente una volta, scalabile, con latenze end-to-end di 100 millisecondi e tollerante ai guasti costruito su questo modulo (Spark SQL) e utilizzato per esprimere query su stream di dati. In questo modo sarà lo stesso Spark SQL ad eseguire in modo continuo ed incrementale la query man mano che i dati arrivino;

- GraphX: utile per la gestione di grafi e il calcolo parallelo sui grafi stessi;

- MLlib : libreria per il machine learning, il cui obiettivo è di rendere semplici e scalabili le tecniche di machine learning. Fornisce appositi tool per implementare algoritmi in grado di risolvere problemi (tabella 1.5) di diversa natura, salvare e caricare i modelli, estrarre feature, effettuare trasformazioni, riduzioni, selezioni, etc.;

Tipo di Problema	**Metodi implementati**
Classificazione binaria	Linear SVM, Regressione logistica (binomiale), Decision trees, Random forests, Gradient-booested trees, Naive Bayes
Classificazione multiclasse	Logistic regression (multinomiale), Decision trees, Random forests, Naive Bayes
Clustering	BisectingKMeans, KMeans, GaussianMixture, etc.
Regressione	Lasso, Ridge regression, Decision trees, Random forests, Gradient-booested-trees, Regressione lineare, etc.

Tabella 1.5 - Metodi che implementano il machine learning "tradizionale".

Per quanto riguarda invece "l'online machine learning" la libreria implementa solamente i metodi presenti nella tabella 1.6.

Tipo di Problema	Metodi implementati
Classificazione	Regressione logistica
Clustering	KMeans
Regressione	Regressione lineare

Tabella 1.6 - Metodi che implementano l'online machine learning.

- Spark Streaming: ha esteso il concetto di "batch processing" offrendo un'elaborazione in tempo reale o quasi di dati stream, i quali sono suddivisi in microbatch. E' quindi in grado di supportare le API di Apache Spark supportate dai dati batch. La tecnica del microbatching o fast batching, tuttavia, non rappresenta una soluzione valida in scenari in cui è richiesta una bassa latenza sui dati in arrivo. Infatti, in questo modo non si è in grado di raggiungere performance paragonabili ad altri framework quali Apache Storm, Apache Flink e Apache Apex che sono in grado di gestire in modo "puro" lo streaming. Tuttavia, dalla versione Spark 2.3 è stata introdotta una nuova modalità di elaborazione continua dello stream chiamata "Continuous Processing", che può ottenere latenze end-to-end a partire da 1 millisecondo con garanzia di esecuzione di almeno una volta. Per tutte le query, tranne quelle con funzioni di aggregazione, è possibile decidere la modalità di elaborazione "micro-batch processing" (default) o "continuous processing". Infine, la principale astrazione di alto livello che Spark Streaming fornisce è detta DStream e rappresenta il flusso continuo di dati e supporta le trasformazioni presenti nella tabella 1.7.

Trasformazione	Definizione
map *(func)*	Ogni elemento del DStream sorgente è elaborato dalla funzione *func*. Ritorna un nuovo DStream
flatMap *(func)*	Simile a map, ma ogni elemento in ingresso può ritornare zero o più elementi in uscita
filter *(func)*	Ritorna un nuovo DStream selezionando solo i record del DStream sorgente per cui la funzione *func* ritorna True
union *(otherStream)*	Ritorna un nuovo DStream che contiene l'unione degli elementi tra il DStream sorgente e otherStream

countByValue ()	Se applicata a uno DStream di elementi del tipo K, ritorna un nuovo DStream di (K, Long) coppie dove il valore di ogni chiave è la frequenza
join *(otherStream, [numTasks])*	Se applicata a due DStreams di coppie (K,V) e (K,W), ritorna un nuovo DStream di coppie (K, (V,W)) con tutte le coppie di elementi per ogni chiave

Tabella 1.7 - Principali trasformazioni supportate dal DStream.

Spark Streaming fornisce anche la possibilità di definire delle finestre temporali sulle quali è poi possibile applicare opportune trasformazioni (tabella 1.8).

Trasformazione	**Definizione**
window *(windowLength, slideInterval)*	Restituisce il DStream all'interno della finestra temporale definita
countByWindow *(windowLength, slideInterval)*	Ritorna il numero di elementi dello stream all'interno della finestra temporale definita
reduceByWindow *(func, windowLength, slideInterval)*	Ritorna un nuovo singolo stream creato aggregando elementi dello stream sullo *slideInterval* usando la funzione *func* la quale dovrebbe essere associativa e commutativa in modo che possa essere calcolata correttamente in parallelo
reduceByKeyAndWindow *(func, windowLength, slideInterval, [numTasks])*	Se chiamato su un DStream di coppie (K,V) restituisce un nuovo DStream di coppie (K,V) i cui valori per ogni chiave vengono aggregati usando la funzione di riduzione *func.* Una versione più efficiente permette di calcolare il valore di riduzione di ciascuna finestra usando il valore di riduzione della finestra precedente
countByKeyAndWindow *(func, windowLength, slideInterval, [numTasks])*	Quando viene chiamato su coppie di DStream (K, V), restituisce un nuovo DStream di coppie (K, Long) in cui il valore di ciascuna chiave è la sua frequenza all'interno della finestra scorrevole

Tabella 1.8 - Principali trasformazioni supportate dalla finestra temporale.

Casi di utilizzo:

- Machine learning: classificazione (es.: classificazioni in real-time di mail), clustering (es.: aggregazione delle notizie in base ai titoli) e "collaborative filtering" (es.: mostrare la pubblicità agli utenti in base alla loro posizione, acquisti, etc.);
- Analisi interattive: formulazione di query su stream di dati anche da parte degli utenti del servizio stesso;
- Rilevamento di eventi, streaming ETL, etc.

Storage e formato:

Apache Spark supporta l'interazione con l'HDFS e i formati da esso supportati (orc, parquet, avro), i NoSql database (Apache HBase, Cassandra, mongoDB) e i framework di data ingestion (Apache Kafka, Apache Flume, etc.).

1.5.7 Apache Storm

Apache Storm è un sistema open source di calcolo distribuito e in real-time che permette di rendere l'elaborazione (stateless) di flussi di dati stream illimitati molto semplice ed efficiente. Inoltre, presenta un parallelismo di task intrinseco il quale permette di processare con throughput molto alti (anche un milione di messaggi da 100 byte al secondo per nodo) e latenze molto basse. Rappresenta, inoltre, un sistema fault-tolerant che consente un'elaborazione continua anche in presenza di guasti, grazie alla possibilità di riassegnare i task a nodi diversi.

Concetti importanti:

- spout: è la sorgente che produce dati stream. Solitamente, uno spout legge i dati da sistemi di accodamento come JMS, Kafka, Redis pub/sub, Amazon Kinesis, ma può anche generare il proprio stream o leggere da API di streaming (es.: Twitter). Inoltre, si ha una completa integrazione anche con i RDBMS, i NoSql database (MongoDB, Cassandra, etc.), l'HDFS e di conseguenza con tutti i formati di dati che questi sistemi supportano;

- stream: sequenza di tuple (struttura dati principale) illimitate;

- bolt: elabora un numero qualsiasi di stream di input e produce un numero qualsiasi di nuovi flussi di output. Gestisce la maggior parte della logica di calcolo, come il filtraggio, i flussi di streaming, le aggregazioni di streaming, l'ottenimento dei dati dal database, etc. Inoltre, è in grado di gestire qualunque numero di stream in ingresso;

- topologia: l'elaborazione può essere rappresentata sotto forma di grafo diretto aciclico (DAG). I bolt e gli spout rappresentano i nodi, mentre gli stream rappresentano gli archi;

- utilizza Apache Zookeeper, il quale è un servizio di sincronizzazione e di configurazione distribuito e open source per il coordinamento e la gestione di grandi cluster su cui sono in esecuzione applicazioni distribuite.

API:

Presenta diverse API quali: Trident - per operazioni di analisi su stream -, Stream API - per esprimere elaborazioni su stream -, Storm SQL - per eseguire query SQL su dati stream - e Flux - per la creazione e la distribuzione di computazioni su dati stream -.

Meccanismi di windowing:

Una finestra raggruppa un insieme di tuple ed è descritta da opportuni parametri di configurazione, i quali ne descrivono per esempio il fenomeno temporale alla base (evento o durata), la tipologia di finestra in uso (scorrevole o fissa/tumbling), la dimensione, etc..

Casi di utilizzo:

- Analisi in tempo reale;
- Online machine learning;
- Computazioni continue di dati stream.

1.5.8 Apache Tez

Apache Tez è un framework estensibile per la creazione di applicazioni batch ed interattive. Tez affina il paradigma del MapReduce migliorandone drasticamente la velocità pur mantenendo la capacità, del MapReduce, di scalare fino a petabyte di dati.

E' costruito in cima allo strato Apache Hadoop YARN e modella l'elaborazione dei dati mediante un grafo diretto aciclico. E' utile pensare ad Apache Tez come l'impalcatura di una struttura alla base della costruzione di engine specifici che personalizzano l'elaborazione dei dati in base alle loro esigenze specifiche. Pertanto, questo framework introduce un ulteriore layer che offre la negoziazione delle risorse del cluster, tolleranza agli errori, sicurezza, ottimizzazioni delle prestazioni, etc. Tutto ciò permette di avere costi di sviluppo ammortizzati e prestazioni migliori. A tale proposito ricordiamo come un numero sempre maggiore di progetti appartenenti all'ecosistema Hadoop siano stati riscritti per poter utilizzare questo framework (es.: Apache Hive, Apache Pig, etc.) e i miglioramenti che esso stesso introduce. Di conseguenza, da questo punto di vista, Apache Tez può essere considerato come un insieme di librerie utili per creare "engine" specifici basati su flussi di dati.

API:

- La "DAG API" permette di definire pipeline complesse, la struttura alla base dell'elaborazione dei dati e il rapporto tra produttori e consumatori;
- la "Runtime API" definisce l'interfaccia che framework e applicazioni possono utilizzare per interagire tra di loro e le operazioni per trasformare e spostare dati tra i task.

Storage e formato:

Apache Tez si occupa solamente di gestire il movimento di dati, quali file e stream di byte, senza imporre alcun formato specifico. Inoltre, decide il tipo dei byte e il dispositivo di memorizzazione direttamente sul momento. I dati intermedi possono essere memorizzati su dispositivi distribuiti, locali o direttamente in memoria.

1.5.9 Google MillWheel

MillWheel è un framework per la creazione di applicazioni di elaborazione di dati stream a bassa latenza ampiamente utilizzato da Google e progettato sin dall'inizio prestando particolare attenzione ai concetti di scalabilità e tolleranza ai guasti. Inoltre, rappresenta un sistema che modella la computazione sotto forma di un grafo diretto dinamico.

I dati in input (non necessariamente ordinati o deterministici) e in output sono rappresentati dalla tripla (chiave, valore, timestamp). I valori (stringhe di byte) e i timestamp (momento in cui si verifica l'evento) sono entrambi arbitrari, mentre le chiavi assumono un significato specifico per il sistema. In particolare, esse rappresentano un'astrazione per l'aggregazione e il confronto di record e sono il risultato di opportune funzioni di estrazione. Infine, grazie all'utilizzo di appositi checkpoint si implementa un sistema fault-tolerance. Così facendo in caso di crash l'esecuzione potrà riprendere da risultati intermedi.

Ogni checkpoint viene eliminato quando si riceve l'ack che conferma la corretta ricezione dei record. La sequenza "checkpoint, consegna, ack, garbage collector" è chiamata produzione forte, mentre con il termine "produzione debole" si intende l'invio di eventi prima del salvataggio del loro checkpoint.

1.5.10 Google Cloud Dataflow

Google Cloud Dataflow è un servizio proprietario completamente gestito per l'esecuzione di pipeline (trasformare, arricchire, etc.) di dati stream e batch con affidabilità ed espressività garantite. Il modello di programmazione utilizzato è Apache Beam.

Grazie all'approccio serverless è infine possibile, sostenendo i costi previsti, usufruire di una capacità praticamente illimitata per l'elaborazione dei dati. Tale approccio permette allo sviluppatore dell'applicazione di concentrarsi solo ed esclusivamente sulle caratteristiche dell'applicazione piuttosto che sulla costruzione e gestione dei cluster su

cui sarà eseguita. Infatti, Dataflow in maniera completamente automatica è in grado di accelerare o decelerare l'attività delle macchine virtuali che compongono il cluster.

Meccanismi di windowing:

Il concetto di finestra consente di raggruppare insiemi di dati illimitati, dividendoli in opportune finestre di dimensioni finite, in base ai timestamp dei singoli elementi.

Le tre tipologie di finestre disponibili sono: "fixed time window", "sliding time window" e "session window".

SQL:

Innanzitutto, supportando Apache Beam è possibile formulare query direttamente sulla pipeline tramite "Beam SQL". Inoltre, tra i vari servizi per il cloud offerti da Google ricordiamo, in particolare, "Google BigQuery", un servizio completamente gestito e sviluppato per l'esplorazione dei dati, su un insieme arbitrariamente grande, utilizzando la sintassi del SQL (ANSI 2011).

Storage e formato:

Google Cloud Dataflow supporta l'interazione con diversi sistemi di "data ingestion" (Cloud Pub/Sub, Apache Avro, Apache Kafka), i NoSql database (Cloud Datastore), i RDBMS (es.: Postgres, MySQL), l'HDFS e i formati da essi supportati (dati relazionali, .xml, .json, .gcs, etc.).

Casi di uso:

- Machine learning;
- Rilevamento di frodi;
- Processi di estrazione, trasferimento, caricamento, filtraggio, etc.
- Gioco online;
- Analisi in real-time su piattaforme di commercio.

1.5.11 IBM InfoSphere Streams

IBM InfoSphere Streams offre soluzioni di big data “enterprise-ready” combinando alcune componenti di Apache Hadoop (MapReduce e HDFS) con tecnologie e funzionalità esclusive di IBM. In particolare, esso rappresenta una piattaforma software fault-tolerant che abilita lo sviluppo e l’esecuzione di applicazioni che elaborano dati stream, consentendone un'analisi continua e rapida, con basse latenze, al fine di poter offrire un processo decisionale su cui basare la propria applicazione. Il concetto di stream è, quindi, alla base di questa piattaforma. In particolare, con questo termine si fa riferimento a enormi volumi di dati in movimento, cioè dati che non sono ancora stati memorizzati su un opportuno dispositivo di memorizzazione.

Componenti integrati:

- il servizio di autenticazione Pluggable Authentication Module (PAM) per l'autenticazione utente insieme al backend LDAP [22];
- Apache ZooKeeper per la gestione e l'archiviazione delle informazioni di configurazione;
- un gestore delle risorse, che può essere interno o esterno (es: Apache Hadoop YARN);
- Streams Processing Language (SPL): linguaggio orientato ad un flusso di dati distribuito e presenta primitive e strutture adattate per lo streaming di dati.

Casi di utilizzo:

- Processare dati appena sono generati;
- Processare dati prima che siano scritti sul disco;
- Analisi di dati di tipo diverso: voce, video, audio, pacchetti di rete, etc..

Per carichi di lavoro “transazionali”, solitamente, non lo si utilizza. Tuttavia, se ciò fosse necessario è opportuno che sia presente un livello applicativo che ne garantisca la correttezza delle operazioni “transazionali”.

---

[22] *Protocollo standard per accedere e gestire servizi di directory distribuiti.*

Meccanismi di windowing:

- tumbling: l'intero contenuto della finestra viene elaborato e successivamente eliminato;
- sliding: tuple più recenti sostituiscono le tuple più vecchie se la finestra dovesse essere piena. La stessa tupla può essere presente in più finestre e di conseguenza può essere elaborata più di una volta.

Storage e formato:

Gli stream in ingresso, suddivisibili in tuple, supportano dati strutturati e non strutturati codificati in formato unicode text o binario (video, audio, etc.). Inoltre, si integra completamente con sistemi di memorizzazione quali i RDBMS (DB2, MySQL) e mediante l'utilizzo di opportuni toolkit anche con i NoSQL database (Cassandra, MongoDB) e l'HDFS.

SQL:

IBM InfoSphere Streams non supporta direttamente il linguaggio SQL, tuttavia si integra pienamente con IBM InfoSphere BigInsights. Quest'ultimo, il quale rappresenta uno strumento per l'esecuzione di analisi complesse e con elevate performance, si integra a sua volta con opportuni strumenti utili per formulare query (Apache Pig, etc.). Entrambi gli strumenti offerti da IBM supportano il linguaggio AQL, simile al SQL, per poter estrarre informazioni strutturate da testi non strutturati o semistrutturati.

1.5.12 Twitter Heron

Twitter Heron rappresenta una piattaforma open source, fault-tolerant e distribuita di analisi in real-time. Risulta essere completamente compatibile con Apache Storm in quanto costruita a partire da esso. Infatti, inizialmente Twitter utilizzava Apache Storm ma con l'aumento della popolarità della propria piattaforma e di conseguenza dei dati generati e da gestire, questo framework si è dimostrato non essere più adeguato. Dato ciò, piuttosto che utilizzare soluzioni alternative quali per esempio Apache Spark o Apache Flink e dover riscrivere nuovamente il codice della piattaforma, Twitter ha

deciso di sviluppare Twitter Heron basandosi su Apache Storm e migliorandone i punti deboli. Così facendo, ora, Twitter è in grado di gestire correttamente miliardi di eventi in real-time (es.: trend, conversazioni, suggerimenti, ricerche di tweet, etc.).

In particolare, Twitter Heron ha permesso di ridurre l'utilizzo delle risorse necessarie per la computazione (es.: CPU di 3-5x) e ha migliorato alcune feature fondamentali tra cui la latenza (5-10x), il throughput (7-10x).

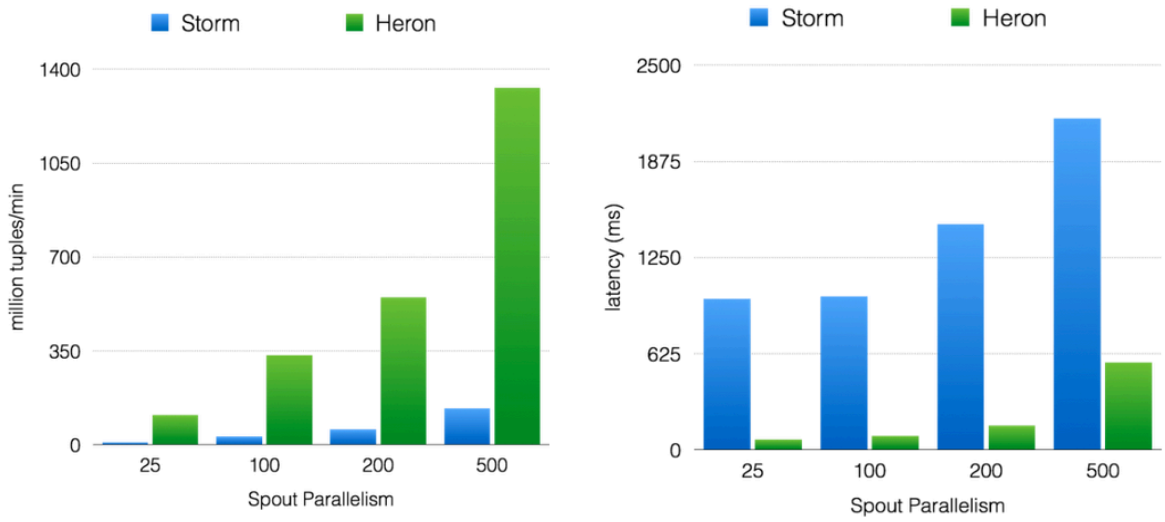

Grafico 1.3 - Confronto delle prestazioni tra Apache Storm e Twitter Heron.

Inoltre, presenta maggior facilità di debug e maggior scalabilità in orizzontale. Infine Heron permette di comporre grafi aciclici diretti (DAG) in grado di esprimere la logica di esecuzione di query in real-time e di inviare tale topologia su un sistema di pianificazione del lavoro (es.: Apache Aurora, YARN, Marathon, etc.) .

Storage e formato:

Twitter Heron è in grado di interagire con framework di "data ingestion" (es.: Apache Kafka, etc.), sistemi di memorizzazione quali l'HDFS, i RDBMS (es.: MySQL, Postgres, etc.) e i formati da essi supportati.

SQL:

E' in fase di sviluppo uno strumento che integri il linguaggio SQL.

Capitolo II

“Machine Learning”

2.1 Introduzione al ML

Innanzitutto è opportuno tener presente che non esiste una definizione universalmente riconosciuta e accettata su cosa esattamente sia il machine learning. Ciò nonostante, ricordiamo Arthur Samuel [1], il quale coniò il termine machine learning per designare il campo di studio che fornisce ai computer la capacità di poter apprendere senza che essi siano programmati esplicitamente.

Più in generale, con il termine machine learning si identifica una categoria di algoritmi alla base di applicazioni in grado di modificare e adattare le proprie azioni, a seconda dei dati in ingresso, affinché esse diventino sempre più accurate. L’accuratezza viene misurata in base a quanto queste azioni riflettano effettivamente le azioni corrette.

2.2 Algoritmi di machine learning

2.2.1 Training e Test Dataset

Ogni algoritmo di machine learning, indipendentemente dalla categoria alla quale appartiene, è caratterizzato da due fasi distinte alquanto fondamentali.

La prima fase è la cosiddetta fase di training, durante la quale si addestra il modello su un training set. Quest’ultimo contiene un ampio elenco di esempi pratici dai quali ottenere le conoscenze alla base del modello che si sta costruendo.

Successivamente si esegue una fase di test, durante la quale si verifica l’accuratezza del modello appena costituito su un insieme di test. Quest’ultimo in particolare deve avere la medesima struttura dell'insieme di addestramento, ma su dati differenti, e deve essere abbastanza grande affinché, statisticamente, contenga valori in grado di produrre

---

[1] *Pioniere americano nel campo dell’intelligenza artificiale.*

risultati significativi. Pertanto è importante non selezionare un insieme di test con caratteristiche differenti da quelle che caratterizzano l'insieme di training.

Inoltre, è opportuno tener presente che, in presenza di modelli di elevata complessità o non lineari è consigliato introdurre anche un validation set. In particolare con questo termine, il quale spesso viene utilizzato in maniera inappropriata come sinonimo di test set, ci si riferisce ad un insieme di dati utilizzati per ridurre al minimo il fenomeno dell'overfitting.

Pertanto, questo insieme è utilizzato per verificare un effettivo aumento dell'accuratezza del modello a seguito di un aumento dell'accuratezza del training set.

Infine, i dataset presi in considerazione sono ottenuti prendendo in analisi solamente un sottoinsieme dei dati a disposizione, solitamente un 40%, affinché in questo modo sia possibile addestrare il modello sulla porzione di dati maggiormente significativa presente nella raccolta.

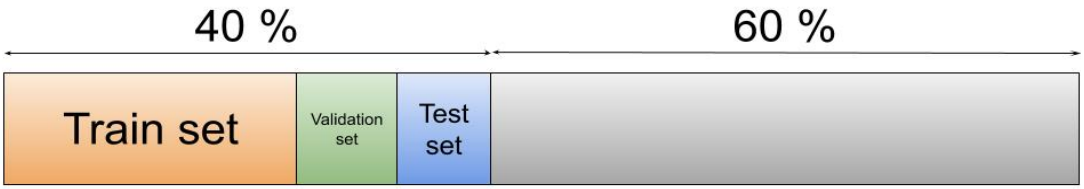

Figura 2.1 - Partizioni dataset.

2.2.2 Fitting del modello: underfitting e overfitting

Molto spesso, durante la costruzione di un modello, si può incorrere in particolari problemi noti come underfitting e overfitting. Con il primo termine ci si riferisce a situazioni in cui il modello non è in grado di cogliere l'andamento dei dati e di conseguenza viene meno la capacità di generalizzare su dati nuovi. Tale fenomeno può essere evitato utilizzando più dati.

Invece, con il secondo termine ci si riferisce a situazioni in cui il test set contiene valori molto simili, o talvolta in comune, con il training set. Così facendo, durante la fase di test, si noteranno risultati sorprendentemente positivi, ma che tuttavia non

rappresentano la vera accuratezza con cui il modello è in grado di generalizzare su dati nuovi.

Caratteristiche / problema	**Underfitting**	**Overfitting**
Performance sul training set	*poor*	*good*
Generalizzazione sul test set	*poor*	*poor*

Tabella 2.1 - Caratteristiche dell'underfitting e dell'overfitting.

Idealmente, nel caso in cui il modello sia in grado di fare previsioni senza errori si dice che esso presenta un buon adattamento ai dati. Questa situazione si colloca tra le due problematiche sopra descritte. Pertanto, quando si costruisce un modello, l'obiettivo principale è sempre quello di raggiungere il più possibile questa situazione ideale di corretto bilanciamento.

Un possibile e valido indicatore della qualità del modello è fornito dal rapporto tra le risposte corrette e il numero di test a cui la macchina viene sottoposta.

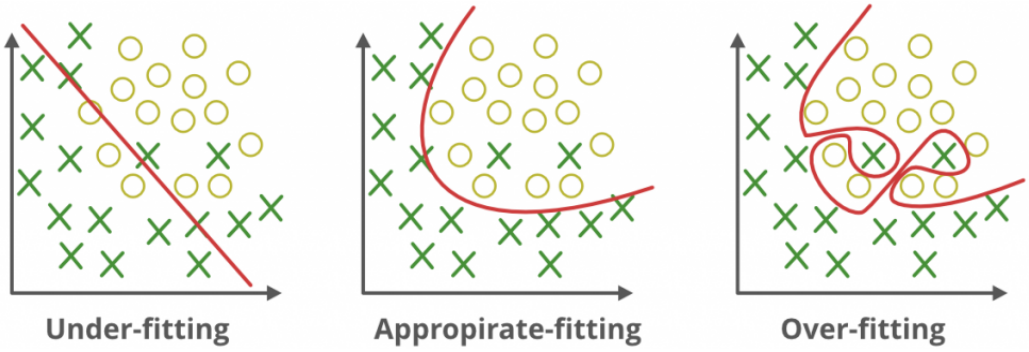

Figura 2.2 - Fitting di un generico modello.

2.2.3 Apprendimento

Esistono diverse categorie di apprendimento, tra le quali ricordiamo l'apprendimento supervisionato (supervised learning) e l'apprendimento non supervisionato (unsupervised learning).

Per quanto riguarda la prima categoria, viene fornito un training set di esempi con le risposte corrette (conferendo in questo modo una logica da seguire) e basandosi su questo insieme l'algoritmo generalizza per rispondere correttamente a tutti i possibili input. Questa tipologia di apprendimento rappresenta una macro categoria di algoritmi in grado di risolvere tipologie di problemi differenti noti come:

- regressione: metodo utilizzato in statistica per analizzare la relazione tra due variabili, una detta dipendente e l'altra indipendente. Quindi lo studio di un problema di regressione consiste nella determinazione di una funzione matematica che esprima la relazione fra le variabili. A tale proposito, la funzione da determinare deve passare il più vicino possibile a tutti i punti forniti. Questa ricerca della funzione matematica che rappresenta l'andamento del fenomeno è meglio nota con il termine di interpolazione;

- classificazione: consiste nel prendere vettori di input e decidere a quale delle N classi essi appartengono in base all'addestramento ottenuto dagli esempi mostrati per ogni classe.

Per quanto riguarda la seconda categoria - l'apprendimento non supervisionato - viene fornito un training set senza, tuttavia, le risposte corrette (in questo modo non viene conferita alcuna logica da seguire). Quindi l'algoritmo prova ad identificare le similarità tra i diversi input affinché quelli che risultano possedere caratteristiche in comune siano classificati insieme. Pertanto è il computer ad essere incaricato nel trovare una logica da seguire. Questa tipologia di apprendimento riassume un insieme di algoritmi in grado di risolvere numerosi problemi di diversa natura tra cui:

- clustering: è necessario il raggruppamento di dati che presentano caratteristiche simili. In particolare, dal punto di vista statistico, l'idea alla base di questo problema è quella di individuare feature simili tra gli elementi e di raggrupparli nel modo più omogeneo possibile. Inoltre, non esistono dei gruppi a priori, ma questi dipendono proprio dalle caratteristiche individuate. Pertanto, il clustering rappresenta un problema di classificazione senza tuttavia avere classi predefinite;

- associazione: è un problema per il quale si vogliono individuare le relazioni e le dipendenze che caratterizzano gli elementi presenti nel dataset a disposizione.

Infine, indipendentemente dalla tipologia di apprendimento, questi algoritmi hanno una caratteristica in comune nota come generalizzazione. Questa caratteristica rende il machine learning particolarmente significativo, poiché permette di produrre output rilevanti anche per input non incontrati durante la fase di training.

2.3 Machine learning: "tradizionale" e "online"

Negli ultimi anni, con l'aumento della complessità delle applicazioni, è emerso come il machine learning "tradizionale" non sia propriamente adatto a gestire tutti i possibili scenari in analisi.

A tale proposito, infatti, ricordiamo che ciò che caratterizza il machine learning "tradizionale" è la fase di training, la quale avviene una sola volta e di conseguenza i parametri del modello rimarranno per sempre costanti. Tuttavia, esistono situazioni in cui tale approccio presenta dei difetti o comunque non risulta essere particolarmente adatto. Esempi a dimostrazione di ciò si hanno in ambienti in cui i comportamenti cambiano molto rapidamente, come per esempio nell'ambito dello shopping online in cui un prodotto che è popolare oggi può non esserlo domani. Per gestire applicazioni di questo tipo è nato il cosiddetto "online machine learning", in cui il modello previsionale è in grado di evolvere nel tempo man mano che nuovi dati in real-time vengono aggiunti al training set.

Parte II

Studio di un'applicazione reale

Capitolo III

“Applicazione reale: progetto”

3.1 Descrizione della realtà da analizzare

La seguente trattazione si pone come obiettivo la progettazione di un’applicazione che risulti essere capace di elaborare efficientemente stream di dati, in real-time, prodotti da indistinti utenti di una tra le più note e utilizzate piattaforme di social network, ovvero Twitter.

In particolare, i dati presi in considerazione altro non sono che i singoli tweet pubblicati quotidianamente dagli utenti e, più nello specifico, quelli riguardanti tematiche concernenti il vasto mondo dell’arte, comprendendo, quindi, opere, artisti, musei, installazioni ed eventi. Tutto ciò al fine di sfruttare opportune tecniche di machine learning per la costruzione di un modello previsionale che risulti essere in grado di evolvere nel tempo e che effettui, nel modo più accurato possibile, previsioni riguardo l’apprezzamento e, di conseguenza, l’indice di popolarità, che un determinato tweet dovrebbe riuscire a raggiungere sulla piattaforma.

3.2 Progettazione concettuale

3.2.1 Strategia progettuale

Innanzitutto, affinché sia possibile gestire in modo appropriato la complessità dell’applicazione sopra descritta, è opportuno, grazie anche alle analisi effettuate nel capitolo precedente, procedere con una progettazione del sistema per step graduali, andando passo dopo passo a riflettere sulle alternative esistenti e più adatte al soddisfacimento delle richieste progettuali.

A tale proposito, la progettazione avverrà partendo dalla "pipeline concettuale", (figura 3.1) deliberatamente vaga, che riassume in modo conciso i punti chiave alla base dell'applicazione.

Figura 3.1 - Pipeline concettuale.

3.2.2 Sorgente

Il social network Twitter risulta essere un ottimo esempio di piattaforma che produce stream di dati in real-time. Infatti, grazie alla sua popolarità e ai milioni di utenti iscritti sparsi nel mondo, si ha la certezza di avere un stream di dati continuo durante tutto il corso della giornata. Tuttavia, visto il numero elevato di utenti che ogni giorno usufruiscono della piattaforma e la grande vastità di tematiche dei vari tweet pubblicati, è stato necessario, al fine di riuscire nella costruzione di un modello previsionale corretto e che rispetti le richieste progettuali, ridurre, innanzitutto, in modo opportuno i dati generati dagli utenti della piattaforma.

A tale proposito, quindi, è stato applicato un filtraggio iniziale in base alla tematica riguardo cui gli utenti pubblicano tweet per poi applicarne uno ulteriore sul numero di follower di questi utenti, al fine di riuscire ad ottenere un insieme di dati il più possibile "omogeneo".

3.2.3 Framework di data ingestion

Innanzitutto, è stato deciso di estendere la pipeline mostrata dalla figura 3.1 introducendo un framework di data ingestion. L'introduzione di questa tipologia di framework risulta essere particolarmente utile in scenari in cui è necessario "raccogliere" dati, che necessitano di medesime elaborazioni, da sorgenti diverse.

Ovviamente questo scenario non rientra specificamente nell'analisi qui trattata, tuttavia, approfondendo i diversi framework di data ingestion (Flink, NiFi, Sqoop, Kafka, etc.) al momento gestiti dall'Apache Foundation, è stato notato un framework in particolare, il quale, rispetto agli altri, presenta particolari caratteristiche che in scenari di cluster reali rappresentano veri e propri punti di forza per il corretto funzionamento del sistema.

Il framework in questione è Apache Kafka, e tra le diverse funzionalità che esso offre, ricordiamo la realizzazione di un disaccoppiamento tra la sorgente e l'engine di elaborazione. Ciò è reso possibile grazie alla memorizzazione intrinseca dei flussi di record in modo duraturo e tollerante ai guasti e al modello di messaggistica che esso stesso implementa. Grazie a tali feature, in scenari di guasti o in caso di necessità di manutenzione dei server su cui è in esecuzione l'engine, i dati prodotti durante il periodo di inattività dell'engine saranno memorizzati temporaneamente (a seconda delle configurazioni scelte) in attesa di un ripristino del corretto funzionamento dell'engine.

Sebbene l'applicazione in questione non verrà eseguita in un ambiente cluster o in scenari di guasti, è stato ugualmente deciso di introdurre questo framework al fine di approfondire le configurazioni necessarie per il corretto funzionamento della pipeline.

Inoltre, grazie alle feature di questo framework, nel caso specifico dell'applicazione in questione, ogniqualvolta verrà riscontrato un bug nel codice dell'engine, sarà possibile apportare le modifiche evitando di perdere i dati generati durante il periodo di inattività, dovuto alla fase di debugging.

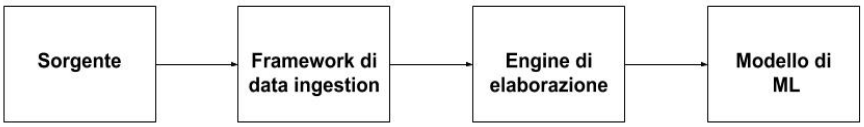

Figura 3.2 - Pipeline concettuale raffinata.

3.2.4 Scelta del framework

Data la presenza di più framework, la scelta ha seguito i seguenti parametri:

- framework open source;
- framework che implementa un engine di elaborazione;
- possibilità di gestire il processing di dati stream;
- integrazione nativa ed estensibile con il machine learning su dati in streaming;
- supporto al linguaggio di programmazione Python;
- popolarità dei framework: il grafico rappresenta l'interesse di ricerca, su Google, nel mondo durante l'ultimo anno (grafico 3.1).

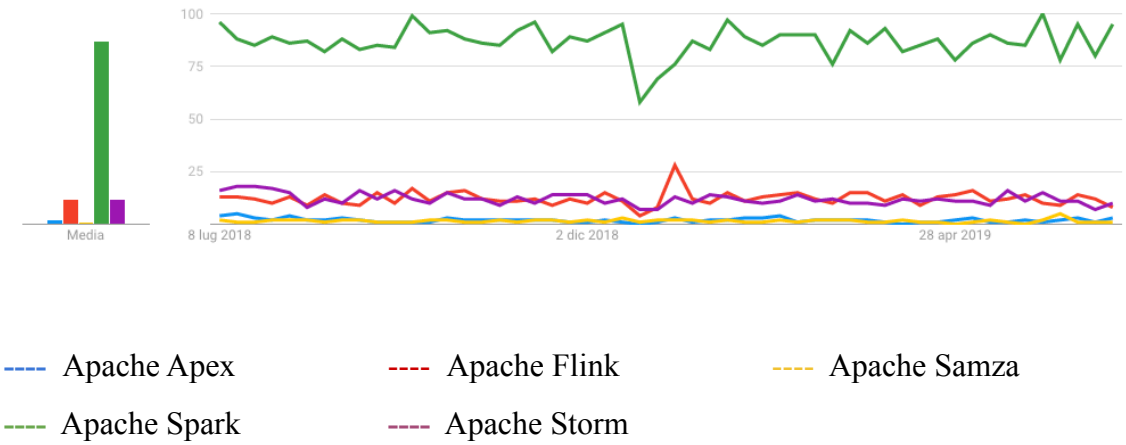

Grafico 3.1 - Popolarità dei framework.

Il valore 100 indica la massima frequenza di ricerca del termine, il valore 50 indica la metà delle ricerche. Un punteggio pari a 0, invece, indica che non sono stati rilevati dati sufficienti.

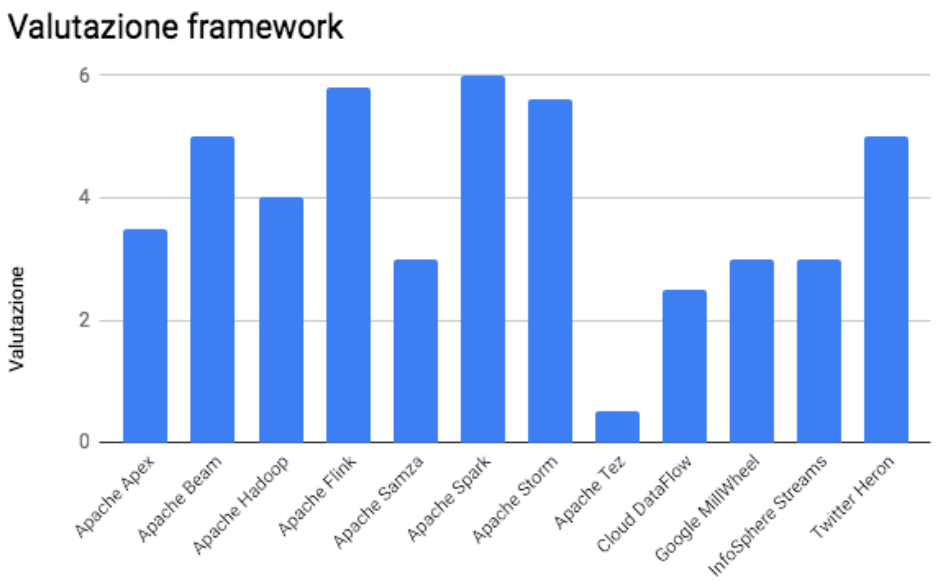

Grafico 3.2 - Valutazione framework.

Il grafico 3.2 è ottenuto assegnando ad ogni sistema un punteggio in base al fatto che esso soddisfi o meno ognuno dei parametri sopra elencati.

Si noti come i framework Apache Flink, Apache Spark e Apache Storm risultano essere i più promettenti. Di conseguenza è stato deciso di effettuare ulteriori approfondimenti. A tale proposito, da un'analisi più accurata di Apache Flink è emerso che esso:

- non supporta la versione nativa del Python ma piuttosto una sua implementazione scritta in Java e meglio nota come Jython;
- l'ultima versione di Python, così supportata, è la 2.7;
- la documentazione di esempio in Python è ancora molto ridotta;
- presenta difficoltà di integrazione con framework di data ingestion, quali per esempio Apache Kafka;
- l'API DataStream, in Jython, è ancora in versione beta e presenta diversi bug noti;
- non presenta una API per l'online machine learning.

Tutto ciò non pregiudica la validità di Apache Flink bensì, al momento, si ha semplicemente un maggior controllo delle funzionalità che esso offre solo in altri linguaggi di programmazione, quali Java e Scala.

Per quanto riguarda Apache Storm, invece, è emerso che esso presenta le seguenti limitazioni:

- la documentazione ufficiale riporta che la Stream API e Storm SQL sono ancora in versioni sperimentali;
- Storm SQL non presenta il concetto di windowing, di aggregazione e di join;
- teoricamente è un framework multi-linguaggio, ma per eseguire, per esempio, un codice scritto in Python è necessario scrivere opportune classi in Java per l'invocazione dei metodi scritti in Python. Infine, per poter eseguire un job su Apache Storm è necessario esportare il codice scritto, con tutte le relative dipendenze incluse nel file pom.xml, in formato jar. Per rendere più semplice la scrittura di codice in Python, è possibile utilizzare la libreria streamparse, la quale permette di creare spout (interfacciabili anche con Apache Kafka) e bolt (per l'elaborazione dello stream), ma che, tuttavia, non permette l'utilizzo di tutte le funzionalità native di Apache Storm (es.: non permette di definire windowing e formulare query in SQL).

A seguito di queste analisi e delle numerose difficoltà incontrate per l'implementazione di applicazioni di test in Python con Apache Storm e Apache Flink, da questo momento in poi, il focus della trattazione verterà unicamente su Apache Spark, il quale oltre ad offrire tutte le principali funzionalità anche in Python, offre anche una valida libreria per l'online machine learning. Prima di concentrarci esclusivamente su Apache Spark è possibile notare come, da queste analisi, sia emersa una certa difficoltà nel trovare un framework, oltre ad Apache Spark, con un completo supporto al Python. In particolare è emersa la centralità, alla base degli engine di elaborazione, ma non solo, del linguaggio di programmazione Java e dei tool offerti dal JDK [1]. E' quindi opportuno soffermarsi sui motivi di tale centralità. A tale proposito ricordiamo:

---

[1] *Ambiente di sviluppo sw che presenta un set di strumenti utili per sviluppare programmi.*

- Hadoop, il quale ricordiamo essere il framework cardine nel panorama dei Big Data, per motivi storici e di compatibilità con il progetto Nutch (web crawler) venne implementato principalmente in Java (piccole porzioni di codice C e script shell);

- elevata presenza del JDK in ambienti cluster;

- portabilità e indipendenza dalla piattaforma grazie alla presenza del Java Virtual Machine;

- funzionalità avanzate del garbage collector per la gestione della memoria;

- facilità di debug a run-time.

Per questi motivi i framework sviluppati nell'ambito dei Big Data sono in prevalenza scritti in Java o talvolta in Scala (utilizza il JVM). Di conseguenza i framework che supportano altri linguaggi di programmazione oltre Java e Scala, tendono ad avere performance di poco inferiori e funzionalità ridotte. Ciononostante, tali limitazioni saranno presenti ancora per poco tempo, dal momento che tutti i principali engine di elaborazione offrono già la possibilità, in modo più o meno semplice, di implementare un insieme ridotto di funzionalità usufruendo di altri linguaggi.

3.2.5 Modello di machine learning

Al fine di poter effettuare una scelta corretta del modello di machine learning adatto alla risoluzione delle richieste progettuali ricordiamo le scelte effettuate nei punti precedenti.

Infatti, tali scelte insieme alle richieste progettuali hanno condizionato in modo inequivocabile la scelta dell'algoritmo corretto da applicare in fase di implementazione.

La tabella 3.1 riassume gli elementi presi in considerazione.

Scelte		Motivazioni
Framework	Apache Spark	Framework emerso in fase di analisi
Tipologia di ML	Online machine learning	Le richieste progettuali specificano la necessità di un modello in grado di evolvere nel tempo
Tipologia di apprendimento	Supervisionato	Può essere dedotto implicitamente dalla tipologia di dati in considerazione
Tipologia di problema	Classificazione binaria	Le richieste progettuali specificano la necessità di determinare se un tweet sarà popolare o meno

Tabella 3.1 - Considerazioni progettuali.

Pertanto, tra i vari metodi implementati dalla libreria di MLlib inclusa da Apache Spark, è stato scelto il metodo denominato "Streaming Logistic Regression".

E' opportuno precisare che, a differenza di quanto possa suggerire il nome, esso può essere utilizzato come classificatore binario, classificando le osservazioni in base alle loro caratteristiche. Questo in quanto rappresenta un modello di regressione non lineare utilizzato quando la variabile dipendente è di tipo dicotomico con l'obiettivo di stabilire la probabilità con cui un'osservazione può generare uno o l'altro valore della variabile dipendente. Tale classificatore può essere descritto dalla seguente formula:

$$P(y_i = 1 \mid X) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}$$

Il primo termine rappresenta la probabilità che all'i-esima osservazione, y_i valga 1 dato l'insieme di training X e con β_0 e β_1 come parametri da apprendere.

Il secondo termine rappresenta una funzione logistica anche detta sigmoide, il cui esponente è rappresentato dal modello lineare ($\beta_0 + \beta_1 x$).

Capitolo IV

"Applicazione reale: implementazione"

La finalità del presente capitolo è di fornire una panoramica che illustri le principali configurazioni approfondite ed implementate, necessarie per un corretto funzionamento dell'applicazione descritta nel capitolo precedente.

4.1 Overview del codice

Innanzitutto, al fine di sfruttare appieno le funzionalità offerte dal framework Apache Kafka (es.: modello di messaggistica del tipo produttore-consumatore, etc.) il codice è stato diviso in due moduli distinti.

Il primo modulo è il cosiddetto produttore e quindi al suo interno sono state implementate le configurazioni per il corretto accesso ai tweet e per il corretto invio delle informazioni necessarie per la fase di training e di test fondamentali per la corretta costruzione del modello previsionale.

Mentre per quanto riguarda il secondo modulo, esso implementa il cosiddetto consumatore e pertanto al suo interno sono stati implementati diversi metodi necessari per la fase di filtraggio dei tweet in ingresso e per la costruzione e l'aggiornamento del modello previsionale.

4.2 Libreria Tweepy

Twitter mette a disposizione diverse API (Standard, Premium, Enterprise, Ads, etc.) per aiutare a creare applicazioni che necessitano di un'integrazione con tale social network. In particolare, per la realizzazione di tale applicazione è stata utilizzata la "Standard API", in quanto pensata appositamente per fini sperimentali. Infine, considerata la scelta del Python, quale linguaggio di programmazione, l'accesso a tale API avviene tramite l'utilizzo della libreria Tweepy. Questa libreria semplifica l'uso

dell'API di Twitter gestendo l'autenticazione, la connessione, la creazione di una sessione, la lettura dei tweet in arrivo, etc.

4.3 PyKafka

PyKafka è un client Kafka "programmer-friendly" scritto in Python.

L'obiettivo principale di questo package è quello di fornire un livello di astrazione simile al client JVM Kafka usando idiomi familiari ai programmatori Python. A dimostrazione di ciò, tale package include le implementazioni dei concetti che caratterizzano il framework Kafka (es.: produttori, consumatori, etc.).

4.4 Configurazione dell'architettura introdotta da Kafka

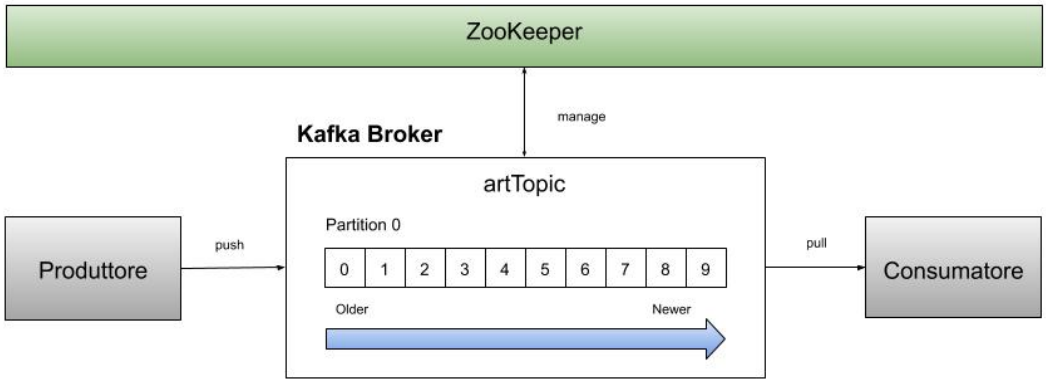

Figura 4.1 - Architettura introdotta da Kafka.

Innanzitutto è necessario configurare tale architettura, dal terminale, tramite il seguente comando:

> bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --config retention.ms=7200000 --topic artTopic

Parametri:

- zookeeper localhost:2181: Kafka utilizza ZooKeeper, il quale si occupa della gestione e dello stato del cluster Kafka. Per questo motivo è necessario specificare la posizione del nodo su cui è in esecuzione il server Zookeeper (di default è in esecuzione sulla porta 2181);

- replication-factor: indica il numero di server in cui ogni messaggio scritto viene replicato. In scenari reali è consigliato avere un fattore di replicazione pari a 2 o 3;

- partitions: parametro utile per una corretta gestione del parallelismo (es.: per aumentare il numero di produttori e di consumatori);

- retention.ms: questo parametro controlla il tempo massimo per il quale si conserva un log prima di eliminare i vecchi segmenti al suo interno per liberare spazio. Il valore di default è 604800 secondi, mentre se impostato su -1 non viene applicato alcun limite di tempo. Il valore impostato per questa applicazione è pari a due ore. Così facendo, ogniqualvolta verrà riscontrato un bug nel codice dell'engine, si avranno a disposizione due ore di tempo per apportare le modifiche, prima di avere un'effettiva perdita di dati;

- topic: categoria all'interno della quale i messaggi sono pubblicati. Per ogni topic, Kafka gestisce il numero di partizioni configurate.

4.5 Produttore

4.5.1 Snippet di codice riguardante la configurazione

```
...

# Configurazione protocollo OAuth.
auth = tweepy.OAuthHandler(consumer_token, consumer_secret)
auth.set_access_token(access_token, access_secret)

# Istanza dell'API.
api = tweepy.API(auth)

# KafkaListener è una sottoclasse di StreamListener.
# Per ulteriori dettagli si rimanda al paragrafo 4.5.3.
streamListener = KafkaListener()

# Istanza necessaria per stabilire una sessione di streaming e indirizzare i messaggi
# all'istanza streamListener.
twitterStream = tweepy.Stream(auth, streamListener)
...

    try:

        # Filtraggio dello stream a seconda delle parole contenute nei tweet.
        twitterStream.filter(track = words)

    except Exception as ex:

        # Gestione delle eccezioni.
        exception = "An exception of type {0} occurred. Arguments:\n{1!r}"
        print(exception.format(type(ex).__name__, ex.args))
```

4.5.2 Spiegazione del codice

OAuth Authentication:

Twitter supporta lo standard OAuth, il quale, ai fini dello sviluppo di applicazioni, rappresenta un modo per gli utenti della piattaforma di condividere informazioni sui propri account con applicazioni di terze parti senza fornire loro le password.

Tweepy cerca di rendere questo meccanismo il più semplice possibile tramite il metodo *OAuthHandler* il quale prende in ingresso il *consumer_token* e il *consumer_secret*.

Inoltre, per avere accesso ai tweet pubblicati e a tutte le informazioni ad essi correlate è necessario impostare ulteriori due token (*access_token* e *access_secret*) tramite il metodo *set_access_token.* Per ottenere i valori di questi quattro token è necessario registrare l'applicazione client su Twitter.

Classe tweepy.API():

Questa classe fornisce un wrapper per l'API fornita da Twitter, i cui parametri in ingresso sono tutti opzionali. L'unico parametro fornito è il cosiddetto *auth_handler*, il quale rappresenta il gestore di autenticazione da utilizzare.

Classe tweepy.StreamListener():

E' stata creata la classe *KafkaListener* la quale eredita da *StreamListener* e ne sovrascrive i metodi *on_status* e *on_error.* Per ulteriori dettagli si rimanda al paragrafo 4.5.3.

Filtraggio:

Il filtraggio dei tweet pubblicati, il quale è case insensitive, avviene sulla base dei termini riportati nella tabella 4.1.

National Gallery	Museo del Prado	Metropolitan Museum of Art	National Gallery of Art	Vatican Museums
Metropolitan Museum	Centre Pompidou	Hermitage Museum	Tate Modern	Reina Sofia
Uffizi	Louvre	Picasso	Buonarroti	Matisse
Rousseau	Cézanne	Magritte	Dalí	Renoir
Modigliani	Chagall	Botticelli	Van Gogh	Caravaggio
Escher	Claude Monet	Miró	Klimt	Pollock
da Vinci	Guggenheim	British Museum	Seurat	Kandinsky
Renoir	Edvard Munch	Bruegel	Vermeer	Paul Gauguin

Tabella 4.1 - Termini su cui è eseguito il filtraggio.

Si noti come sia possibile che tweet contenenti alcuni di questi termini, in realtà non trattino di tematiche artistiche, nonostante si sia cercato di ridurre al minimo tale probabilità, tramite la selezione di termini che siano il più possibile legati a tematiche di tale genere.

Questo insieme di termini garantisce, per tutto il corso della giornata, un input rate medio di 0,9 record/sec (grafico 4.1).

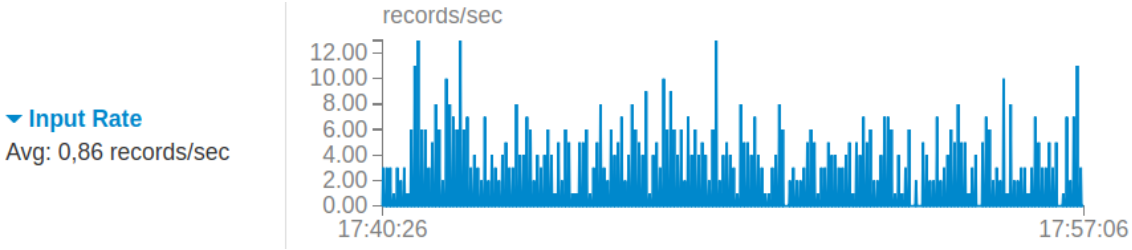

Grafico 4.1 - Input rate.

4.5.3 Snippet di codice riguardante la classe KafkaListener

```
def __init__(self):

    ...

    # Configurazione del server su cui è in esecuzione Kafka.
    self.client = pykafka.KafkaClient("localhost:9092")

    # Creazione di un produttore di messaggi per l'argomento specificato.
    self.producer = self.client.topics[bytes("artTopic","utf-8")].get_producer()

# Metodo per la costruzione e l'aggiornamento dell'insieme di training e di test.
def update(self):
    ...

    # Invocazione del metodo per ottenere i tweet che aggiorneranno il modello.
    # Per ulteriori dettagli si rimanda al paragrafo 4.5.5.
    modelElement = updateModelML()

    if modelElement:

        # Invio dell'elemento al modello in costruzione.
        [self.producer.produce(bytes(element, "utf-8")) for element in modelElement]

# Override del metodo ereditato da StreamListener.
def on_status(self, status):

    # Invocazione del metodo per conoscere la tipologia del tweet ("originale", retweet,
    # di risposta, etc.).
    infoTweet = TweetCreator.from_creator(status)
```

```
# Informazioni minime caratterizzanti il tweet (tipologia, id, follower dell'autore).
streamData =  infoTweet + " " + str(status.id_str) + " "  + \
                    str(status.user.followers_count)

# Invio delle informazioni all'engine di elaborazione.
self.producer.produce(bytes(streamData,"utf-8"))

...
```

4.5.4 Spiegazione del codice

Classe pykafka.KafkaClient():

Tramite l'istanziazione di un oggetto di tale classe è innanzitutto possibile configurare la porta (es.: 9092) su cui il server Kafka è in esecuzione e su cui rimarrà in ascolto.

Inoltre, è fondamentale istanziare anche un produttore per l'argomento - artTopic - creato in precedenza durante la fase di configurazione dell'architettura.

Metodo on_status():

Tale metodo invia ogni singolo tweet contenente almeno uno dei termini riportati nella tabella 4.1 all'engine di elaborazione. Dal momento che in questa fase ancora non è noto se tale tweet verrà effettivamente utilizzato per la costruzione o l'aggiornamento del modello previsionale, ci si limita ad inviare all'engine le informazioni minime per effettuare tale decisione. In particolare si inviano soltanto le informazioni riguardanti:

- la tipologia del tweet. Infatti, esistono diverse tipologie di tweet, quali per esempio i tweet "originali", i retweet, i tweet in risposta ad altri tweet, etc. Inoltre, si evidenzia come tutte queste tipologie, tranne ovviamente i tweet originali, essendo in relazione con un altro tweet, in aggiunta alle proprie informazioni, contengono anche tutte le informazioni di tale tweet. Inoltre, si rileva che, statisticamente, la probabilità che anche tali tweet contengano a loro volta una delle parole elencate nella tabella 4.1 è decisamente molto bassa. Infine, si consideri che il numero e la tipologia dei campi

che li caratterizzano risulta essere molto variabile. Per tali motivi l'attenzione è stata focalizzata solamente sui cosiddetti tweet "originali", evitando di complicare la fase di filtraggio, che avrebbe potuto produrre un insieme di tweet un minimo più ampio, con conseguente potenziale peggioramento delle performance;

- id: identificativo univoco del tweet;

- numero di follower dell'autore del tweet: la conoscenza di tale valore è fondamentale, in quanto l'engine di elaborazione effettuerà un filtraggio dei tweet anche in base a questo valore. Questo in quanto è stata effettuata la scelta di concentrarsi sugli utenti con un numero di follower compreso tra 1.000 e 30.000 al fine di avere un insieme di utenti omogenei e aventi caratteristiche simili. Inoltre, tale range è stato scelto a seguito dell'analisi dei seguenti intervalli:

1. follower < 1.000: ~ 5,8 tweet/minuto soddisfano la fase di filtraggio, tuttavia tendono ad essere in prevalenza tweet con un numero di like molto basso e di conseguenza si hanno delle difficoltà nella creazione dell'insieme contenente tweet good e tweet bad;

2. follower >= 1.000 e follower <= 30.000: ~ 3,6 tweet/minuto soddisfano la fase di filtraggio. La frequenza di tweet good risulta essere maggiore rispetto alla soglia descritta in precedenza;

3. follower >= 30.000 e follower <= 100.000: ~ 0,7 tweet/minuto soddisfano la fase di filtraggio;

4. follower >= 100.000: ~ 0,4 tweet/minuto soddisfano la fase di filtraggio;

Si noti come tali valori sono da intendersi come valori medi misurati in fasce orarie differenti.

Metodo update():

Questo metodo, ogni minuto, viene eseguito in background tramite un apposito thread al fine di verificare la presenza di tweet utili per quanto riguarda l'aggiornamento o il test del modello previsionale in costruzione. Tutto ciò è reso possibile tramite l'invocazione del metodo *updateModelML()*. In tal modo l'engine di elaborazione continuerà in contemporanea a ricevere, oltre ai tweet per l'aggiornamento del modello, anche quelli pubblicati sul momento.

4.5.5 Snippet di codice riguardante il metodo updateModelML

```
# Metodo per il recupero degli elementi che aggiorneranno il modello.
def updateModelML():

    ...

    # Lettura dei tweet salvati in passato e in attesa di essere elaborati.
    rowsArrivedTweet = utilityFile.readRows("arrivedTweet.txt")

    # Ciclo eseguito per ogni tweet memorizzato temporaneamente.
    for row in rowsArrivedTweet:

        fields = row.split(" ")

        # Estrazione dell'id del tweet.
        idTweet = fields[0]

        ...

        # Calcolo del tempo trascorso dalla pubblicazione del tweet.
        timeInterval = computeTime(timeValue[:-1])
```

```
# Verifica che l'intervallo di tempo sia di almeno un'ora e di aver letto
# un id valido.
 if timeInterval >= 60 and idTweet.isdigit():
    ....
    try:
       stato = api.get_status(idTweet)

    # Potrebbe essere sollevata un'eccezione nel caso in cui il tweet
    # sia stato eliminato.
    except Exception as ex:
       print("\n Impossibile aggiornare il tweet :", idTweet, ex)

    else:
       # Numero di like del tweet.
       like = stato.favorite_count

       infoTweet = ''

       # Soglia oltre la quale un tweet è considerato GOOD.
       if like >= 15:
          infoTweet = 'GOOD'
          goodCount += 1

       # Soglia al di sotto della quale un tweet è considerato BAD.
       elif like <= 3:
          infoTweet = 'BAD'
         badCount += 1

       # Verifica che il tweet sia stato etichettato come GOOD oppure come BAD.
       # In tale modo, implicitamente, si ha un filtraggio che rimuove i tweet della
       # gray area (tweet con  like > 3 e like < 15).
```

```
if infoTweet:

    # Numero di url in esso contenuti.
    numberUrls = len(stato.entities['urls'])

    # Numero di tag ( @ ) ad utenti in esso contenuti.
    numberMentions = len(stato.entities['user_mentions'])

    # Numero di hashtag ( # ) in esso contenuti.
    numberHashtags = len(stato.entities['hashtags'])

    # Numero di media (foto, video, gif) nativi in esso contenuti.
    numberMedias = len(stato.entities['media']) if ('media' in stato.entities.keys())  \
                        else 0
    try:

        # Numero di media (foto, video, gif) in esso contenuti.
        if hasattr(stato, 'extended_tweet') and  \
          'media' in stato.extended_entities.keys():
          numberMedias = len(stato.extended_entities['media'])

    except:
        pass

     # Tweet per il training del modello.
    typeTweetML = 'TRAIN'
    ...

    # Verifica la presenza di un addestramento bilanciato.
    if ('GOOD' in infoTweet and goodCount < 10) or \
      ('BAD' in infoTweet and badCount < 10):
```

```
# Verifica che sia in esecuzione la prima fase di training
if firstTrain < 180:
    firstTrain += 1

# Per le fasi di training successive alla prima.
else:
    nextTrain +=1

...

# Verifica che sia in atto la fase di test, la quale avviene al
# termine di ogni iterazione di training. Quest'ultima è eseguita
# su 80 elementi, tranne la prima fase che invece viene eseguita
# su 180 elementi.
elif nextTrain > 0 and (('GOOD' in infoTweet and goodTest < 10) \
     or ('BAD' in infoTweet and badTest < 10)) :

    typeTweetML = 'TEST'

    if 'GOOD' in infoTweet:
        goodTest += 1
    else:
        badTest += 1

# Informazioni da inviare per il training e il test del modello.
modelInfo  =  infoTweet + " " + typeTweetML + " " +  \
              str(stato.retweet_count) + " " + str(len(stato.text))  + \
              " " + str(numberUrls) + "  " + str(numberMentions) + \
              " " + str(numberMedias) + "  " + str(numberHashtags)
```

4.5.6 Spiegazione del codice

Tramite l'invocazione del metodo *computeTime()* si verifica periodicamente - ogni minuto - il tempo trascorso dall'arrivo di ogni singolo tweet memorizzato temporaneamente. Infatti, è opportuno evidenziare come ogni tweet, che rispetti i parametri impostati nella fase di filtraggio (eseguita da parte dell'engine), venga salvato temporaneamente (per un'ora) in un file. In particolare, si ricordi come vengono memorizzate le minime informazioni necessarie (id del tweet e timestamp).

Al superamento dell'ora di osservazione, se il tweet dovesse soddisfare una delle due soglie impostate (like <= 3 oppure like >= 15) si effettua, tramite l'id memorizzato nello step precedente, il recupero delle feature che lo caratterizzano. Per quanto riguarda i valori delle soglie, essi sono stati scelti dopo un'accurata analisi di numerosi tweet aventi le caratteristiche descritte nei precedenti paragrafi (presentano almeno un termine della tabella 4.1 e l'autore del tweet ha un numero di follower compresi tra 1.000 e 30.000). In particolare, la strategia seguita per la scelta dei valori delle soglie è stata quella di raccogliere tweet per la durata di mezz'ora e di verificare, visitando il profilo dei relativi autori, la popolarità raggiunta dai tweet da essi pubblicati in passato e riguardanti l'arte. Inoltre, tutto ciò è stato possibile, in modo molto proficuo, in quanto è emerso come gli autori di tale tipologia di tweet siano in maggioranza autori di nicchia, cioè autori che in prevalenza pubblicano tweet inerenti l'arte.

I tweet che soddisfano tali soglie vengono utilizzati per la fase di training o per quella di test. In particolare, ad ogni iterazione di aggiornamento del modello, si costruisce un insieme di tweet in cui i primi 80 sono utilizzati per la fase di training mentre i successivi 20 per la fase di test. L'unica eccezione si ha solamente durante la prima iterazione, durante la quale la fase di training avviene su un insieme di 180 tweet. In ogni caso, in entrambi gli scenari, la costruzione degli insiemi di training e di test avviene in maniera bilanciata, garantendo un addestramento e una fase di test equilibrati. Infatti, si noti come la frequenza di tweet "bad" (~ 100 tweet bad ogni 1.500 tweet arrivati) risulti essere decisamente superiore a quella di tweet "good" (~ 100 tweet good ogni 30.000 tweet arrivati) e di conseguenza è necessario evitare di sovraddestrare

il modello su tweet “bad”. Tutto ciò non cambia in maniera significativa anche nel caso in cui si decidesse di rimanere in attesa, di un tweet, per più di un’ora.

Infine, prima di poter inviare con successo tali tweet all’engine, è necessario, ovviamente, recuperare le feature che li descrivono. A tale proposito si noti che, se un tweet dovesse contenere media nativi (condivisi con l'interfaccia utente di Twitter anziché tramite un link), sarà presente anche la sezione extended_entities. Infatti, per un qualsiasi media nativo (foto, video o GIF), Twitter ha introdotto un ulteriore campo per la memorizzazione dei metadati. In questo modo, rispetto al campo entities (mantenuto per motivi di compatibilità), potranno essere memorizzate fino a quattro foto e con metadati più dettagliati.

4.6 Consumatore

4.6.1 Snippet di codice riguardante l’engine di elaborazione

```
…

# Definisce le informazioni sull'applicazione (modalità di esecuzione “standalone”,
# e il nome dell’applicazione).
# Nota: l'istanza SparkContext non è supportata per la condivisione tra più
# processi e PySpark non garantisce l'esecuzione multi-processing.
# E’ quindi necessario utilizzare i thread per elaborazioni simultanee.
sc = SparkContext("local", appName="Streaming-Application")

# Definizione dello Streaming Context (entry point principale per tutte le
# funzionalità di streaming) con un intervallo di lettura, dal producer, di 1 secondo.
ssc = StreamingContext(sc, 1)

# Impostazione della directory in cui salvare i checkpoint.
# (Possibili salvataggi: HDFS, DB)
ssc.checkpoint("check")
```

```
# Creazione dello stream necessario per una corretta lettura delle informazioni
# prodotte  dal producer. I parametri in ingresso sono: l'istanza dello Streaming
# Context, il topic (descritto nel paragrafo 4.4) nel quale il producer pubblica le
# informazioni ricevute e la porta sulla quale è in esecuzione Kafka.
kafkaStream = KafkaUtils.createDirectStream(ssc, ["artTopic"], \
                    {"metadata.broker.list": "localhost:9092" })

# Salvataggio delle informazioni minime (id e timestamp) caratterizzanti i tweet che
# soddisfano la fase di filtraggio (tweet originale e follower compresi tra 1.000 e
# 30.000). Per ulteriori informazioni si rimanda al paragrafo 4.6.2.
kafkaStream.filter( lambda x: filterTweet(x[1])) \
                     .map(lambda x: fieldToSave(x[1])) \
                    .map(lambda x : saveOnFile(x,"arrivedTweet.txt","a+")) \
                   .pprint()

# Istanziazione del modello di machine learning.
# Per ulteriori informazioni si rimanda al paragrafo 4.6.2.
model = StreamingLogisticRegressionWithSGD()
model.setInitialWeights([0.0,0.0,0.0,0.0,0.0,0.0])

# Generazione di un DStream per il training del modello.
trainingData = kafkaStream.filter( lambda x: filterTrain(x[1])) \
                                  .map(lambda x: x[1]) \
                                 .map(setLabel)

# Operazione per dare inizio alla computazione.
trainingData.pprint()

# Generazione di un DStream per la fase di test del modello.
testData = kafkaStream.filter( lambda x: filterTest(x[1])) \
                              .map(lambda x: x[1]) .map(setLabel)
```

 # Operazione per dare inizio alla computazione.

testData.pprint()

 # Allenamento del modello sul training set.

model.trainOn(trainingData)

 # Esecuzione delle previsioni sul test set.

prevision = model.predictOnValues(testData.map(lambda lp: (lp.label, lp.features)))

*# **Operazione per dare inizio alla computazione e salvare le previsioni effettuate.***

prevision.map(savePrevision).pprint()

 # Inizio della computazione.

ssc.start()

 # Attende il termine della computazione.

ssc.awaitTermination()

4.6.2 Spiegazione del codice

Operazioni eseguite su kafkaStream:

Sullo stream in ingresso vengono eseguite diverse trasformazioni, al fine di filtrare e poi salvare i tweet che soddisfano i parametri impostati (tweet originale, follower compresi tra 1.000 e 30.000).

Le trasformazioni effettuate sono:

- filter: genera un nuovo DStream selezionando solo i record del DStream sorgente per cui la funzione *filterTweet()* ritorna True. Tale funzione riceve in ingresso la tipologia, l'id e il numero di follower dell'autore e ritorna True se il tweet in analisi soddisfa i parametri discussi nei paragrafi precedenti (tweet originale e follower compresi tra 1.000 e 30.000);

- map: ogni elemento del DStream, ottenuto dalla fase di filtraggio, è elaborato prima dalla funzione *fieldToSave()* e poi dalla funzione *saveOnFile()*. Il primo metodo riceve in ingresso la tipologia, l'id e il numero di follower dell'autore, ritornando i campi da memorizzare su file (id e timestamp). Mentre con il secondo metodo si scrivono tali campi su un apposito file (es.: *arrivedTweet.txt*);

- pprint(): operazione eseguita sul DStream, la cui esecuzione è fondamentale per dare inizio alla computazione. Infatti, si noti come Apache Spark si basi sul concetto di valutazione lazy, la quale, come il nome stesso suggerisce, prevede che l'esecuzione delle trasformazioni definite, non inizi fino a quando non verrà innescata esplicitamente un'azione. Tale approccio assume un ruolo fondamentale nell'ottimizzazione delle performance in ambienti cluster reali.

Classe StreamingLogisticRegressionWithSGD():

Questa classe eredita dalla classe *StreamingLinearAlgorithm,* la quale definisce i metodi *predictOn()* e *predictOnValues()* comuni a tutte le classi che definiscono modelli del cosiddetto "online machine learning".

La fase di training avviene secondo la discesa stocastica del gradiente (SGD), utile per costruire e aggiornare il modello all'arrivo di dati nuovi dal DStream. Inoltre, si noti come questo metodo iterativo sia ampiamente utilizzato per l'allenamento di una vasta tipologia di modelli probabilistici e di apprendimento automatico.

E' sempre necessario fornire un vettore di pesi iniziali con i quali inizializzare il modello e fornire stream di dati con un numero di feature costanti.

Infine il modello può opzionalmente ricevere in ingresso i seguenti parametri:

- stepSize (default: 0.1): meglio noto come "learning rate". Rappresenta un iperparametro che determina in che misura le nuove informazioni acquisite sovrascrivono quelle vecchie. In generale tale valore è sempre minore di 1, altrimenti verrà meno la convergenza dell'apprendimento;

- numIterations (default: 50): iperparametro che specifica il numero di volte che il training set corrente viene utilizzato per l'aggiornamento dei pesi del modello. Ovviamente, nel caso in cui si dovesse raggiungere la convergenza impostata prima del raggiungimento dal valore assunto da tale parametro, le iterazioni effettuate durante la fase di training saranno interrotte;

- miniBatchFraction (default: 1.0): iperparametro che rappresenta la frazione dell'insieme di dati da utilizzare per l'aggiornamento del modello;

- regParam (default: 0.0): parametro di regolarizzazione L2 per ridurre il rischio di overfitting. Risulta essere importante in scenari in cui il training set non ha grandi dimensioni;

- convergenceTol (default: 0.001): valore utilizzato per determinare la convergenza del modello e di conseguenza decidere quando terminare le iterazioni.

DStream trainingData e testData:

Innanzitutto, si ricordi come il produttore descritto nei paragrafi precedenti produca le seguenti tre differenti categorie di stream:

- stream necessario per la fase di filtraggio iniziale e contenente la tipologia del tweet, l'id e il numero di follower dell'autore del tweet;

- trainingData: stream necessario per la fase di training del modello e contenente informazioni sul numero di retweet, di url, di tag, di hashtag, di media, la lunghezza del testo e la stringa "TRAIN";

- testData: stream necessario per la fase di test del modello e contenente informazioni sul numero di retweet, di url, di tag, di hashtag, di media, la lunghezza del testo e la stringa "TEST".

Per questi ultimi due tipi di stream si eseguono le seguenti trasformazioni:

- filter: genera un nuovo DStream, il quale a seconda dello stream a cui è applicata conterrà solamente dati di train o di test;

- map: ogni elemento del DStream ottenuto dalla fase di filtraggio è elaborato dalla funzione *setLabel()*. Quest'ultima assegna ad ogni tweet una label, il cui valore è 0.0 nel caso in cui il tweet sia stato classificato come BAD, altrimenti 1.0.

4.6.3 Snippet di codice riguardante le funzioni invocate dalle trasformazioni eseguite dall'engine

```
# Metodo per ottenere un DStream di tweet originali e i cui autori hanno un numero
di follower compresi tra 1.000 e 30.000.
def filterTweet (inputTweet):

    # Verifica che non sia lo stream utilizzato per la fase di training o quella di test.
    if ("TRAIN" not in inputTweet and "TEST" not in inputTweet) :

        fields = inputTweet.split(" ")

        # Verifica che sia un tweet originale, scritto da un autore con un numero
        # di follower compreso tra 1.000 e 30.000.
        if 'True' in fields[0] and int(fields[2])>=1000 and int(fields[2])<=30000:
            return True

    return False
```

```
# Metodo che elabora il DStream di tweet, ottenuti dal filtraggio precedente,
# generando un DStream i cui elementi contengono le informazioni minime di cui
# si è interessati a memorizzare (id, timestamp).
def fieldToSave(inputTweet):

    fields = inputTweet.split(" ")

    info = fields[1] + " " + time.strftime("%Y-%m-%d %H:%M:%S", \
                                           time.gmtime())+"\n"
    return info

...

# Metodo per ottenere il DStream utilizzato per la fase di training del modello.
def filterTrain(inputTweet):
    return True if "TRAIN" in inputTweet else False

# Metodo per ottenere il DStream utilizzato per la fase di test del modello.
def filterTest(inputTweet):
    return True if "TEST" in inputTweet else False

# Metodo per assegnare ad un tweet l'opportuna label.
def setLabel(dataML):

    # Label da assegnare a tweet BAD.
    label = 0.0

    # Label da assegnare a tweet GOOD.
    if "GOOD" in dataML:
        label = 1.0
```

```
fields = dataML.split(" ")

# Parametri in ingresso: label, [numero di retweet, lunghezza del testo, numero
# di url,  numero di tag, numero di media, numero di hashtag].
return LabeledPoint(label, [float(fields[2]),  float(fields[3]), float(fields[4]), \
                            float(fields[5]), float(fields[6]), float(fields[7])])
```

Capitolo V

"Criteri di valutazione"

Lo scopo del presente capitolo è di fornire una visione dei risultati ottenuti con i parametri introdotti nel capitolo precedente e che caratterizzano l'applicazione in analisi.

In particolare, tali risultati vengono esposti tramite opportune metriche di valutazione al fine di analizzare in maniera più accurata il comportamento del sistema.

5.1 Metriche di valutazione

Esistono diverse metriche di valutazione per i modelli di classificazione binaria, ognuna delle quali descrive un aspetto diverso del modello previsionale. Tra le principali metriche si ricordano:

- Accuratezza: esprime la percentuale di previsioni classificate correttamente e solitamente rappresenta la prima metrica che viene osservata durante le valutazioni. Affinché tale valore mostri realmente l'efficacia del classificatore è fondamentale che i dati presenti nell'insieme di test siano equamente bilanciati tra le due classi (positivi e negativi). Infine, bisogna tenere ben presente che, a seconda degli scenari applicativi, anche un'accuratezza del 99%, può non essere necessariamente considerata come ottima, in quanto quell'1% di errore può avere conseguenze molto diverse. Esempio: applicazioni di rilevamento frodi.

$$\text{Accuratezza} = \frac{TP + TN}{TP + FP + TN + FN} =$$

$$= \frac{somma\ degli\ elementi\ veramente\ positivi\ e\ negativi}{tutti\ gli\ elementi}$$

- Precisione: misura la percentuale di elementi positivi classificati correttamente. Si conferisce maggiore importanza a tale metrica nei casi in cui si voglia avere una maggiore certezza che tutti i veri positivi vengano effettivamente classificati come positivi, anche a patto di avere, in tale classificazione, alcuni falsi positivi, piuttosto invece di correre il rischio che qualche vero positivo possa essere classificato come negativo. Esempio: si preferisce avere alcune email di spam nella casella di posta in arrivo, piuttosto che alcune email regolari nella casella di spam.

$$\text{Precisione} = \frac{TP}{TP + FP} = \frac{elementi\ veramente\ positivi}{tutti\ gli\ elementi\ etichettati\ come\ positivi}$$

- Recall (sensibilità): misura la percentuale di elementi effettivamente positivi dato l'insieme di elementi classificati come positivi. In altre parole, ciò equivale a chiedersi quanti elementi positivi sono stati predetti, tra tutti gli elementi positivi presenti. Si conferisce maggiore importanza a tale metrica nei casi in cui l'idea di avere dei falsi positivi sia comunque migliore di avere dei falsi negativi, e quindi se il verificarsi di questi ultimi risulti essere inaccettabile per il sistema. Esempio: si preferisce avere delle persone sane etichettate come malate, piuttosto che etichettare una persona malata come sana.

$$\text{Recall} = \frac{TP}{TP + FN} = \frac{elementi\ classificati\ correttamente\ come\ positivi}{tutti\ gli\ elementi\ positivi}$$

- Score F1: rappresenta una misura che tiene conto sia della precisione che della recall riassunte da un unico valore dato dalla media armonica delle due misure. Tuttavia, è sempre opportuno tenere conto anche dei singoli valori della precisione e della recall per comprendere meglio il comportamento del classificatore. Risulta essere un indicatore particolarmente valido per distribuzioni non uniformi.

$$\text{F1} = 2 \times \frac{precisione \times recall}{precisione + recall}$$

- Curva receiver operating characteristic (ROC): è una tecnica statistica utilizzata per visualizzare, organizzare e valutare le performance di un modello di classificazione binario. Solitamente si ha una soglia *t* per decidere se classificare un elemento come positivo o come negativo.

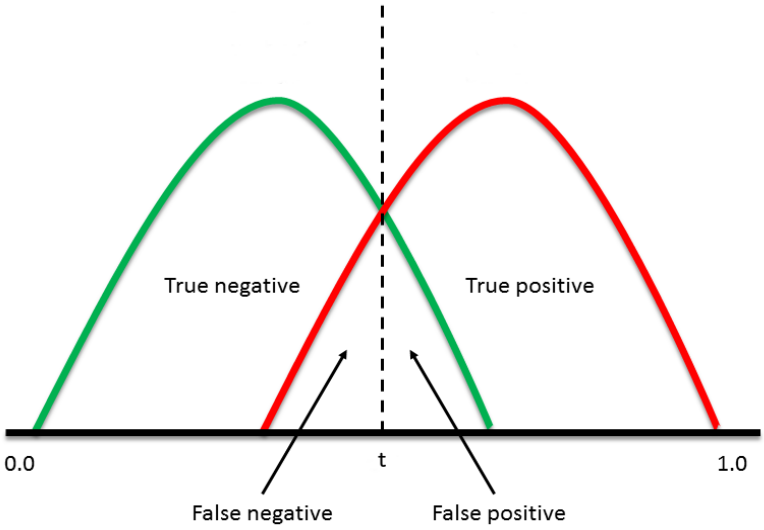

Grafico 5.1 - Distribuzione degli esiti in uno scenario reale.

Si noti come una distribuzione ideale prevede che le due curve non siano sovrapposte.

Tale tecnica può essere utilizzata per confrontare curve ROC aventi soglie differenti o comunque per avere informazioni sulle caratteristiche del singolo modello in analisi.

In quest'ultimo caso è più corretto parlare di spazio ROC. In entrambi i casi è noto che, quanto più i punti della curva saranno vicini all'angolo superiore sinistro tanto più le prestazioni del classificatore saranno migliori. Al contrario, più i punti della curva si troveranno vicino alla diagonale tanto più il classificatore tenderà a generare delle stime al limite della casualità.

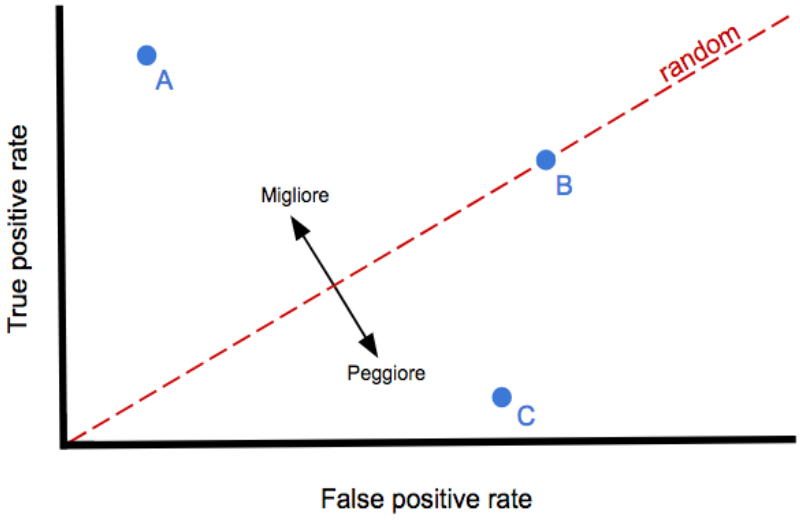

Grafico 5.2 - Spazio ROC.

Si noti che il "false positive rate" (FPR) rappresenta la probabilità di avere dei falsi positivi (FPR = $\frac{FP}{FP + TP}$).

- Matrice di confusione: delinea l'accuratezza della classificazione statistica, fissata una soglia, mostrando il numero di previsioni positive esatte, negative esatte, positive errate e negative errate. Di conseguenza, implicitamente, fornisce informazioni sulla precisione, la recall e l'accuratezza.

Valori reali / valori predetti	**Positivi**	**Negativi**	
Positivi	TP	FP	$\textbf{\textit{precisione}} = \frac{TP}{TP + FP}$
Negativi	FN	TN	$\textbf{\textit{F1}} = 2 \times \frac{precisione \times recall}{precisione + recall}$
	$\textbf{\textit{recall}} = \frac{TP}{TP + FN}$		$\textbf{\textit{accuratezza}} = \frac{TP + TN}{TP + FP + TN + FN}$

Tabella 5.1 - Esempio della struttura di una matrice di confusione.

Si noti che i termini “positivo” e “negativo” si riferiscono alla previsione fornita dal modello, mentre i termini “vero” e “falso” fanno riferimento alla correttezza di queste risposte rispetto alle risposte esatte.

5.2 Valutazione del modello costruito

Overview dei parametri scelti:

Parametri	
Termini della tabella 4.1	Analizzati nel paragrafo 4.5.2
Follower >= 1.000 e Follower <= 30.000	Analizzati nel paragrafo 4.5.4
Tempo di attesa 60 minuti	Analizzati nel paragrafo 4.5.6
Like <=3 e Like >= 15	Analizzati nel paragrafo 4.5.6

Tabella 5.2 - Parametri scelti per la costruzione del modello.

Valutazioni:

Valori reali / valori predetti	**Good**	**Bad**	
1° iterazione (diurna)			
Good	TP = 4	FN = 8	*precisione = 33 %*
Bad	FP = 6	TN = 2	*F1 = 36 %*
	recall = 40 %		*accuratezza = 30 %*
2° iterazione (notturna)			
Good	TP = 8	FN = 10	*precisione = 44 %*
Bad	FP = 2	TN = 0	*F1 = 57 %*
	recall = 80 %		*accuratezza = 40 %*

3° iterazione (diurna)			
Good	TP = 0	FN = 3	*precisione = 0 %*
Bad	FP = 9	TN = 8	*F1 = n/a*
	recall = 0 %		*accuratezza = 40 %*
4° iterazione (notturna)			
Good	TP = 4	FN = 4	*precisione = 50 %*
Bad	FP = 6	TN = 6	*F1 = 44 %*
	recall = 40 %		*accuratezza = 50 %*
5° iterazione (diurna)			
Good	TP = 8	FN = 10	*precisione = 44 %*
Bad	FP = 2	TN = 0	*F1 = 57 %*
	recall = 80 %		*accuratezza = 40 %*
6° iterazione (notturna)			
Good	TP = 6	FN = 4	*precisione = 60 %*
Bad	FP = 4	TN = 6	*F1 = 60 %*
	recall = 60 %		*accuratezza = 60 %*
7° iterazione (diurna)			
Good	TP = 8	FN = 0	*precisione = 100 %*
Bad	FP = 2	TN = 10	*F1 = 89 %*
	recall = 80 %		*accuratezza = 90 %*
8° iterazione (notturna)			
Good	TP = 5	FN = 2	*precisione = 71 %*
Bad	FP = 5	TN = 8	*F1 = 59 %*
	recall = 50 %		*accuratezza = 65 %*

9° iterazione (diurna)			
Good	TP = 6	FN = 0	*precisione = 100 %*
Bad	FP =4	TN = 10	*F1 = 75 %*
	recall = 60 %		*accuratezza = 80 %*
10° iterazione (notturna)			
Good	TP = 9	FN = 3	*precisione = 75 %*
Bad	FP = 1	TN = 7	*F1 = 82 %*
	recall = 90 %		*accuratezza = 80 %*
11° iterazione (diurna)			
Good	TP = 10	FN = 10	*precisione = 50 %*
Bad	FP = 0	TN = 0	*F1 = 67 %*
	recall = 100 %		*accuratezza = 50 %*
12° iterazione (notturna)			
Good	TP = 9	FN = 0	*precisione = 100 %*
Bad	FP = 2	TN = 9	*F1 = 90 %*
	recall = 82 %		*accuratezza = 90 %*
13° iterazione (diurna)			
Good	TP = 10	FN = 1	*precisione = 91 %*
Bad	FP = 0	TN = 9	*F1 = 95 %*
	recall = 100 %		*accuratezza = 95 %*
14° iterazione (notturna)			
Good	TP = 10	FN = 10	*precisione = 50 %*
Bad	FP = 0	TN = 0	*F1 = 67 %*
	recall = 100 %		*accuratezza = 50 %*

15° iterazione (diurna)			
Good	TP = 10	FN = 0	*precisione = 100 %*
Bad	FP = 0	TN = 10	*F1 = 100 %*
	recall = 100 %		*accuratezza = 100%*
16° iterazione (notturna)			
Good	TP = 10	FN = 2	*precisione = 83 %*
Bad	FP = 0	TN = 8	*F1 = 91 %*
	recall = 100 %		*accuratezza = 90 %*
17° iterazione (diurna)			
Good	TP = 9	FN = 0	*precisione = 100 %*
Bad	FP = 1	TN = 10	*F1 = 95 %*
	recall = 90 %		*accuratezza = 95 %*
18° iterazione (notturna)			
Good	TP = 10	FN = 10	*precisione = 50 %*
Bad	FP = 0	TN = 0	*F1 = 67 %*
	recall = 100 %		*accuratezza = 50 %*
19° iterazione (diurna)			
Good	TP = 10	FN = 10	*precisione = 50 %*
Bad	FP = 0	TN = 0	*F1 = 67 %*
	recall = 100 %		*accuratezza = 50 %*
20° iterazione (diurna)			
Good	TP = 1	FN = 4	*precisione = 25 %*
Bad	FP = 9	TN = 6	*F1 = 14 %*
	recall = 10 %		*accuratezza = 35 %*

21° iterazione (diurna)			
Good	TP = 7	FN = 1	*precisione = 88 %*
Bad	FP = 3	TN = 9	*F1 = 78 %*
	recall = 70 %		*accuratezza = 80 %*
22° iterazione (notturna)			
Good	TP = 9	FN = 0	*precisione = 100 %*
Bad	FP = 1	TN = 10	*F1 = 95 %*
	recall = 90 %		*accuratezza = 95 %*

Tabella 5.3 - Matrice di confusione con valori di recall, precisione, score F1 e accuratezza.

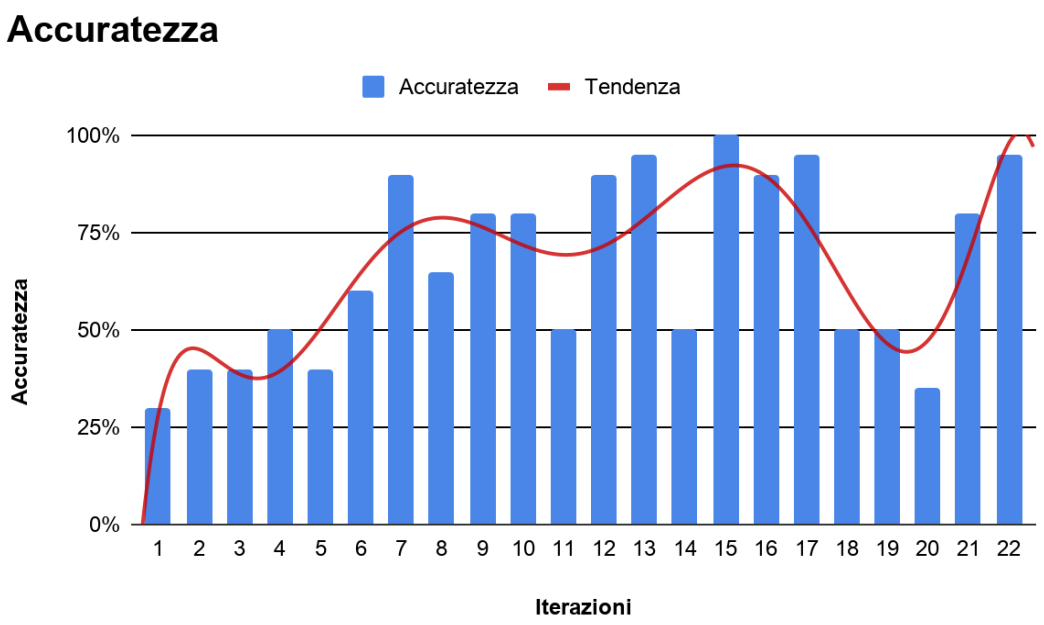

Grafico 5.3 - Andamento dell'accuratezza all'aumentare delle iterazioni.

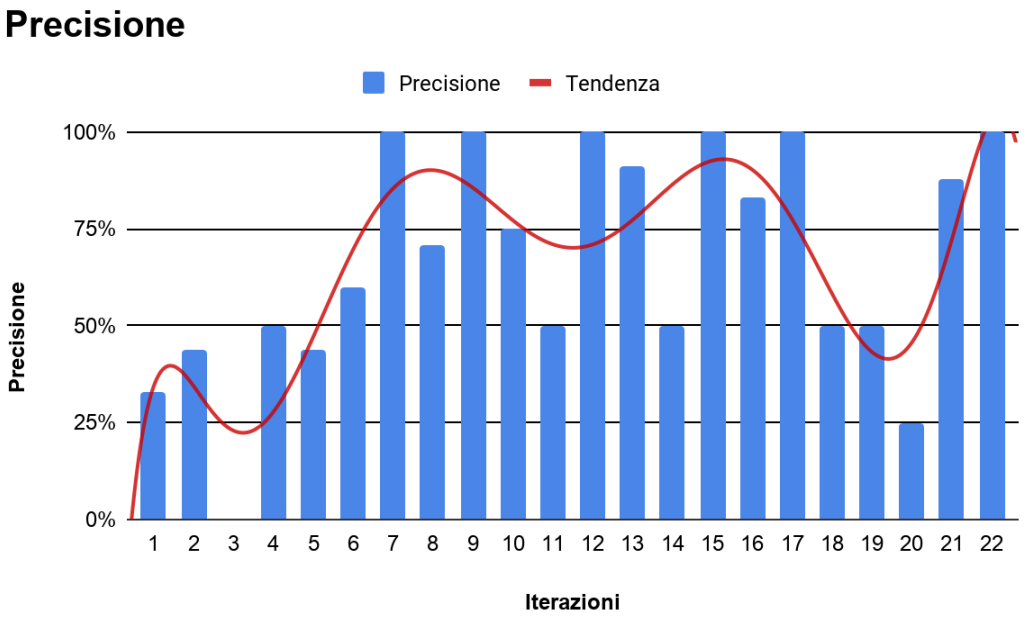

Grafico 5.4 - Andamento della precisione all'aumentare delle iterazioni

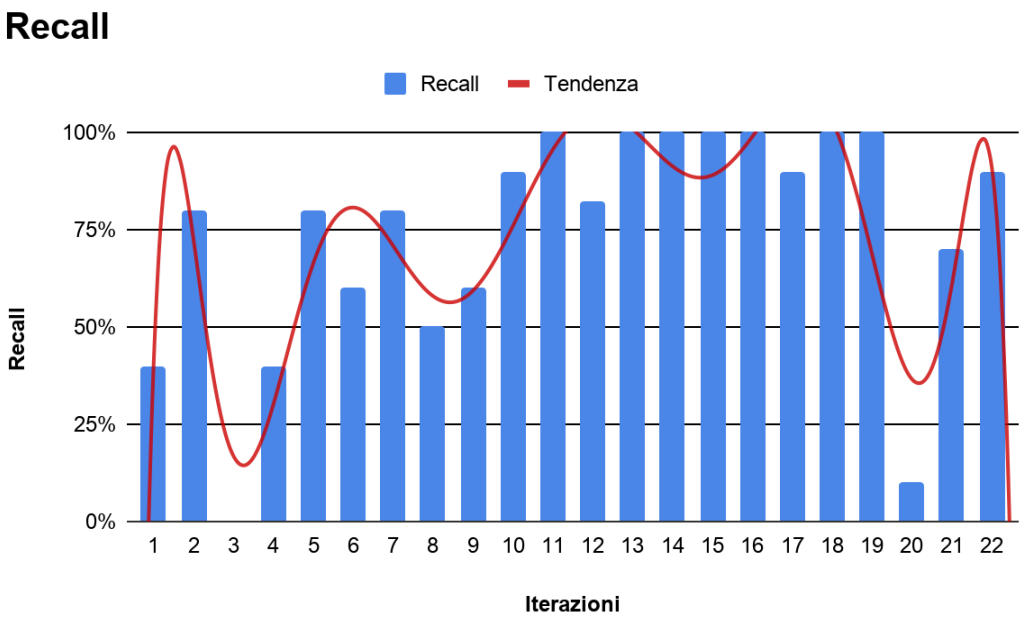

Grafico 5.5 - Andamento della recall all'aumentare delle iterazioni.

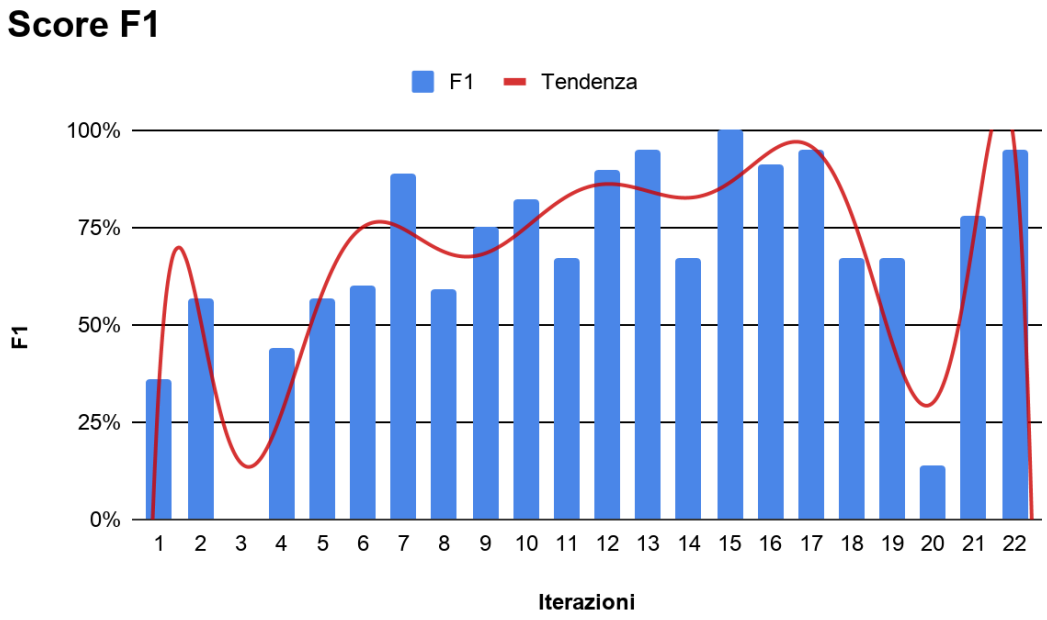

Grafico 5.6 - Andamento dello score F1 all'aumentare delle iterazioni.

5.3 Considerazioni

Innanzitutto, è opportuno evidenziare come le misurazioni siano avvenute nell'arco di circa una settimana, questo in quanto, come già descritto nel capitolo precedente, per avere un insieme di training e di test equamente bilanciato occorre attendere, in media, circa 10 ore.

Inoltre, dai grafici sopra riportati, è possibile notare come, man mano che le iterazioni di training aumentino, le caratteristiche del modello previsionale tendano a migliorare, cercando di assestarsi attorno ad un valore medio.

Tuttavia, dai grafici si può notare come dalla 18° iterazione in poi, i valori di accuratezza, precisione e recall, attorno al quale il modello, dalla 7° iterazione, stava iniziando a stabilizzarsi, hanno subìto un cambiamento improvviso.

Il fenomeno alla base di tale cambiamento improvviso, peraltro osservato in maniera completamente fortuita, ha evidenziato, nel migliore dei modi, le caratteristiche chiave e i vantaggi - capacità di adattamento rapido a scenari in cui i comportamenti subiscono cambiamenti improvvisi - del cosiddetto "online machine learning.

In particolare, il cambiamento in questione, si è verificato il 2 maggio 2019, giorno della ricorrenza del cinquecentesimo anniversario della morte di Leonardo da Vinci (termine presente nella tabella 4.1).

Di conseguenza, il modello previsionale, che stava iniziando a trovare un giusto bilanciamento interno basato sulle caratteristiche intrinseche degli autori di nicchia - che solitamente pubblicano tweet inerenti l'arte - al verificarsi di tale cambiamento ha subìto inizialmente un peggioramento in termini di accuratezza, precisione e recall (iterazione 18, 19 e 20) per poi riuscire ad adattarsi nuovamente alla situazione circostante.

Il primo indicatore che ha evidenziato il cambiamento dello scenario, fino a quel momento analizzato, è stato l'input rate. In particolare, tale valore è raddoppiato rispetto i giorni precedenti.

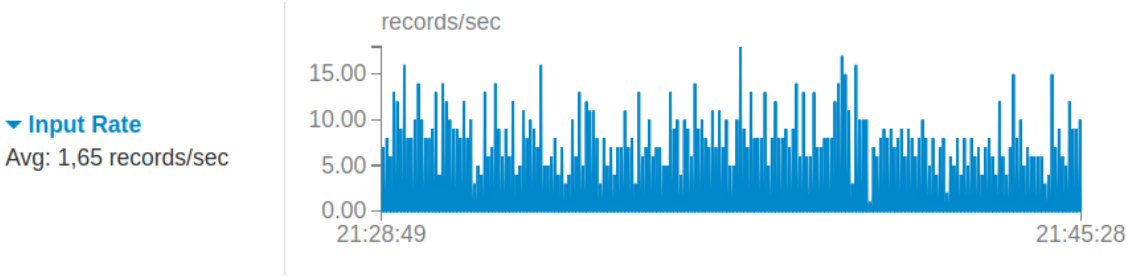

Grafico 5.7 - Input rate 2 maggio 2019.

Inoltre si noti come, durante il corso di questa giornata (2 maggio 2019), sia aumentata anche la frequenza di tweet good, con una conseguente riduzione del tempo occorrente per la costruzione dell'insieme necessario per l'esecuzione di una singola iterazione di aggiornamento del modello. A tale proposito, si noti come le iterazioni dalla 18esima alla 22esima siano state eseguite nell'arco di 30 ore, invece che in 50 ore.

A dimostrazione di tutto ciò si analizza lo spazio ROC, il quale riassume, nei migliori dei modi, quanto appena descritto (grafico 5.8).

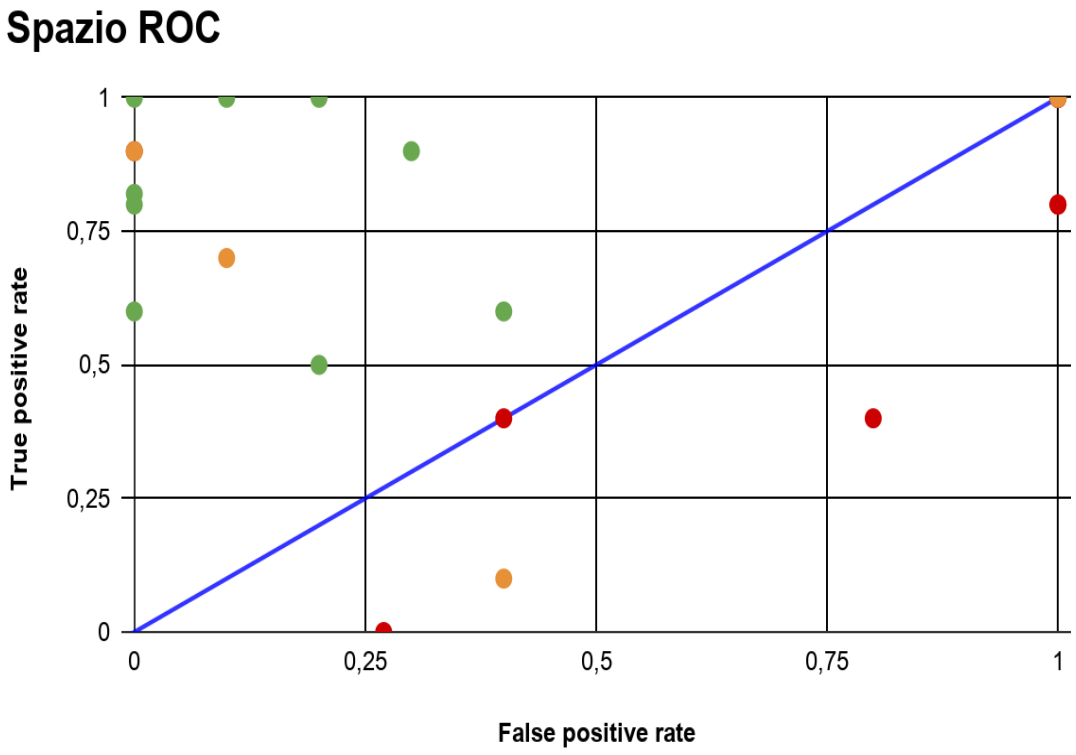

Grafico 5.8 - Spazio ROC.

Da questo grafico è possibile notare come:

- durante le prime sei iterazioni di aggiornamento del modello (punti in rosso) le performance erano praticamente casuali;
- dalla settima iterazione fino alla diciassettesima il modello stava iniziando ad avere performance sempre migliori (punti in verde), avvicinandosi al punto di coordinate (0,1), il quale rappresenta una situazione di classificazione perfetta;
- durante la diciottesima e la diciannovesima iterazione, visto il cambiamento dello scenario, il modello ha assunto nuovamente performance predittive casuali (punti in giallo su/sotto la diagonale);
- dalla ventesima iterazione in poi, il modello è tornato ad avere performance predittive man mano sempre migliori (punti in giallo sopra la diagonale).

Conclusioni

Si conclude qui la presente trattazione prendendo in esame, in ultima istanza, alcune considerazioni emerse durante lo studio e l'approfondimento dei framework - che caratterizzano il panorama dei Big Data - con il fine di riuscire a delineare, nella maniera più accurata possibile, i risultati ottenuti in seguito all'effettiva realizzazione degli obiettivi precedentemente posti.

Innanzitutto, nella prima parte di tale elaborato, grazie alla consultazione di numerose pubblicazioni e articoli scientifici inerenti all'argomento, è emersa l'esistenza di una gamma di diverse alternative, dipendenti rispettivamente dall'ambito preso in considerazione, utili al fine di processare, memorizzare e analizzare i Big Data, motivo per cui, in primo luogo, risulta essere sempre opportuno affrontare preliminarmente uno studio riguardante la realtà da modellare, per comprendere al meglio i tool dei quali si può usufruire.

In secondo luogo, specificamente nella seconda parte del lavoro della presente tesi, l'oggetto della trattazione è risultato essere un approfondimento del framework Apache Spark, il quale tramite la progettazione di un'applicazione - in grado di elaborare efficientemente stream di dati, in real-time, prodotti da indistinti utenti del social network Twitter - ha permesso una concreta realizzazione di alcuni tra i concetti teorici descritti nella prima parte della tesi.

Infine, in base a tali studi, è stato possibile sia costruire un modello previsionale in grado di evolvere nel tempo che verificare, sperimentalmente, le effettive capacità di quest'ultimo.

Parte III

Appendice

Configurazioni

Apache Kafka

Step 1 - installazione:

1) > *wget http://mirror.nohup.it/apache/kafka/2.1.1/kafka_2.11-2.1.1.tgz*

2) > *tar -xzf kafka_2.11-2.1.1.tgz*

3) > *sudo apt-get install zookeeperd*

In alternativa, a quest'ultimo comando, è possibile utilizzare lo script all'interno del package di Kafka per ottenere in modo rapido un'istanza Zookeeper:

> *kafka_2.11-2.1.1/bin/zookeeper-server-start.sh *
kafka_2.11-2.1.1/config/zookeeper.properties.

Step 2 - avvio del server Zookeeper:

Apache Kafka utilizza i servizi offerti da Zookeeper e pertanto è necessario verificare l'attività di tale server, nel seguente modo:

1) > *systemctl status zookeeper*

2) *Nel caso in cui non sia in stato di running eseguire:*

> *systemctl status start zookeeper*

3) *Comando per far sì che Zookeeper si avvii in automatico al boot:*

> *systemctl status enable zookeeper*

Step 3 - avvio del server Kafka:

1) > *kafka_2.11-2.1.1/bin/kafka-server-start.sh *
kafka_2.11-2.1.1/config/server.properties

Step 4 - creazione di un topic:

> *kafka_2.11-2.1.1/bin/kafka-topics.sh --create --zookeeper localhost:2181 *
--replication-factor 1 --partitions 1 --config retention.ms=7200000 --topic artTopic

Per verificare la creazione di un topic:

> *kafka_2.11-2.1.1/bin/kafka-topics.sh --list --zookeeper localhost:2181*

Si noti che è necessario che il server Kafka sia in esecuzione (step 3).

Step 5 - esecuzione del produttore implementato:

> *python3 producer.py*

Si noti che è necessario che il server Kafka sia in esecuzione (step 3).

Apache Spark

Si noti come il supporto per Java 7, Python 2.6 e per tutte le versioni precedenti sono stati rimossi a partire da Apache Spark 2.2.0.

Step 1 - installazione:

1) > *wget https://archive.apache.org/dist/spark/spark-2.4.0/spark-2.4.0-bin-hadoop2.7.tgz*

2) > *tar -xzf spark-2.4.0-bin-hadoop2.7.tgz*

3) > *cd spark-2.4.0-bin-hadoop2.7*

Step 2 - esecuzione del consumatore:

1) > *PYSPARK_PYTHON=python3 bin/spark-submit --packages *
org.apache.spark:spark-streaming-kafka-0-8_2.11:2.2.0 consumer.py

Eventualmente è possibile scaricare il JAR *spark-streaming-kafka-0-8-assembly* da *https://jar-download.com* e aggiungerlo al comando spark-submit tramite l'opzione --jars.

Bibliografia

- Stephen Marsland, *Machine Learning: An Algorithmic Perspective,* Boca Raton, Chapman & Hall/CRC, 2009.
- Mauro Cerasoli, Giuseppe Tomassetti, *Elementi di Statistica, introduzione alla matematica dell'incerto*, Bologna, Zanichelli, 1991.
- Anna M. Gambotto Manzone e Claudia Susara Longo, *Probabilità e statistica 2*, Milano, Tramontana, 1990.

Sitografia

- Mert Onuralp Gökalp, Kerem Kayabay, Mohamed Zaki, Altan Koçyiğit, P. Erhan Eren, and Andy Neely, *Big-Data Analytic Architecture for Businesses: a comprehensive review on new open-source big-data tools*, in: *https://cambridgeservicealliance.eng.cam.ac.uk/resources/Downloads/Monthly%20Papers/2017OctPaperBigDataAnalytics.pdf*
- Sara Landset, Taghi M. Khoshgoftaar, Aaron N. Richter, Tawfiq Hasanin, *A survey of open source tools for machine learning with big data in the Hadoop ecosystem,* in: *https://journalofbigdata.springeropen.com/articles/10.1186s40537-015-0032-1*
- Kapil Bakshi, *Considerations for big data: Architecture and approach*, in: *https://ieeexplore.ieee.org/abstract/document/6187357*
- April Reeve, *Big Data and NoSQL: The Problem with Relational Databases*, in: *https://infocus.dellemc.com/april_reeve/big-data-and-nosql-the-problem-with-relational-databases/*
- Sam Madden, *From Databases to Big Data*, in: *https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6188576*
- *Munvo, The Facts: Hadoop Big Data vs. Relational Databases*, in: *https://munvo.com/2017/07/18/hadoop-big-data-vs-relational-databases/*

- A B M Moniruzzaman e Syed Akhter Hossain, *NoSQL Database: New Era of Databases for Big data Analytics - Classification, Characteristics and Comparison*, in: *https://arxiv.org/pdf/1307.0191.pdf*
- Edwin Anto, *Transitioning From Relational Database to Big Data, https://www.ijraset.com/fileserve.php?FID=5141*
- Simplilearn, *Big Data and Hadoop Ecosystem Tutorial*, in: *https://www.simplilearn.com/big-data-and-hadoop-ecosystem-tutorial*
- Hugj J. Watson, *Tutorial: Big Data Analytics: Concepts, Technologies, and Applications*, in: *https://www.researchgate.net/profile/Hugh_Watson3/publication/263563679_Tutorial_Big_Data_Analytics_Concepts_Technologies_and_Applications/links/576e809608ae10de639a4531/Tutorial-Big-Data-Analytics-Concepts-Technologies-and-Applications.pdf*
- Shiqi Wu, *Big Data processing with Hadoop*, in: https://www.theseus.fi/bitstream/handle/10024/96998Big%20Data%20*Processing %20With%20Hadoop.pdf?sequence=1&isAllowed=y*
- Chandan Prakash, *Spark Streaming vs Flink vs Storm vs Kafka Streams vs Samza: Choose Your Stream Processing Framework*, in: *https://medium.com/@chandanbaranwal/spark-streaming-vs-flink-vs-storm-vs-kafka-streams-vs-samza-choose-your-stream-processing-91ea3f04675b*
- Ahmed Oussous, Fatima-Zahra Benjelloun, AyoubAit Lahcen, Samir Belfkih, *Big Data technologies: A survey*, in : *https://www.sciencedirect.com/science/article/pii/S1319157817300034*
- Gowthamy Vaseekaran, *Big Data Battle : Batch Processing vs Stream Processing*, in : *https://medium.com/@gowthamy/big-data-battle-batch-processing-vs-stream-processing-5d94600d8103*
- Daniel Abadi, Shivnath Babu, Fatma Özcan, Ippokratis Pandis, *Tutorial: SQL-on-Hadoop Systems,* in: *http://www.cs.umd.edu/~abadi/papers/sql-on-hadoop-tutorial.pdf*
- Apache Projects Directory, in: *https://projects.apache.org/projects.html?category#big-data*

- Hao Zhang, Gang Chen, Beng Chin Ooi, Kian-Lee Tan e Meihui Zhang, *In-Memory Big Data Managementand Processing: A Survey*, in *https://www.comp.nus.edu.sg/~memepic/pdfs/mem-survey.pdf*
- *How much data is generated every minute?,* in: *https://web-assets.domo.com/blog/wp-content/uploads/2017/07/17_domo_data-never-sleeps-5-01.png*
- Xu Fei's, *Hadoop Ecosystem*, in: *https://autofei.wordpress.com/2017/07/06/hadoop-ecosystem/*
- Xinhui Tian, Rui Han, Lei Wang, Gang Lu, Jianfeng Zhan, *Latency critical big data computing in finance*, in: *https://www.sciencedirect.com/science/article/pii/S2405918815000045*
- Celeste Duran, *Kerberos & Hadoop: Security Big Data*, in: *https://blog.datatons.com/2017/06/21/kerberos-hadoop-securing-bigdata-1/*
- Andrea Mostosi, *The Big-Data Ecosystem Table*, in: *http://bigdata.andreamostosi.name/*
- Javi Roman, *The Hadoop Ecosystem Table*, in: *https://hadoopecosystemtable.github.io/*
- Awesome Big Data, in: *https://github.com/onurakpolat/awesome-bigdata#frameworks*
- Andreas Steffan, *Why does Scala need JVM and Java to work?,* in: *https://www.quora.com/Why-does-Scala-need-JVM-and-Java-to-work*
- Saravan Chidambaram, Sujoy Saraswati, Ranganath Ramachandra, Jayashree B Huttanagoudar, N Hema, R, Roopalakshmi, *JVM characterization framework for workload generated as per machine learning benckmark and spark framework*, in: *https://ieeexplore.ieee.org/document/7808102/figures#figures*
- Astha S., Why was Hadoop written in Java?, in: *https://www.linkedin.com/pulse/why-hadoop-written-java-astha-srivastava*
- Hadoop Wiki, *JobTracker,* in: *https://wiki.apache.org/hadoop/JobTracker*
- Jason Brownlee, *Overfitting and Underfitting With Machine Learning Algorithms,* in: *https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/*

- Krzysztof J. Cios, Roman W. Swiniarski, Witold Pedrycz e Lukasz A. Kurgan, *Unsupervised Learning: Association Rules*, in: https://link.springer.com/chapter/10.1007%2F978-0-387-36795-8_10
- Amazon Machine Learning, in: *https://docs.aws.amazon.com/it_it/machine-learning/latest/dg/evaluating_models.html*
- Microsoft, *Come valutare le prestazioni del modello in Azure Machine Learning Studio,* in: *https://docs.microsoft.com/it-it/azure/machine-learning/studio/evaluate-model-performance*
- Apache Apex, in: *https://apex.apache.org/docs/apex/*
- Thomas Weise, *From Batch to Streaming ET(L) with Apache Apex*, in: *https://www.slideshare.net/Hadoop_Summit/from-batch-to-streaming-etl-with-apache-apex*
- Mike Wheatley, *Spark rival Apache Apex hits top-level status*, in: *https://siliconangle.com/2016/04/26/spark-rival-apache-apex-hits-top-level-status/*
- Vlad Rozov, *Exactly-Once End-To-End Processing with Apache Apex,* in: *https://www.slideshare.net/ydn/february-2017-hug-exactlyonce-endtoend-processing-with-apache-apex*
- Devendra Tagare, *Dimensions Computation With Apache Apex*, in: *https://events.static.linuxfound.org/sites/events/files/slides/ApexDimensionsComputeDev.pdf*
- Thomas Weise, *Stream Processing with Apache Apex*, in: *http://materials.dagstuhl.de/files/17/17441/17441.ThomasWeise.Slides.pdf*
- Thomas Weise, Apache Apex Fault Tolerance and Processing Semantics, in: *https://www.slideshare.net/ApacheApex/apache-apex-fault-tolerance-and-processing-semantics*
- Szabolcs Feczak, *Google Cloud Dataflow the next generation of managed big data service based on the Apache Beam programming model, https://www.slideshare.net/SzabolcsFeczak/apache-beam-and-google-cloud-dataflow-idg-final-64440998*
- Apache Beam, *Beam SQL overview,* in: *https://beam.apache.org/documentation/dsls/sql/overview/*

- Apache Beam, Built-in I/O Transforms, in: *https://beam.apache.org/documentation/io/built-in/*
- Fabian Hueske, Shaoxuan Wang e Xiaowei Jiang**,** *Continuous Queries on Dynamic Tables,* in: *https://flink.apache.org/news/2017/04/04/dynamic-tables.html*
- Mokabyte, *Stateful Stream Processing con Apache Flink*, in: *http://www.mokabyte.it/2018/06/flink-1/*
- Apache Flink, *What is Apache FLink? - Applications*, in: *https://flink.apache.org/flink-applications.html*
- Apache Flink, *Operator*, in: *https://ci.apache.org/projects/flink/flink-docs-master/dev/stream/operators/*
- Apache Flink, *Flink DataSet API Programming Guide*, in: *https://ci.apache.org/projects/flink/flink-docs-master/dev/batch/*
- Apache Flink, *Table API*, in: *https://ci.apache.org/projects/flink/flink-docs-master/dev/batch/*
- Apache Flink, *SQL*, in: *https://ci.apache.org/projects/flink/flink-docs-master/dev/table/sql.html#group-windows*
- Educba, *Apache Pig vs Apache Hive - Top 12 Useful Differences*, in: *https://www.educba.com/apache-pig-vs-apache-hive/*
- Hortonworks, *Apache Pig*, in: *https://it.hortonworks.com/apache/pig/#section_2*
- Samza, *Core concepts,* in: *http://samza.apache.org/learn/documentation/1.0.0/core-concepts/core-concepts.html*
- Srinivasulu Punuru, *Samza SQL*, in: *https://www.slideshare.net/SamarthShetty2/stream-processing-using-samza-sql*
- Kay Ewbank, *Apache Samza Adds SQL*, in: *https://www.i-programmer.info/*news/*197-data-mining/11445-apache-samza-adds-sql.html*
- Samza, *Samza SQL*, in: *https://samza.apache.org/learn/documentation/latest/api/samza-sql.html*
- Samza, *Windowing,* in: *https://samza.apache.org/learn/documentation/0.7.0/container/windowing.html*

- Samza, *Run on YARN,* in: *https://samza.apache.org/learn/documentation/latest/deployment/yarn.html*
- Apache Spark, *Classification and Regression - RDD-based-API*, in: *https://spark.apache.org/docs/latest/mllib-classification-regression.html*
- Apache Spark, *Spark Streaming Programming Guide,* in: *https://spark.apache.org/docs/latest/streaming-programming-guide.html*
- Apache Spark, *Structured Streaming Programming Guide*, in: https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#continuous-processing
- Apache Spark, *Spark SQL, DataFrames and Datasets Guide*, in: https://spark.apache.org/docs/latest/sql-programming-guide.html
- Hortonworks, *Data Processing API in Apache Tez,* in: *https://it.hortonworks.com/blog/expressing-data-processing-in-apache-tez/*
- Roopesh Shenoy, *What is Apache Tez?, https://www.infoq.com/articles/apache-tez-saha-murthy*
- Hortonworks, *Tez Data Processing over YARN, https://www.slideshare.net/Tech_InMobi/tez-hadoopmeetupv3*
- Bikas Saha, Hitesh Shah, Siddharth Seth, Gopal Vijayaraghavan, Arun Murthy, Carlo Curino, *Apache Tez: A Unifying Framework for Modeling and Building Data Processing Applications*, in: *http://web.eecs.umich.edu/~mosharaf/Readings/Tez.pdf*
- Amit Verma, *Apache Storm Vs. Apache Spark [Comparison]*, in: *https://www.whizlabs.com/blog/apache-storm-vs-apache-spark/*
- Apache Storm, *Windowing Support in Core Storm*, in: *http://storm.apache.org/releases/1.0.6/Windowing.html*
- Tyler Akidau, Alex Balikov, Kaya Bekiroglu, Slava Chernyak, Josh Haberman, Reuven Lax, Sam McVeety, Daniel Mills, Paul Nordstrom, Sam Whittle, *MillWheel: Fault-Tolerant Stream Processing at Internet Scale*, in: *https://storage.googleapis.com/pub-tools-public-publication-data/pdf/41378.pdf*
- Michael Whittaker, *MillWheel: Fault-Tolerant Stream Processing at Internet Scale*, in: *https://mwhittaker.github.io/papers/html/akidau2013millwheel.html*

- Google Developers, *Cloud Platform at Google I/O - new Big Data, Mobile and Monitoring products,* in: *https://developers.googleblog.com/2014/06/cloud-platform-at-google-io-new-big.html*
- Google Cloud, *Cloud dataflow*, in: *https://cloud.google.com/dataflow/*
- Google Cloud, *Cloud Dataflow security and permissions*, in: *https://cloud.google.com/dataflow/docs/concepts/security-and-permissions*
- Google Cloud, *Exploratory queries with Google BigQuery*, in: *https://cloud.google.com/community/tutorials/data-science-exploration*
- Matt Lang, *Managing streaming pipelines in Google Cloud Dataflow just got better,* in: *https://cloud.google.com/blog/products/gcp/managing-streaming-pipelines-in-google-cloud-dataflow-just-got-better*
- Cosmin Arad, *Google Cloud Dataflow*, in: *http://ictlabs-summer-school.sics.se/2015/slides/google%20cloud%20dataflow.pdf*
- Jelena Pjesivac-Grbovic, *Data Processing with Apache Beam (incubating) and Google Cloud Dataflow*, in: *https://www-conf.slac.stanford.edu/xldb2016/talks/published/Weds_4_Pjesivac-Grbovic_Data_Processing_With_Apache_Beam.pdf*
- Alex Van Boxel, Cloud Dataflow (A Google Service and SDK), in: *https://www.slideshare.net/AlexVanBoxel/google-cloud-dataflow-58499530*
- Christopher Alghini, *Your Introduction to Google Cloud's Dataflow Model,* in: *https://www.coolheadtech.com/blog/your-introduction-to-google-clouds-dataflow-model*
- Gord Sissons, *10 reasons to love IBM InfoSphere BigInsights for Hadoop,* in: *https://www.ibmbigdatahub.com/blog/10-reasons-love-ibm-infosphere-biginsights-hadoop*
- Dakshi Agrawal, *Fault tolerance by IBM Streams (aka exactly-once streaming analytics),* in: *https://developer.ibm.com/streamsdev/2015/06/24/fault-tolerance-ibm-streams-aka-exactly-streaming-analytics/*
- Roch Archambault e Richard King, *IBM® InfoSphere® Streams v3.0 Performance Report*, in: *https://public.dhe.ibm.com/software/data/sw-library/infosphere/whitepapers/InfoSphere_Streams_Performance_wp.pdf*

- IBM, *Security objects and access permissions for InfoSphere Streams domain and instances*, in: *https://www.ibm.com/support/knowledgecenter/en/SSCRJU_4.0.1/com.ibm.streams.cfg.doc/doc/ibminfospherestreams-security-objects.html*
- Chuck Ballard, Andy Frenkiel, Bugra Gedik, Roger Rea, Mike Spicer, Vitali N. Zoubov, Kevin Forester, Senthil Nathan, Deepak Rajan, Brian Williams, Michael P. Koranda, *IBM InfoSphere Streams (Assembling Continuous Insight in the Information Revolution),* in: *https://books.google.it/books?id=ZqzEAgAAQBAJ&pg=PA354&lpg=PA354&dq=IBM+InfoSphere+Streams+documentation&source=bl&ots=3nHTwmpdX6&sig=ACfU3U2fY5b5K-EXqmww9tW8W_tluN12Vg&hl=it&sa=X&ved=2ahUKEwixs8CV-4jhAhVCqaQKHQi5DbM4ChDoATACegQIBBAB#v=onepage&q=IBM%20InfoSphere%20Streams%20documentation&f=true*
- IBM, *Developing and running applications that use the HDFS Toolkit*, in: *https://www.ibm.com/support/knowledgecenter/en/SSCRJU_4.0.0/com.ibm.streams.toolkits.doc/doc/tk$com.ibm.streamsx.hdfs/tk$com.ibm.streamsx.hdfs$1.html*
- Karthik Ramasamy, *Flying faster with Twitter Heron,* https://blog.twitter.com/engineering/en_us/a/2015/flying-faster-with-twitter-heron.html
- Karthik Ramasamy, *Introduction to Heron,* in: *https://www.slideshare.net/streamlio/introduction-to-heron*
- Sanjeev Kulkarni, *Heron going exactly-once: https://streaml.io/blog/heron-going-exactly-once*
- Incubator Wiki, *Heron Proposal*, in: *https://wiki.apache.org/incubator/HeronProposal*
- Apache Calcite, in: *https://calcite.apache.org/*

Ringraziamenti

Un *grazie sincero* è rivolto al Professore Riccardo Martoglia, Relatore di questa Tesi, non solo per la fiducia nell'avermi affidato una tematica tanto delicata e attuale, qual è quella dei Big Data, ma anche per la grande disponibilità e cortesia dimostratemi.

Infine, un *grazie speciale* va alla mia famiglia e ai miei amici che mi hanno sempre sostenuto, incoraggiato e appoggiato durante l'intero percorso di studi.