A cura di: Vincenzo Fardella

Appunti Big Data

A. A. 2022/2023 - Università degli Studi di Palermo

**11 - Hadoop 1**

11.1 - MapReduce 1

11.2 - YARN 3

11.3 - Ecosistema Hadoop 4

**12 - HDFS 7**

12.1 - Caratteristiche Avanzate 7

**13 - Spark 9**

13.1 - API di Spark 9

13.2 - Meccanismi di Esecuzione 12

13.3 - Modalità di Esecuzione 12

13.4 - Ciclo di Vita dell'Applicazione 12

13.5 - Uso dell'API di Alto Livello 13

**14 - Machine Learning 14**

14.1 - Tipologie di Compiti 14

14.2 - Utilizzo dei Dati 15

14.3 - Capacità di Generalizzazione 15

14.4 - Tecniche di Addestramento 16

**15 - Clustering 18**

15.1 - Features Selection 18

15.2 - Clustering Basato su Prototipi Rappresentativi 19

15.3 - Clustering Gerarchico 21

15.4 - Clustering Probabilistico 23

15.5 - Clustering per Densità e per Griglia 24

15.6 - Misure di Bontà del Clustering 26

**16 - Classificatori 29**

16.1 - Selezione delle Features 29

16.2 - Alberi di Decisione 31

16.3 - Random Forests 32

16.4 - Classificatori Probabilistici 33

16.5 - Support Vector Machines 35

16.6 - Valutazione della Bontà della Classificazione 37

11 - Hadoop

Hadoop è un framework open source per il processing distribuito di grandi volumi di dati su cluster di computer.

Il suo obiettivo principale è quello di fornire un modo efficiente per elaborare, archiviare e analizzare grandi quantità di dati, suddividendoli in blocchi e distribuendoli su un cluster di macchine.

Il cuore di Hadoop è composto dalle seguenti componenti:

* il file system distribuito HDFS (Hadoop Distributed File System)
* il framework per il calcolo parallelo MapReduce
* il framework di gestione delle risorse YARN (Yet Another Resource Negotiator)
* un framework di elaborazione dati distribuito come Spark

**HDFS** è il file system distribuito di Hadoop che consente di archiviare dati su un insieme di nodi, in modo che i dati siano ridondanti, disponibili e scalabili. I file sono suddivisi in **blocchi di dimensioni predefinite**, generalmente tra 64 e 128 MB, e **replicati** in più nodi per garantire la tolleranza ai guasti (se un nodo fallisce, è comunque garantita la continuità grazie agli altri nodi).

**MapReduce** è invece un framework per la **programmazione parallela e distribuita** che permette di elaborare i dati archiviati su HDFS in modo efficiente e scalabile.   
Il framework suddivide un'operazione iniziale in operazioni più piccole che possono essere eseguite in parallelo su un cluster di macchine. In questo modo, i dati possono essere processati in modo distribuito su diversi nodi del cluster, con un notevole aumento delle prestazioni.

**YARN** gestisce l'allocazione delle risorse del cluster tra le applicazioni Hadoop in esecuzione. Consente di **eseguire diverse applicazioni** Hadoop contemporaneamente e di **allocare le risorse in modo dinamico** in base alle esigenze delle applicazioni.

**Spark** supporta un'ampia varietà di workload, tra cui elaborazioni batch, analisi in tempo reale e machine learning. Utilizza un modello di elaborazione in-memory che consente di ottenere prestazioni significativamente migliori rispetto a MapReduce.

Hadoop è stato progettato per funzionare su cluster di computer commodity, cioè su **hardware standard e a basso costo**. Grazie a questo approccio, è possibile costruire cluster di elaborazione ad alte prestazioni senza dover investire in costose macchine proprietarie.

Vediamo più in dettaglio come funziona ciascuna componente.

11.1 - MapReduce

Consente di elaborare grandi quantità di dati in parallelo su un cluster di computer, suddividendo il lavoro in **due fasi**: la fase di **map** e la fase di **reduce**.

Nella fase di **map**, i dati vengono elaborati **in modo parallelo** da un insieme di **mapper**. Ogni mapper legge un blocco di dati di input e **applica una funzione di mapping** ad ogni elemento del blocco. La funzione di mapping prende in input un elemento e **produce un insieme di coppie chiave-valore** (key-value pair), dove la chiave rappresenta una categoria di raggruppamento e il valore rappresenta l'informazione associata a quella categoria.

Successivamente alla fase di mapping, in cui viene generate una lista di coppie chiave-valore, abbiamo un'operazione detta di shuffle e sort, con cui i dati vengono mescolati e riordinati da due componenti dell'architettura MapReduce: **Combiner** (responsabile dello shuffle) e **Partitioner** (responsabile del partizionamento della lista). L'uso di questo meccanismo permette di alleggerire il traffico di rete (viaggiano dati meno pesanti) e di migliorare il throughput.

Per quanto riguarda l'operazione di sorting, esso avviene per chiave e, comunque, prima di finire al reducer (che quindi non deve fare queste operazioni)

Nella fase di **reduce**, i dati vengono elaborati **in modo parallelo** da un insieme di **reducer**. Ogni reducer riceve tutte le coppie chiave-valore prodotte dai mapper che hanno la stessa chiave, e **applica una funzione di riduzione** a ciascun gruppo di coppie. La funzione di riduzione prende in input la chiave e un insieme di valori associati alla chiave, e **produce un insieme di coppie chiave-valore di output**.

Il framework MapReduce di Hadoop gestisce la distribuzione dei dati tra i mapper e i reducer, la gestione degli errori, la tolleranza ai guasti e la sincronizzazione dell'esecuzione delle fasi di map e reduce.

MapReduce è molto potente perché consente di elaborare grandi quantità di dati in modo scalabile, utilizzando un approccio dichiarativo che separa la logica dell'elaborazione dei dati dall'implementazione specifica del codice. Inoltre, MapReduce può essere utilizzato in combinazione con altri framework di Hadoop, come HDFS per la gestione del file system distribuito e YARN per la gestione delle risorse, per costruire soluzioni di elaborazione di dati distribuite e scalabili.

11.1.1 - Classi Hadoop MapReduce

**InputFormat** è una tipologia di classe Java capace di specificare il formato in input al mapper, gestendo la lettura dei dati da HDFS e la creazione degli split appartenenti alla classe InputSplit che vengono letti da istanze di RecordReader per generare le coppie chiave-valore. È implementata in:

* **TextInputFormat** - usato per le linee di testo: (k, v) → (offset text-line, text-line)
* **KeyValueTextInputFormat** - usata per spezzare le linee di testo con il carattere di escape '\t' -   
  (k, v) → (offset text-line fino a '\t', text-line dopo '\t')
* **SequenceFileInputFormat** - usato per file binari che serializzano qualunque tipo di dato -   
  (k, v) definiti dall'utente
* **SequenceFileAsTextInputFormat** - come il precedente, ma usa il metodo toString() per ottenere la versione stringata di chiave e valore
* **SequenceFileAsBinaryInputFormat** - che restituisce chiave e valore come oggetti binari
* **NLineInputFormat** - permette di inviare al mapper N linee di testo gestite come (k, v) di tipo TextInputFormat
* **DBInputFormat** - che carica chiavi e valori da un DB relazionale tramite connettore JDBC.

**InputSplit** e **RecordReader** non vanno usate esplicitamente, ma vanno richiamate da altre classi di input con i metodi getSplits() e createRecordReader().

**OutputFormat** gestisce la scrittura dei risultati su HDFS e da essa derivano tipi duali rispetto a quelli implementati in InputFormat.

La gestione complessiva di un job è demandata all'esecuzione di un **JobClient** per il quale è stata impostata una configurazione definita nella classe **JobConf** che contiene le seguenti informazioni:

* Nome del job
* Nomi delle classi che implementano mapper e reducer
* Tipo della chiave e dei valori restituiti dal mapper
* Tipo della chiave e dei valori restituiti dal reducer
* Tipo del formato in input
* Tipo del formato in output

11.2 - YARN

Il funzionamento di YARN si basa su un'architettura client-server composta da tre componenti principali:

* **ResourceManager**
* **NodeManager**
* **ApplicationMaster**

Il **ResourceManager** è il componente centrale di YARN. Esso **è responsabile dell'allocazione delle risorse** del cluster alle applicazioni in esecuzione. Il ResourceManager tiene traccia delle risorse disponibili nel cluster e le assegna alle applicazioni in base alle loro richieste. Il ResourceManager è in grado di gestire più code di lavoro e di impostare limiti di utilizzo delle risorse per ciascuna di esse.

Il **NodeManager** è il componente che viene eseguito su ciascun nodo del cluster. Esso è responsabile della **gestione delle risorse locali del nodo**, tra cui la CPU, la memoria e lo spazio su disco. Il NodeManager comunica con il ResourceManager per informarlo sulle risorse disponibili nel nodo e per ottenere l'autorizzazione per l'esecuzione delle applicazioni.

L'**ApplicationMaster** è un componente **specifico dell'applicazione** che viene eseguito su un nodo del cluster. Esso è responsabile della **gestione delle risorse richieste dall'applicazione** e della **coordinazione** con il ResourceManager **per ottenere le risorse necessarie**. L'ApplicationMaster comunica con il NodeManager per monitorare l'utilizzo delle risorse del nodo e per assicurarsi che l'applicazione abbia abbastanza risorse a disposizione.

Il processo di esecuzione di un'applicazione su YARN prevede i seguenti passi:

1. L'utente sottomette la richiesta di esecuzione dell'applicazione al ResourceManager.
2. Il ResourceManager cerca i nodi del cluster con risorse sufficienti per soddisfare la richiesta dell'applicazione.
3. Il ResourceManager assegna al NodeManager di ciascun nodo del cluster le risorse richieste dall'applicazione.
4. L'ApplicationMaster viene eseguito su un nodo del cluster e richiede le risorse necessarie all'esecuzione dell'applicazione al ResourceManager.
5. L'ApplicationMaster comunica con il NodeManager per monitorare l'utilizzo delle risorse del nodo e per assicurarsi che l'applicazione abbia abbastanza risorse a disposizione.
6. L'applicazione viene eseguita sui nodi del cluster assegnati dal ResourceManager e gestiti dai NodeManager.

Un altro componente fondamentale di YARN è il **JobSubmitter**, che consente agli utenti di far eseguire jobs al sistema YARN. Esso viene **eseguito sul client**, ovvero sulla macchina dell'utente che vuole eseguire un lavoro sul cluster Hadoop.

Il JobSubmitter di YARN fornisce un'interfaccia per la sottomissione dei lavori al sistema, che può essere effettuata tramite la **riga di comando** o tramite un'**API**. Quando un lavoro viene sottomesso, il JobSubmitter si occupa di **incapsularlo in un container**, che rappresenta l'unità di esecuzione di un lavoro su YARN. Il container viene quindi **assegnato a un NodeManager sul cluster**, che si occuperà dell'esecuzione del lavoro.

Il JobSubmitter di YARN **gestisce** anche **la fase di configurazione del lavoro**, ovvero la definizione delle risorse richieste dal lavoro, dei parametri di configurazione e delle dipendenze. Inoltre, esso **fornisce meccanismi per il monitoraggio** e **il controllo dell'esecuzione dei lavori sottomessi**, ad esempio per l'invio di comandi di stop o di segnalazioni di stato.

11.3 - Ecosistema Hadoop

11.3.1 - Hive

**Hive** è un framework di elaborazione di dati distribuito in Hadoop che permette di interrogare i dati memorizzati in Hadoop utilizzando una **sintassi simile al linguaggio SQL**. Hive **converte le query SQL in una serie di attività di MapReduce** che vengono eseguite sul cluster Hadoop. In questo modo, gli utenti possono interrogare i dati in modo semplice e naturale, come se stessero lavorando con un database relazionale tradizionale, senza dover scrivere codice MapReduce.

L'accesso a Hive è consentito a client eterogenei, grazie a connettori specifici per la connessione. Ogni drive di Hive non si appoggia direttamente su MapReduce, ma usa uno **store di metadati** per schemi e tabelle.

Hive utilizza una serie di componenti per l'elaborazione dei dati, tra cui:

1. **HiveQL**: HiveQL è un dialetto SQL-like che permette di scrivere query per l'elaborazione dei dati. HiveQL supporta una vasta gamma di funzioni SQL standard, come SELECT, WHERE, GROUP BY, JOIN e altre. Hive converte automaticamente le query HiveQL in attività MapReduce da eseguire sul cluster.
2. **Hive Metastore:** Hive Metastore è un database che memorizza i metadati dei dati archiviati in Hadoop, come lo schema dei dati e le informazioni sulle partizioni. Hive utilizza i metadati del metastore per generare le attività MapReduce necessarie per eseguire le query.
3. **Driver Hive**: Il Driver Hive è il componente principale del framework Hive. Esso riceve le query dall'utente e le traduce in attività MapReduce che possono essere eseguite sui nodi del cluster Hadoop.
4. **Hive Server**: Il server Hive fornisce un'interfaccia per l'accesso ai dati Hive tramite protocollo JDBC/ODBC. Questo consente alle applicazioni di accedere ai dati Hive in modo simile a come accedono a un database relazionale tradizionale.
5. **Hadoop Distributed File System** (HDFS): HDFS è un sistema di file distribuito che viene utilizzato da Hive per memorizzare i dati elaborati.

Ma come avviene il processo di esecuzione in Hive?

Inizialmente **viene invocato il driver**, che legge le istruzione SQL e avvia una fase di **compilazione** il cui risultato è un DAG detto **piano**. A partire dal piano, **il driver cerca** nel Metastore **le informazioni sugli schemi logici** nel sistema **ed innesca l'esecuzione** vera e propria.

Una caratteristica importante di Hive è il **sistema di cache**, grazie al quale vengono evitate interazioni continue con HDFS.

Per quanto concerne i tipi di dato, abbiamo i **tipi semplici** (tipici di SQL) e i **tipi complessi**, come **Array**, **Map**, **Struct** e **UnionType**.

Le operazioni sulle tabelle sono le stesse di SQL, mentre un discorso a parte va effettuato per i dati creati in una tabella. I dati possono essere divisi in partizioni, suddivise a loro volta in **bucket**, ossia gruppi di righe raggruppati per chiave e utilizzati per rendere l'indicizzazione più efficiente.

Un'altra caratteristica interessante di Hive è che permette di **caricare i dati sia dal file system locale**, attraverso la parola chiave LOCAL, oppure da **HDFS**, attraverso la parola INPATH seguita da un percorso specifico. La parola chiave PARTITION, invece, consente di specificare la struttura di partizioni della tabella (se ammesse).

11.3.2 - Pig

È uno strumento che fa uso di **programmazione procedurale** attraverso degli **script** (detti *pig latin*) ma, a parte questa differenza, opera esattamente come **Hive**, dato che l'esecuzione di uno script genera un piano per l'execution engine, il quale genera i processi MapReduce.

Pig, inoltre, permette una maggiore versatilità di linguaggio, consentendo le **funzioni definite dall'utente**.

I **tipi di dato** sono sia **semplici**, come int, long, float, double e chararray, che **complessi**, come **tupla** (ossia la riga di un database), **mappa** (coppie chiave-valore), **bag** (insieme non ordinato di tuple).

Abbiamo, infine, **operatori di base**, come quelli aritmetici, booleani e di casting, **funzioni di load/store**, per il caricamento e la scrittura di dati in un file system locale/su HDFS, **funzioni di descrizione dei dati**, come **dump** (per descrivere il contenuto di un alias), **explain** (per mostrare la struttura del piano e la sua decomposizione in azioni semplici), **describe** (per mostrare lo schema dell'aggregato) e **illustrate** (che funziona come describe, ma inserisce esplicitamente alcuni campioni per la visualizzazione), e **operatori** **relazionali**, come order by, rank, ecc.

12 - HDFS

**HDFS** (Hadoop Distributed File System) è un **sistema di file distribuito**, parte integrante di Apache Hadoop, progettato per archiviare e gestire grandi quantità di dati in modo efficiente e affidabile su cluster di computer commodity (computer economici e di basso costo).

HDFS è **progettato per gestire file di grandi dimensioni**, tipicamente di dimensioni comprese tra i gigabyte e i petabyte, suddividendoli in **blocchi di dimensioni fisse**, tipicamente di 128 MB o 256 MB. I blocchi vengono **distribuiti su diversi nodi** del cluster e **replicati** per garantire l'affidabilità dei dati e la tolleranza ai guasti. HDFS è in grado di gestire la replicazione dei blocchi, la gestione del failover, l'accesso parallelo ai dati e la scalabilità orizzontale del cluster.

Le componenti principali di HDFS sono:

* **NameNode**: il NameNode è il componente centrale di HDFS e gestisce il namespace del file system, ovvero i nomi dei file e delle directory, nonché le informazioni di posizione e di replicazione dei blocchi di dati. Il NameNode funziona come un singolo punto di fallimento, quindi è importante avere un meccanismo di backup o di ripristino in caso di problemi.  
  Il NameNode è caratterizzato da metadati scritti in due file, uno statico e uno dinamico, aventi la stessa struttura: il primo è FsImage, che contiene la lista dei nomi del file system, il secondo è EditLogs, che registra tutte le operazioni fatte nello spazio dei nomi.
* **DataNode**: il DataNode è il componente di storage di HDFS e gestisce i blocchi di dati, ovvero il contenuto dei file. Il DataNode si occupa dell'accesso ai dati, della lettura e scrittura dei blocchi e della replicazione dei blocchi su altri nodi del cluster. I datanode sono fisicamente organizzati in rack, ossia vengono considerati come facenti parte di un unico insieme.
* **Secondary NameNode**: il Secondary NameNode è un nodo di supporto che aiuta il NameNode a gestire le informazioni sul file system. Il Secondary NameNode crea regolarmente snapshot del namespace del file system e delle informazioni di posizione dei blocchi di dati, che vengono utilizzati in caso di ripristino del sistema.

Vi sono anche altri nodi che permettono di mantenere la sincronizzazione dell'architettura, come il **CheckpointNode** e il **BackupNode**, le cui funzionalità sono intuitive.

Una proprietà importante per HDFS è la **rack awareness**, per cui si progetta un algoritmo che permetta al NameNode di **scegliere il DataNode ad esso più vicino**, in modo da limitare l'uso di banda di rete e ridurre i costi delle operazioni read/write.

Un'altra caratteristica saliente di HDFS è la cosiddetta **high availability**, ossia quella caratteristica per cui, quando un client richiede una risorsa e non ottiene una risposta per il percorso associato alla risorsa, **il NameNode riesce a restituire la nuova locazione** per leggere dei blocchi dagli altri DataNode e ottenere l'informazione richiesta.

12.1 - Caratteristiche Avanzate

* **High availability del NameNode** - Uso di almeno due NameNode ridondanti sempre in funzione, uno **attivo** e uno **passivo**. Il NameNode passivo entra in gioco nel momento in cui quello attivo va in fault, venendo messo prima in stand-by e poi divenendo il nuovo nodo passivo.  
  Per implementare tale strategia, viene fatto uso del **QuorumJournalNode**, ossia un **pool di nodi** di journaling, in cui ogni nodo **genera** e **condivide** **i propri EditLog**, che poi il nodo candidato a diventare attivo deve leggere e garantire di aver aggiornato il proprio namespace in modo corretto. Si dimostra che, per nodi di journaling, sono gestibili fallimenti.
* **Federazione di HDFS** - Per come è pensata, HDFS è letteralmente il collo di bottiglia delle applicazioni distribuite, pertanto soffre di problemi come:
  + **forte accoppiamento** tra **storage dei blocchi** e **spazio dei nomi**;
  + **poca scalabilità dello spazio dei nomi**;
  + **performance limitate dal throughput del NameNode**;
  + assenza di isolamento tra le applicazioni e tra chi gestisce il cluster e la gestione dello spazio dei nomi.  
    Pertanto, si può pensare di **federare più HDFS**, facendo in modo che ogni NameNode abbia un pool di blocchi, che vi sia gestione separata dei DataNode, che vi sia isolamento tra le applicazioni, scalabilità dello spazio dei nomi e, in generale, un aumento delle performance.
* **Disk Balancing** - Ossia il **bilanciamento del carico** a livello dei singoli DataNode. Quando uno dei DataNode risulta in sovraccarico, **HDFS sposta internamente dei blocchi** per cercare di mantenere omogenea l'occupazione di spazio su tutti i dischi del DataNode.
* **Erasure Coding** - Ossia un metodo di **protezione dei dati** per cui questi ultimi vengono suddivisi in frammenti, espansi e codificati con parti di dati ridondanti, per poi venire archiviati in posizioni diverse. In questo modo, se un'unità è guasta o i dati vengono danneggiati, essi possono essere ricostruiti a partire da tali frammenti, superando, di fatto, alcune limitazioni legate alla struttura a tre repliche di un blocco.

13 - Spark

**Apache Spark** è un framework di analisi open-source usato per **l'elaborazione dei dati su larga scala**.

Spesso si fa riferimento a un **ecosistema Spark**, sebbene esso sia uno dei componenti del più ampio Hadoop, in quanto Spark può essere usato in **modalità standalone** su una singola macchina.

Le API di Spark consentono di adoperarlo in Python, Java, Scala e altri linguaggi di programmazione, sebbene Scala sia il linguaggio nativo del framework. Il supporto a Java è garantito attraverso la JVM, mentre quello a Python attraverso l'uso di driver.

13.1 - API di Spark

Il framework si compone delle seguenti API:

* **Spark Core** - Consente di lavorare indistintamente sui vari linguaggi di programmazione
* **Spark SQL** - Permette l'astrazione dei dati SQL
* **Spark Streaming** - Permette la manipolazione dei dati
* **MLlib** - API usata per il machine learning
* **GraphX** - API che consente la manipolazione dei dati su grafo
* **PySpark** - API che consente di lavorare in Python

Vediamo, ora ciascuna API in dettaglio.

13.1.1 - Spark Core

È l'API principale con cui opera Spark, di cui ne rappresenta il **motore di esecuzione** vero e proprio. Il flusso di lavoro di Core consiste in una serie di **trasformazioni** sui dati in input per la successiva esecuzione di una o più **azioni**.  
Tra le caratteristiche garantite da Core abbiamo la **fault recovery**, ossia la capacità di ridurre al minimo la corruzione dei dati dovute a errori accidentali.

Un'altra caratteristica interessante di Core è che le **operazioni** vengono **eseguite direttamente in memoria**, così da ridurre al minimo le interazioni con il file system. La rappresentazione dei dati usata in memoria prende il nome di **Resilient Distributed Dataset (RDD)**.

Durante il flusso di lavoro di Core, possiamo rappresentare i dati sostanzialmente in due livelli di astrazione:

* ad **alto livello**, su cui viene definito un **DataFrame** parallelo a SQL
* a **basso livello**, dove troviamo gli **RDD**, su cui è possibile lavorare tramite un'interfaccia a oggetti.

Per definizione, **un RDD è una collezione immutabile di dati**, pertanto, ad ogni trasformazione, verrà generato un nuovo RDD; discorso diverso per le **partizioni**, ossia la **suddivisione logica dei dati**, dato che sono mutabili e quindi è possibile trasformarle dinamicamente.

Come in altri framework (Pig e Hive), le operazioni vengono formulate attraverso **piani**, ossia DAG: in questo modo, ricordando che dopo ogni operazione viene generato un nuovo RDD, in caso di errore è possibile ripercorrere il DAG all'indietro per recuperare l'ultima RDD "funzionante".  
Definiamo **data lineage** la **sequenza di RDD generati dalle varie trasformazioni**.

Un'altra caratteristica interessante di Core è l'uso della cosiddetta **lazy evaluation**, secondo cui i dati vengono modificati solo quando bisogna eseguire l'azione. Questa proprietà consente di **massimizzare il throughput** di computazione e di concepire le **operazioni sulle RDD come una pipeline**.

Per quanto concerne le trasformazioni possibili, Spark ne distingue due tipologie:

* **Narrow** - Trasformazioni che avvengono **internamente alle partizioni**. Possono essere eseguite in pipeline. A questa categoria di trasformazioni appartengono:
  + **Map**
  + **FlatMap**
  + **MapPartition**
  + **FIlter**
  + **Sample**
  + **Union**
* **Wide (shuffle)** - Trasformazioni che possono coinvolgere **più partizioni** e che prevedono operazioni di **shuffle**. A questa categoria di trasformazioni appartengono:
  + **Intersection**
  + **Distinct**
  + **ReduceByKey**
  + **GroupByKey**
  + **Join**
  + **Cartesian**
  + **Repartition**
  + **Coalesce**

Le azioni sono operazioni che non producono RDD, ma sono vere e proprie computazioni sui dati. Alcune azioni sono:

* **first**()
* **take**()
* **reduce**()
* **collect**()
* **count**()

13.1.2 - Spark SQL

È una API orientata ai **dati strutturati**, in quanto utilizza il livello più alto di rappresentazione dei dati. Fornisce una rappresentazione a **DataFrame**, equivalente alla tabella relazionale. Questa rappresentazione ad alto livello astrae quella a più basso livello della RDD. Spark SQL usa un **linguaggio SQL-Like** per eseguire query distribuite. Fornisce un'interfaccia SQL all'ambiente Spark, con connettività JDB/ODBC e, grazie ad Hive, permette di eseguire **processing a batch**.

13.1.3 - Spark Streaming

È un'API che permette l'elaborazione di **flussi di dati**, letti da **fonti eterogenee** (es. Kafka e Flume) o direttamente da una socket TCP.

Le caratteristiche principali di Spark Streaming sono:

* **Gathering** - Ossia la **raccolta dei dati** da sorgenti semplici (file system o socket TCP) o complesse
* **Processing** - Ossia applicazione di trasformazioni e azioni su un modello dei dati specifico, il **DiscretizedStream (DStream)**, banalmente una sequenza di RDD
* **Data storage** - Ossia la memorizzazione dei dati su file system, strutture di dashboard o database.

13.1.4 - Spark ML

È un'API contenente algoritmi di machine learning. Al suo interno possiamo trovare:

* Primitive per operazioni di **data preprocessing** (es. cleaning, imputazione)
* **Algoritmi di apprendimento supervisionato** per classificazione, regressione, decision trees, random forests, Bayes-naive, ecc.
* **Algoritmi di apprendimento non supervisionato** (es. K-means, misture di gaussiane)
* **Algoritmi di recommendation**, utili per il mining di pattern frequenti
* **Algoritmi di analisi dei grafi** per operazioni di ricerca di cammini tra nodi, PageRank, ecc.
* **Algoritmi di deep learning**, in quanto Spark ML ha solo un MLP nativo per reti non profonde, ma offre possibilità di integrazione con motori di deep learning come TensorFlow.   
  Quest'ultima integrazione è possibile grazie a librerie come TensorFrames, che rende i DataFrame di Spark il backend per gestire la computazione e TensorFlow il backend per le operazioni deep, e TensorFlowOnSpark, che distribuisce il job TensorFlow su un cluster Spark.

13.1.5 - Spark GraphX

È un'API dedicata alla **manipolazione di grafi**. Gestisce algoritmi di ricerca propri del machine learning, algoritmi di ricerca di cammini, attraversamento, clustering e classificazione su grafi.

13.1.6 - SparkR

È un'API per l'**interazione Spark-R**, con quest'ultimo che usa un proprio modello di DataFrame per gestire i dati. Fornisce metodi di connettività a database, machine learning, analisi dei dati, ecc.

13.1.7 - PySpark

È un'interfaccia Python per Spark, disponibile tramite **pacchetto** in ambiente Python. **Fornisce l'accesso ad una SparkSession**. È possibile utilizzarlo sia tramite CLI che tramite IDE, anche se in questo caso bisogna impostare una variabile d'ambiente e bisogna aver preinstallato Hadoop.

Dopo aver eseguito l'accesso alla SparkSession, è possibile accedere alle varie proprietà di una sessione, come il contesto, e alle varie API che abbiamo trattato.

13.2 - Meccanismi di Esecuzione

Un'applicazione Spark ha inizio nel momento in cui **vengono inizializzate una SparkSession** (dall'utente) **e l'interazione con il cluster manager**, che insieme prendono il nome di **processo Driver**.

Il processo Driver **coordina i processi esecutori** distribuiti sul cluster, i quali comunicano con il processo Driver attraverso il **Cluster Manager** (che può essere standalone, YARN o Mesos).

La SparkSession è un processo Java, dunque necessita di una JVM. La gestione della SparkSession differisce nelle varie versioni di Spark:

* nelle **versioni 1.x** si utilizzavano SparkContext, SQLContext e HiveContext
* nelle **versioni 2.x** si utilizza SparkSession, che integra l'oggetto SparkContext, responsabile della connessione al cluster.

13.3 - Modalità di Esecuzione

Dal punto di vista della distribuzione dei job abbiamo **tre modalità**:

* **cluster mode**, in cui l'utente sottomette il driver al cluster manager, che poi gestisce la distribuzione degli esecutori
* **client mode**, in cui il driver rimane presso la macchina dell'utente, fuori dal cluster. In questo caso il client è responsabile del funzionamento degli esecutori
* **local mode**, in cui l'esecuzione è standalone su singola macchina e l'unica forma di parallelizzazione è a livello di thread (motivo per cui è spesso usata per scopi didattici, test e debug).

13.4 - Ciclo di Vita dell'Applicazione

Il ciclo di vita di un'applicazione Spark può essere suddiviso in più fasi:

* **Configurazione**: In questa fase, vengono configurati i parametri dell'applicazione, come ad esempio il numero di processi executor da utilizzare e la quantità di memoria da assegnare a ogni processo.
* **Creazione del contesto**: In questa fase viene creato il contesto Spark, che rappresenta l'entry point per l'applicazione Spark. Il contesto Spark viene creato dal processo driver.
* **Creazione dei RDD**: In questa fase, l'applicazione Spark crea i RDD, secondo le specifiche viste in precedenza.
* **Trasformazione dei RDD**: In questa fase, l'applicazione Spark applica le trasformazioni ai RDD.
* **Azione sui RDD**: In questa fase, l'applicazione Spark esegue le azioni sui RDD, che rappresentano le operazioni che restituiscono un risultato.
* **Chiusura dell'applicazione**: In questa fase, l'applicazione Spark viene chiusa e i processi executor vengono terminati.

Durante le fasi intermedie, vi è uno scambio continuo di informazioni tra i processi in esecuzione sul cluster, ossia Driver ed executors, mappati, almeno su YARN, al livello di application master.

13.5 - Uso dell'API di Alto Livello

Per invocare l'API di alto livello, l'**utente deve scrivere del codice** capace di manipolare DataFrame, Dataset o codice SQL. Tale codice viene poi **validato da Spark** per la generazione di un piano logico, reso ottimale grazie a **Catalyst**, che poi viene tradotto in un **piano fisico**. Spark esegue, successivamente, il piano fisico, che altro non è che una sequenza di trasformazioni e azioni.

Entrando più nel dettaglio:

1. Dal codice utente (che non ha informazioni sui Dataframe) **viene generato un piano logico** non risolto e non valido, in termini di sequenze di trasformazioni e azioni.
2. Il piano viene, quindi, sottoposto a **Catalyst**, che interroga **Catalog**, un repository contenente tutte le informazioni sui Dataframe, **al fine di validare il piano** rispetto ai Dataframe finora generati.
3. Dopo aver validato il piano logico, **Catalyst lo ottimizza**, anche alterando la sequenza di operazioni pensata dall'utente.
4. Poiché ad un piano logico possono corrispondere più piani fisici di esecuzione, **Spark stabilisce quale sia il migliore**, che sarà poi eseguito tramite pipeline.

14 - Machine Learning

Il **machine learning** si riferisce allo sviluppo di programmi per computer capaci di **apprendere dai dati**. Più in generale, una macchina avrà a disposizione un'esperienza E rispetto a una classe di task T da svolgere e dovrà misurare una certa performance P, secondo la logica che P deve migliorare sui compiti T usando l'esperienza E.

Volendo fornire una rappresentazione più matematica del significato di machine learning, definiamo:

* un insieme di campioni a disposizione del calcolatore,
* un modello di apprendimento, ossia una caratterizzazione della famiglia di forme funzionali usate o dell'algoritmo usato,
* i parametri del modello, ovvero delle quantità direttamente coinvolte nella forma funzionale di f che devono essere apprese
* gli iperparametri del modello, cioè quantità che influenzano l'evoluzione dell'algoritmo, ma che devono essere fissate in fase di apprendimento (dovendo essere, il più delle volte, stimati)

L'apprendimento viene misurato dalla relazione .

14.1 - Tipologie di Compiti

In base al tipo di apprendimento, gli algoritmi di machine learning devono impiegare f distinte, specifiche per il compito in questione.   
Innanzitutto, distinguiamo **compiti con output strutturato e non**. Tra i compiti con output non strutturato abbiamo:

* **Classificazione**, in cui l'algoritmo deve stimare una funzione che fa corrispondere ogni ingresso a una delle classi possibili, secondo una relazione . Questa rappresentazione è accettata poiché, spesso, i problemi di classificazione fanno riferimento a classi numeriche.  
  L'utilità della classificazione non si limita soltanto all'attribuzione di una classe, ma può essere usata per stimare delle features mancanti.
* **Regressione**, in cui, a partire dai dati in input, bisogna predire un valore numerico reale per una data funzione. L'approccio è simile alla classificazione, ma cambia il codominio della funzione , che diventa in quanto deve predire singoli valori reali.
* **Trascrizione**, in cui si deve estrarre una trascrizione in forma testuale di dati scarsamente strutturati (basti pensare alle OCR o alla speech recognition). Tale compito può essere pensato come una forma di classificazione in cui le etichette non sono numeriche, ma sono piuttosto le parole di un testo.

**Traduzione automatica**, con cui si traduce una sequenza discreta di simboli in un'altra.

Per quanto concerne i compiti con output strutturato ricordiamo:

* **Parsing**, ossia la trasformazione di un testo in una sequenza di simboli che definisce dei ruoli.
* **Quality** **Assurance**, che consiste in un processo sistematico che determina se un prodotto o un servizio soddisfino (o meno) dei requisiti specifici
* **Segmentazione di immagini**, in cui viene restituito un output di etichette, corrispondente alle classi di segmentazione (o regione dell'immagine).

Tra gli altri compiti abbiamo:

* **Outlier Detection**, il cui scopo è quello di rilevare anomalie nel comportamento di determinati ingressi. Spesso si associa al clustering/classificazione interpretati in forma duale (ovvero si restituiscono gli elementi con comportamento "non normale").
* **Sintesi e campionamento**, che consiste nella generazione di campioni strutturalmente simili con i dati di ingresso, in modo da incrementare la popolazione di ingresso con dati "come gli originali" e migliorare forme di apprendimento statistico.
* **Imputazione di dati mancanti**, con cui si predicono le features mancanti a partire da un input incompleto.
* **Denoising**, ossia la predizione di campioni non corrotti a partire da input rumorosi.
* **Stima di una densità di probabilità o di una funzione massa**, con cui bisogna stimare una funzione da intendersi come funzione di densità di probabilità (per variabili continue) o funzione di massa (per variabili discrete).

Sebbene ne siano stati elencati diversi, sostanzialmente si parlerà sempre di tre tipologie di problemi: **classificazione**, **regressione** e **clustering**.

14.2 - Utilizzo dei Dati

Distinguiamo tre modalità di uso dei dati in fase di apprendimento:

* **apprendimento non supervisionato**, in cui gli algoritmi hanno l'obiettivo di apprendere una distribuzione di probabilità che sottende il dataset, o meglio, la **struttura dei dati**. Esempi di apprendimento non supervisionato sono il clustering e il denoising.
* **apprendimento supervisionato**, in cui gli algoritmi hanno il compito di catalogare un campione in un'etichetta y facendo in modo da **massimizzare la probabilità** . È l'approccio tipico di classificazione e regressione.
* **apprendimento con rinforzo**, in cui l'algoritmo apprende su un ambiente dinamico attraverso **sensori e attuatori**, secondo un metodo try/error.

14.3 - Capacità di Generalizzazione

Perché un algoritmo di apprendimento possa essere considerato valido, è necessario che sia capace di **generalizzare**, ossia di ottenere buone performance su campioni mai visti. Per far ciò è necessario che l'algoritmo sia capace di **minimizzare due misure di errore**: il **training error** e il **test/generalization error**.

La procedura standard prevede che, qualora non sia disponibile un test set, si **suddivida il dataset di partenza in due partizioni**, un training set e un test set. A questo punto, la forma funzionale L delle misure di errore, nota come **funzione di perdita**, deve essere la stessa per entrambe le misure di errore e deve essere scelta accuratamente in base al tipo di compito che l'algoritmo deve svolgere.

**La capacità di generalizzare si basa sulla teoria dell'apprendimento statistico**, secondo cui esiste un processo statistico unico capace di descrivere un fenomeno sotto esame e che genera dati sia di addestramento che di test. Questo processo è descritto da un'unica pdf dei dati p\_\mathrm{data}, non nota a priori.  
Si può stabilire, quindi, un'ipotesi aprioristica secondo cui i dati dei due set sono indipendenti e hanno la stessa distribuzione di probabilità, motivo per cui avranno lo stesso valore atteso. L'ideale allora diventa **cercare i parametri** \bold{w} **tali che i due errori siano uguali**, sebbene ciò sia impossibile a causa di alcune distorsioni dovute al campionamento per la scelta dei dati.   
A causa di questo motivo, allora, si procede in questo modo:

1. Si cerca di **minimizzare il training error** attraverso l'addestramento.
2. Si cerca di **mantenere minima la differenza tra training error e generalization error**.

Se non si rispettano entrambi i principi si possono verificare le situazioni di **overfitting** (scarsa capacità di generalizzazione) o **l'underfitting** (scarsa efficacia dell'algoritmo), due situazioni legate a un eccessivo sbilanciamento verso l'uno o l'altro errore in relazione alla capacità del modello (la possibilità del modello di approssimare la maggiore gamma di forme funzionali possibile).

I **parametri** del modello, invece, **definiscono una classe di funzioni che contribuiscono a selezionare una particolare forma funzionale** interna alla classe. Discorso diverso vale per gli **iperparametri**, che permettono di **spaziare le funzioni che si vogliono stimare**; tra gli iperparametri della rete (che sono impostati a priori) abbiamo: il numero di layer nascosti, lo schema di inizializzazione dei pesi, il learning rate, il learning rate decay, ecc.

14.3.1 - Teorema No Free Lunch

È un teorema che afferma che l**e performance di un qualunque algoritmo di apprendimento mediata su qualunque distribuzione di probabilità è la stessa**.  
Ciò equivale a dire che non esistono algoritmi di base migliori di altri, e che la scelta di un algoritmo è solitamente guidata dalla distribuzione di probabilità p\_\mathrm{data}.

Un'altra considerazione va fatta, poi, in merito al fatto che il **training error e la generalizzazione di un algoritmo dipendono fortemente dalle dimensioni del training set**.

14.4 - Tecniche di Addestramento

Sono tecniche che permettono di **personalizzare l'addestramento** tradizionale, al fine di aumentare la capacità effettiva del modello, che quasi sempre differisce dalla sua capacità teorica.   
Un tipico **approccio di addestramento sugli iperparametri**, che poi influenza la capacità di rappresentazione del modello, è quello **a griglia**: si definisce un intervallo di variazione per ogni parametro, si campiona un certo numero di valori in ogni intervallo di variazione e si crea la combinazione (da qui la griglia) di tutti i valori possibili degli iperparametri.

Due esempi di tecniche capaci di migliorare la capacità effettiva sono la **regolarizzazione** e la **validation set e cross-validation**.

14.4.1 - Regolarizzazione

La **regolarizzazione** è una qualunque **modifica dell'algoritmo di apprendimento mirata a ridurre esplicitamente l'errore di generalizzazione, ma non il training error**. Consiste nell'aggiunta di un termine di regolarizzazione nella funzione L che consente di preferire alcune forme funzionali rispetto ad altre nel rispetto dello spazio delle ipotesi.

Un esempio di regolarizzazione è la **weight decay**, che esprime preferenza per piccoli valori di \bold{w}, e che consiste nell'aggiunta di un iperparametro \lambda alla funzione di loss allo scopo di **privilegiare una certa configurazione dei parametri**.

14.4.2 - Validation set e cross-validation

L'uso di un **validation set** costituisce un'altra tecnica di modifica dell'algoritmo di apprendimento per ovviare al problema dell'apprendimento degli iperparametri. Definiamo validation set una **porzione di training set estratta da quest'ultimo** prima di iniziare l'addestramento vero e proprio. Viene utilizzato per verificare l'andamento del generalization error, modificando gli iperparametri per minimizzarlo.

Si può affermare che **l'errore commesso in fase di validazione approssimi il generalization error meglio di quanto non faccia il training error**.

Un'altra tecnica di addestramento è la **k-fold cross-validation**, con cui si suddivide il dataset in k partizioni (fold) non sovrapposte e, per ogni partizione, si usa una strategia in cui quest'ultima funziona da test set e le restanti k-1 fungono da training set.  
Con questa tecnica, ad ogni fold, **non cambia l'insieme dei parametri e degli iperparametri**, ma si passa semplicemente da uno all'altro fold. Poiché con questa tecnica ogni test set viene incontrato k-1 volte come training set, **non si calcola un singolo errore di generalizzazione, ma ne viene calcolata una media** su tutte le partizioni.

**N.B.**: Tipicamente, prima di applicare la k-fold cross-validation si effettua uno shuffle del dataset, spesso anche all'interno delle varie epoche di addestramento, in modo da evitare possibili bias legate all'ordine di presentazione dei campioni.

15 - Clustering

Si definisce **clustering** il processo di **partizionamento di un dataset** in modo che ogni partizione (o cluster) contenga **dati molto simili** tra loro rispetto ai dati appartenenti ad altre partizioni.

Quella appena data è, però, una definizione molto approssimativa del processo, in quanto fortemente **influenzato dai diversi tipi di problema** in cui può essere impiegato (es. Data summarization, Customer segmentation, Social Network Analysis, ecc.).

Come visto nel capitolo precedente, gli algoritmi di clustering rappresentano forme di apprendimento non supervisionato.

Gli algoritmi di clustering sono affetti da un **problema legato alla presenza di *rumore*** o al contenuto poco informativo di alcune features, che portano a dire che, prima di applicare un algoritmo di clustering, è necessario effettuare una *features selection*.

15.1 - Features Selection

La **features selection** si pone l'obiettivo di **rimuovere le features con rumore o**, comunque, quelle che hanno scarsa tendenza alla *clusterizzazione*.  
Distinguiamo **due classi** di modelli per fare features selection:

* **Modelli filter-based**, in cui si valuta la **tendenza di ogni feature a contribuire al clustering**, assegnandole un punteggio numerico attraverso l'uso di un criterio di similarità. Le features che non raggiungono un certo punteggio vengono rimosse.  
  Tendenzialmente sono modelli **più potenti** poiché tengono conto dell'impatto incrementale dell'aggiunta di una feature ad altre.
* **Modelli wrapper,** in cui si **procede iterativamente su sottoinsiemi di features**. In questo approccio una buona scelta delle features dipende dal cluster e viceversa, andando a scegliere features sulla base della particolare metodologia usata nella generazione dei cluster.  
  Sono modelli **computazionalmente onerosi**, ma possono fornire una base per la scelta del migliore algoritmo di clustering per il problema in esame.

15.1.1 - Modelli Filter-Based

Abbiamo:

* **Term strength**, adatta per l'analisi di **domini sparsi** (come quello testuale). Si parla di **presenza/assenza di un termine** e di similarità piuttosto che di distanze, e permette di individuare le parole più rilevanti rispetto al dataset.  
  La term strength è definita come la **frazione di coppie di documenti simili in cui occorre un certo termine** , ossia .  
  Può essere estesa a dati multidimensionali, discretizzando gli attributi quantitativi in valori binari e rappresentando gli attributi come vettori binari di presente/assente.
* **Dipendenza predittiva da attributi**, che parte dalla considerazione che **attributi correlati risulteranno sempre in clustering migliori** rispetto ad attributi non correlati. Il metodo sfrutta algoritmi di regressione per attributi numerici o algoritmi di classificazione per attributi non numerici. La rilevanza di un attributo i è data dalla precisione dell'algoritmo di prevedere il valore di i venendo applicato su tutti gli altri attributi.
* **Entropia**, con cui si indica una famiglia di modelli accomunati dal fatto che i dati raggruppati in cluster riflettono alcune **caratteristiche di clustering sulle distribuzioni di distanza**. In generale, il calcolo dell'entropia su un sottoinsieme di features k-dimensionale prevede che si discretizzino le features al fine di stimare la pdf di ogni intervallo, su cui poi viene calcolata l'entropia (secondo le modalità descritte nei capitoli precedenti); il risultato di un tale approccio è il sottoinsieme di features che ha **generato l'entropia minore** (poiché tendenti a formare un cluster). Il problema dell'approccio appena descritto risiede nella **dimensionalità** **del sottoinsieme di features**, in quanto bisogna far uso di una misura di distanza su griglie multidimensionali (quindi tendenzialmente sparse).
* **Statistica di Hopkins**, per cui, considerato un campione costituito da punti casuali, e un campione sintetico di punti spazio dei dati, e posti e , le distanze dei punti in e dal più vicino campione di , definisce la quantità , che ha valori in e per cui implica dati distribuiti uniformemente ( e simili), implica dati clusterizzati ().

15.1.2 - Modelli Wrapper

Si tratta di modelli che consistono nell'**uso di algoritmi di clustering su sottoinsiemi di features scelti e nella verifica della qualità del clustering**. Tra i metodi usati per la scelta dei sottoinsiemi di features si possono usare **algoritmi greedy** (che scartano le fatures che causano i maggiori incrementi nel criterio di validità del cluster) oppure la scelta delle singole features sulla base di un **criterio di selezione** (col vantaggio che si calcola la bontà della singola feature, piuttosto che di un intero sottoinsieme).

Per la loro natura, i modelli wrapper sono spesso utilizzati con modelli filter-based per creare **modelli ibridi**, al fine di raggiungere un'efficienza migliore.

15.2 - Clustering Basato su Prototipi Rappresentativi

Si tratta di algoritmi di clustering più semplici, in quanto fanno riferimento a nozioni intuitive come **distanze/similarità dei dati del cluste**r e generano cluster **privi di relazioni gerarchiche**.

L'obiettivo di tali algoritmi è la creazione di cluster stabiliti dall'utente, a cui gli elementi vengono assegnati sulla base di una funzione di distanza, secondo l'idea che cluster significativi sono strettamente collegati a prototipi rappresentativi significativi.

Per capire meglio l'approccio, supponiamo che un dataset contenga dati nello spazio d-dimensionale, indicati con , e che l'obiettivo sia di determinare prototipi rappresentativi, indicati con , capaci di minimizzare la funzione , dove è una funzione caratteristica dell'algoritmo di clustering.

Quello appena descritto è, in realtà, un **problema di ottimizzazione**, risolto spesso con un **approccio iterativo** in cui i prototipi rappresentativi e le assegnazioni vengono utilizzati per migliorarsi a vicenda, in cui non possono mancare:

* **Passo di assegnazione**, in cui **si assegna ogni campione** al prototipo rappresentativo più vicino (sulla base della scelta) e si denotano cluster
* **Passo di ottimizzazione**, in cui **si determina il prototipo rappresentativo** per ogni cluster che minimizza la funzione obiettivo locale .

L'idea è quella di migliorare la funzione obiettivo su più iterazioni, sebbene **i miglioramenti più significativi si riscontrano solo nelle prime iterazioni**. Il collo di bottiglia di questo genere di problemi è il passo di assegnazione, in cui è necessario calcolare tutte le distanze dati-prototipi.

15.2.1 - K-Means

È un algoritmo caratterizzato da , ossia dallo **scarto quadratico medio tra il campione e il prototipo rappresentativo**. Si dimostra che i valori ottimali di sono i valori medi dei punti nel cluster .

Tra le varianti di questo algoritmo abbiamo:

* **Mahalanobis K-Means**, che usa la **distanza di Mahalanobis** locale per l'assegnazione dei dati nel cluster, sfruttando le matrici di covarianza del cluster, secondo la relazione . La covarianza locale del cluster fornisce una misura implicita di densità del cluster, per cui si presta alla gestione di **cluster a densità variabile**.  
  L'algoritmo adattato con la distanza di Mahalanobis è **adatto per cluster di forma sferica**, dunque non rappresenta la scelta migliore per cluster di forma arbitraria.
* **K-Means del kernel**, che rappresenta **un'estensione del k-means per cluster di forma arbitraria**. Tale estensione, ottenuta con il **"kernel trick"**, permette di trasformare implicitamente i dati in modo che i **cluster di forma arbitraria siano mappati a cluster euclidei in un nuovo spazio**, risolvendo il problema della non-linearità dei dati.  
  Consideriamo una funzione che mappa un vettore in uno spazio a più alta dimensionalità: definiamo kernel una qualunque funzione ottenuta come prodotto interno di due mapping uguali e dal loro spazio di partenza a uno con una più alta dimensionalità o, in altri termini, .  
  Da notare che il kernel è un operatore lineare nello spazio trasformato, per cui si possono utilizzare algoritmi lineari nel nuovo spazio definito da .  
  Importante è, infine, notare che **non bisogna mai calcolare** , in quanto la funzione viene definita in termini di vettori nello spazio di ingresso.
* **K-Medians**, che fa uso della **distanza di Manhattan** . Si dimostra che **il prototipo ottimale** **è la mediana dei punti lungo ciascuna dimensione** nel cluster , e che (solitamente) **non apparterrà al dataset**.  
  Questo approccio si dimostra **robusto rispetto ai valori anomali** grazie all'uso dei mediani, cosa che non è garantita dal clustering k-means classico.  
  Spesso quest'algoritmo viene confuso con quello K-Medoids.
* **K-Medoids**, che, pur usando la nozione dei prototipi rappresentativi, usa una struttura algoritmica diversa. La principale differenza è che i prototipi rappresentativi sono sempre scelti **all'interno del dataset** per due motivi: 1) Il prototipo rappresentativo di un cluster k-means può essere distorto dalle anomalie nel cluster (si può trovare in una regione distinta dalla maggior parte dei punti del cluster) e 2) Può essere difficile calcolare il prototipo rappresentativo su un tipo di dato complesso (es. le serie temporali).  
  L'algoritmo fa uso di una **strategia di hill-climbing**, per cui all'inizio si scelgono i prototipi come un insieme di punti di , per poi venire migliorato con scambi iterativi tra prototipi e punti di . In base a come vengono eseguiti gli scambi possono insorgere **forti problemi computazionali**, anche se esistono varianti della strategia che permettono di abbattere questo costo.

15.2.2 - Problemi Pratici e di Implementazione

* **Criteri di inizializzazione**: il criterio più semplice è di scegliere punti casuali nel dominio dello spazio o di campionare (solitamente meglio poiché porta a prototipi statistici migliori).  
  Di base, gli algoritmi con prototipi sembrano robusti rispetto all'inizializzazione, sebbene ciò non è garantito (per migliorare la situazione si possono campionare più punti di rispetto a ed utilizzare clustering agglomerativi gerarchici).
* **Scelta del numero di cluster**, che spesso dev'essere una sovrastima di quello ipotizzato dall'analista, in modo da rendere meno probabile l'unione errata di cluster e da posticipare alla fine l'unione di cluster più simili.
* **Presenza di anomalie**, in quanto la scelta di un prototipo anomalo (soprattutto nelle fasi iniziali dell'algoritmo) può avere un impatto negativo. Per ovviare al problema una soluzione può essere aggiungere all'algoritmo una fase in cui vengono scartati i centroidi con cluster molto piccoli e si sostituiscono con punti scelti a caso in .

15.3 - Clustering Gerarchico

Rappresenta una famiglia di algoritmi che generano **cluster aggregabili attraverso una misura di distanza** (o altri metodi come la densità o l'uso di grafi).  
I cluster generati in modo gerachico sono utili in quanto permettono di dare maggiori informazioni ai vari livelli di granularità.

Esistono due categorie di algoritmi di clustering gerarchico, distinte in base all'approccio:

* **Algoritmi Agglomerativi (o bottom-up)**, in cui i dati vengono **accorpati consecutivamente** in cluster di livello superiore.   
  Di solito, un algoritmo bottom-up inizia con i singoli dati che **definiscono ognuno un cluster**; nelle iterazioni successive, si scelgono i due cluster più vicini/simili e vengono sostituiti da un cluster aggregato. Poiché ad ogni iterazione il numero viene ridotto di 1, bisogna trovare il giusto metodo per misurare la distanza in modo da aggregare i cluster.  
  Da un punto di vista implementativo, si tiene traccia delle distanze in una matrice (con il numero di cluster dopo t iterazioni) e, dopo ogni unione, si definiscono i valori aggiornati delle distanze in una nuova matrice .  
  L'algoritmo di generazione dei cluster termina al **soddisfacimento di una condizione** che può essere la massima distanza tra due cluster o il numero minimo di cluster.
* **Algoritmi Divisivi (o top-down)**, in cui i dati vengono **divisi successivamente** in una struttura ad albero.  
  In generale, un algoritmo top-down usa un algoritmo di clustering flat come subroutine (indicato con ). L'algoritmo inizializza l'albero nel nodo radice contenente tutto il dataset. Ad ogni iterazione il dataset di un particolare nodo dell'albero corrente viene suddiviso in più nodi (rappresentanti i cluster). Se è randomizzato, come il k-means, è possibile eseguire più volte lo stesso algoritmo in un dato nodo e **selezionare il risultato migliore**.

15.3.1 - Algoritmi Agglomerativi

In quello che segue, indicheremo i gruppi da unire con gli indici e . I metodi per calcolare le distanze tra i due gruppi sono:

* **Single linkage (best linkage)**, in cui l**a distanza è il minimo** tra le coppie di distanze possibili, ossia alla distanza associata alla **coppia di elementi più vicini**.  
  Dopo aver effettuato l'unione, si eliminano le righe e le colonne e da e si sostituiscono con una sola riga e una sola colonna rappresentante l'unione. La creazione della nuova riga può avvenire usando il minimo dei valori nella coppia di righe (o colonne) eliminate da .  
  Per quanto riguarda gli altri elementi, dato il cluster , si scelgono per le righe e per le colonne di .  
  L'approccio single linkage è un metodo agglomerativo molto efficace per scoprire cluster di forma arbitraria.
* **Complete linkage (worst linkage)**, in cui **la distanza tra due cluster è la massima** tra le coppie di distanze possibili, ossia alla distanza associata alla **coppia di elementi più distanti**.  
  Il procedimento di aggiornamento di è analogo a quello del single linkage, soltanto che per gli elementi si scelgono per le righe e per le colonne di .  
  Caratteristica di questo algoritmo è quella di generare cluster con il diametro minimo (in quanto evidenzia la dissimilarità tra due cluster).
* **Group-average linkage**, in cui la **distanza è uguale alla distanza media** tra tutte le possibili coppie di distanze.  
  Per generare la nuova riga di in sostituzione di e , si utilizza la media pesata della i-esima e j-esima riga (colonna), mentre per generare le righe si considerano per le righe e per le colonne di .
* **Centroide più vicino**, in cui si uniscono i cluster con i centroidi più vicini. In genere questo metodo è da evitare in quanto i centroidi perdono informazioni sulle propagazioni relative dei cluster.
* **Varianza dei cluster**, che minimizza le variazioni della funzione obiettivo come risultato della fusione. Con questo criterio, si cerca di fondere i cluster quando l**a variazione della funzione obiettivo è la minima possibile** (in modo da preservare le proprietà statistiche del cluster). Ad ogni unione, si eliminano le righe e da e vengono aggiunte una riga e una colonna per il cluster risultante, mentre per gli elementi , la riga (colonna) viene calcolata usando le statistiche del momento del cluster.
* **Metodo di Ward**, che non usa la variazione di varianza come criterio di fusione ma la **somma** (non in scala) **dell'errore quadratico**.

In generale, usando un approccio bottom-up è più difficile controllare la struttura dell'albero gerarchico. Pertanto, si preferisce un approccio top-down quando si desidera una tassonomia di una struttura specifica.

15.3.2 - Bisecting K-Means

È un algoritmo in cui ogni nodo è diviso **esattamente in due figli** per ogni iterazione, usando un algoritmo 2-means. Per dividere un nodo in due figli vengono usate diverse esecuzioni casuali della divisione, per poi scegliere la divisione che ha il **miglior impatto sull'obiettivo** di clustering generale.

15.4 - Clustering Probabilistico

Gli algoritmi di **clustering probabilistico** assumono che ogni dato può appartenere a un cluster con **una certa probabilità** (quindi può, potenzialmente, appartenere a tutti i cluster); possono essere resi deterministici assegnando il dato al cluster la cui probabilità di appartenenza è massima.

Supponiamo che sia una **mistura di distribuzioni** , con cluster, con un insieme di prior . Ogni dato , viene generato dal nostro modello selezionando una mistura con prior e, supposto che sia stata scelta la r-esima, si genera un dato da ; indichiamo il **modello generativo** con .

Di base, **né i prior né i parametri delle distribuzioni sono noti a priori**, anche se spesso assumeremo che le distribuzioni siano delle gaussiane. La scelta della famiglia di distribuzioni è importante in quanto indice della **comprensione dell'utente sulla distribuzione e la forma dei cluster**.

Dopo aver stimato i parametri delle distribuzioni, è possibile determinare le probabilità di generazione a posteriori dei dati rispetto a ciascun cluster.  
Indichiamo con la pdf della distribuzione . La probabilità di generato da è data dalla **somma pesata delle pdf sulle diverse distribuzioni**, posto che il peso è il prior , secondo la legge .  
Quanto detto, per l'intero dataset, può essere tradotto nella produttoria degli , ossia .  
Per agevolare i costi computazionali, spesso si utilizza la **forma log dell'algoritmo** descritto, che permette di convertire la produttoria in una sommatoria, secondo una relazione del tipo .  
L'**obiettivo** del clustering probabilistico è quello di **massimizzare la log-likelihood**.

Da notare che, se le probabilità dei dati generati fossero note a priori, il calcolo dei parametri ottimali del modello per ciascuna distribuzione sarebbe più semplice.

Possiamo anche definire un algoritmo di **Expectation Maximization iterativo**. Supponiamo di avere vettore che rappresenta **l'intero insieme dei parametri** che descrivono la mistura di distribuzioni, prior inclusi. L'obiettivo è quello di **determinare i parametri** di che **massimizzino la probabilità dei dati complessivi di essere generati dal modello**.  
L'algoritmo si sviluppa in questo modo:

* Si parte con un **scelto arbitrariamente** (il più delle volte sulla base di assegnazioni casuali dati-distribuzioni).
* Passo di **expectation**: si utilizza il disponibile per calcolare la probabilità di generato da ciascuna distribuzione, che poi viene utilizzata per il calcolo della probabilità di Bayes che sia stato generato da
* Passo di **maximization**: dopo aver calcolato le probabilità, **si aggiornano** tutti i valori di al fine di **massimizzare la log-likelihood** sulla base delle nuove assegnazioni. Per far ciò, bisogna **eguagliare a zero la derivata parziale della log-likelihood** rispetto ai corrispondenti parametri del modello.

Gli step 2 e 3 vengono eseguiti ripetutamente al fine di migliorare il criterio di massima verosimiglianza.

15.5 - Clustering per Densità e per Griglia

Uno dei problemi degli algoritmi probabilistici e di quelli basati sulla distanza è che **la forma dei cluster è**, in qualche modo, **definita implicitamente**. Per ovviare a questi problemi, si possono sfruttare **algoritmi per densità**, che prima individuano le **regioni "dense" di dati** e poi compongono i cluster a partire da questi "blocchi costituenti".

Quasi tutti i metodi basati sulla densità possono essere considerati **algoritmi gerarchici a due livelli**, dove la seconda fase è quella che permette di effettuare un maggiore post-processing per la determinazione delle forme complesse dei cluster.

Per i **metodi a griglia**, quello che cambia, rispetto ai metodi per densità, è il modo di definire i blocchi costituenti, che in questo caso saranno delle **griglie nello spazio dei dati**.

15.5.1 - Algoritmi a Griglia

In questa famiglia di algoritmi, i dati vengono **discretizzati** in intervalli ad ampiezza costante, che porta ad avere ipercubi per dati d-dimensionali.  
Per determinare gli ipercubi "densi", **si stabilisce una soglia**, per cui certi ipercubi saranno considerati e altri no; il cluster risultante sarà dato da diversi ipercubi collegati da un lato o un angolo (regioni connesse).

Risulta evidente, quindi, come la scelta della dimensione degli intervalli e della soglia di densità sia di fondamentale importanza, dato che:

* **Celle troppo grandi** possono contenere punti di **diversi cluster**, mentre **celle troppo piccole** possono generare un **numero molto elevato di celle vuote**.
* Una **soglia troppo bassa** fa collassare **più celle in un cluster**, mentre una **soglia troppo alta** tende a **spezzare i cluster**.

L'altro problema di fondo è la **dimensionalità dei dati**, dato che il numero di ipercubi cresce esponenzialmente al variare di essa, il ché sconsiglia di utilizzare algoritmi a griglia in situazioni ad elevata dimensionalità.

15.5.2 - Algoritmi Basati sulla Densità

Il principale algoritmo basato sulla densità di cui parliamo è DBSCAN. Questo algoritmo si basa su un principio simile agli algoritmi a griglia, ma in questo caso i blocchi costituenti sono ottenuti considerando la **densità dei dati**. Inizialmente ogni dato fa da blocco costituente ed è caratterizzato dal concetto di densità, definito come dal numero di punti che si trovano entro un certo raggio da quel blocco.  
La densità dei punti permette di classificarli in tre categorie:

* **Core points**, se contiene **almeno** punti nel suo intorno di raggio .
* **Border points**, se contiene **meno di** punti nel suo intorno, **ma contiene almeno un core point**.
* **Noise points**, se non è né un core point né un border point.

Dopo averli classificati, viene costruito un **grafo di connettività dei core points**, in cui ogni nodo del grafo corrisponde a un core point; due nodi vengono connessi solo se si trovano uno nell'intorno dell'altro e, successivamente, **si assegnano i border points alle componenti connesse** sulla base della loro connessione, secondo un approccio di agglomerazione molto simile al metodo single linkage applicato ai core points.  
Il risultato di questo algoritmo è un insieme di cluster (inteso come gruppo di punti).

L'approccio di DBSCAN presenta come problemi:

* un **costo computazionale di ricerca dei core points potenzialmente elevato** di , che può essere ridotto a con apposite strutture di indicizzazione (anche se ciò è valido solo per basse dimensionalità)
* la scelta di un parametro **fissato** comporta problemi nella **gestione di cluster a densità variabile**. In particolare, dopo aver scelto , si può utilizzare come valore di cutoff della distanza di ogni punto dai suoi vicini più vicini.

15.6 - Misure di Bontà del Clustering

Si tratta di criteri per fare cluster validation, e distinguiamo tra **criteri di validazione interni** e **criteri di validazione estern**i:

* i primi sono basati sui dati da analizzare e, dunque, **indipendenti dall'algoritmo** utilizzato;
* i secondi sono **indipendenti dai dati** e vengono applicati principalmente quando il dataset è generato in modo sintetico (cluster noti a priori).

15.6.1 - Criteri di Validazione Interni

Si tratta di criteri che sono formulati prendendo in prestito alcuni aspetti della funzione obiettivo da ottimizzare; di conseguenza, tendono a favorire algoritmi che usano lo stesso tipo di funzione obiettivo. La loro utilità risiede soprattutto quando è impossibile usare un criterio esterno.

Tra i criteri di validazione interni ricordiamo:

* **Somma delle distanze al quadrato dai centroidi** - si determinano i centroidi dei cluster e viene utilizzata la somma dei quadrati delle distanze come funzione obiettivo, associando a piccoli valori di questa una buona qualità dei cluster. Si mostra particolarmente adatta sugli algoritmi che usano distanze, come il k-means, piuttosto che su altri algoritmi, come il DBSCAN.
* **Rapporto delle distanze intra-cluster/inter-cluster** - si campionano coppie di punti del dataset , di cui appartengono ad un certo cluster trovato con l'algoritmo e no. Definite le misure di distanza intra-cluster ed inter-cluster, si procede al calcolo del rapporto, di cui piccoli valori sono associati a un buon clustering.
* **Within Cluster Sum of Squares (WCSS)** - si calcola come . È un criterio che si sposa perfettamente con gli algoritmi che generano **cluster sferici**. Da notare che, all'aumentare del numero di cluster, WCSS diminuisce a prescindere, dato che i cluster saranno generalmente più piccoli.
* **Silhouette Coefficient** - poste la distanza media di un punto dagli altri punti del suo stesso cluster e la distanza media di un punto dagli altri cluster, si calcola come . Valori alti (vicini a 1) di indicano buon clustering, mentre valori negativi indicano un certo mescolamento tra i dati.
* **Misure probabilistiche** - si usa un metodo simile al passo di massimizzazione dell'approccio EM calcolando la log-likelihood di una mistura di distribuzioni che sottenda ipoteticamente i dati. Questa misura è utile quando è nota la forma che i cluster dovrebbero avere.

Come detto in precedenza, queste misure **dipendono fortemente dall'algoritmo di clustering**, che tende a favorire specifiche forme di clustering, che potrebbero non essere quelle reali. Questa criticità rende difficile valutare un confronto tra i diversi criteri per testare la bontà di un algoritmo.

È comunque utile notare come, nonostante le criticità intrinseche di questi criteri, essi siano utilizzabili come strumento per il **tuning dei parametri dell'algoritmo di clustering**. L'idea alla base di questa teoria risiede nel fatto che la variazione delle misure di validazione al crescere dei parametri tende a mostrare un punto di flesso per un dato numero di cluster, oltre il quale si evidenzia un plateau (questo comportamento è particolarmente evidente in WCSS e nel rapporto intra-cluster/inter-cluster).

15.6.2 - Criteri di Validazione Esterni

Si tratta di criteri usati quando sono disponibili le etichette di classe (cosa non sempre possibile).  
Si tende a preferirli ai criteri interni poiché **permettono di evitare forme di errore costante** su più dataset, che sono cosa comune per i criteri interni.

Una criticità di questi criteri risiede nel fatto che le **etichette di classe** sui dati sono sempre **collegate al contesto applicativo** del clustering, pertanto **potrebbero non riflettere i cluster naturali** dei dati.   
Per affrontare questa criticità, specie nel caso in cui il numero di cluster è pari al numero di etichette di classe, è possibile creare una **matrice di confusione** che evidenzi la mappatura tra cluster naturali e cluster determinati dall'algoritmo: una matrice di confusione, in cui **le righe sono i cluster reali e le colonne sono i valori predetti** dall'algoritmo, contiene in ogni cella il numero di occorrenze del dataset corrispondenti contemporaneamente a un cluster reale e a un cluster predetto. La somma lungo le colonne sarà la stessa per ogni riga al variare del numero di cluster ammessi dall'algoritmo, poiché il numero di occorrenze fa sempre riferimento al cluster reale.  
**Quando il clustering è di qualità**, inoltre, **è possibile permutare le righe e le colonne in modo da avere i valori più alti lungo la diagonale** (cosa non possibile altrimenti, dato che la distribuzione dei valori risulta più uniforme).

Bisogna, quindi, definire misure particolarmente rigorose per valutare la qualità della matrice di confusione. Due misure comunemente utilizzate sono:

* **Cluster Purity** - si assume che **un cluster j di alta qualità dovrebbe contenere quanti più punti possibili tutti appartenenti alla stessa classe**, dunque si definisce il numero di punti nella classe dominante come il massimo degli nella matrice su diversi valori di , ossia come (un clustering di qualità produce sempre prossimi alla somma di lungo tutte le righe), quindi si calcola la CP come .  
  Tale misura ha un solo problema: considera come indice di purezza solo le classi dominanti, **ignorando la distribuzione su tutte le altre classi**.
* **Gini Index Class-Based** - per cui **un buon clustering è quello in cui i punti sono concentrati in poche classi**. Si può calcolare per riga/colonna e restituirà valori diversi a seconda della direzione.   
  Lavorando per colonne, definiamo il Gini Index per la colonna come .   
  Definiamo quindi lo Average Gini Index come la media pesata dei su tutte le colonne, laddove i pesi sono gli , ossia come .

L'indice di Gini è strettamente correlato al concetto di **entropia** del cluster , dato che è la probabilità che un punto appartenga al cluster .   
L'entropia così definita è pari a .   
Similmente a quanto fatto con l'indice di Gini, possiamo calcolare un'**entropia complessiva per la classe** come .

16 - Classificatori

La **classificazione** è uno dei tipici problemi del Machine Learning, come detto in precedenza, e consiste, dato un dataset in input, in cui ogni occorrenza è etichettata come appartenente a una classe, nel **saper predire un'etichetta di classe ad ogni occorrenza dei dati di test**, mai visti dall'algoritmo di apprendimento.

Gli algoritmi di classificazione rientrano tra quelli di **apprendimento supervisionato**, in quanto l'apprendimento della struttura dei gruppi avviene per esempi (e non attraverso la loro tendenza al clustering).

Si tratta, forse, della tipologia di algoritmo di machine learning più comune, e si compone tipicamente di due fasi:

* **Fase di addestramento**, in cui viene costruito un modello addestrato a partire da esempi di addestramento.
* **Fase di test**, in cui il modello addestrato viene utilizzato per predire l'etichetta di classe di dati mai visti prima.

Grazie all'eterogeneità di fonti di dati che possono fungere da insieme di addestramento, gli algoritmi di classificazione sono **molto versatili** in termini di contesto applicativo.

In generale, dati n punti nello spazio appartenenti al dataset , questi vengono associati ad un insieme di etichette (da notare che, per , si parla di classificazione binaria e le etichette vengono indicate con o ) tramite un algoritmo di classificazione.

L'algoritmo può essere di **predizione esplicita dell'etichetta** o di **probabilità di appartenenza del punto ad una classe**, che può comunque essere riportato alla prima tipologia usando come etichetta esplicita quella la cui probabilità è massima.

Per un compito di classificazione, esistono diversi modelli (alberi di decisione, support vector machines, ecc.), ognuno con le proprie caratteristiche, ma un problema più importante della scelta del modello avviene a monte, e riguarda la scelta delle caratteristiche più informative per la classificazione.

16.1 - Selezione delle Features

È la **prima fase** del processo di classificazione e consiste nella **scelta delle features** nel dataset **con un certo contenuto informativo**, dato che quelle che danno poca informazione ai fini della classificazione rischiano di essere fuorvianti per il modello e rappresentano un peso inutile in termini di computazioni. Per la selezione delle features esistono tre principali metodologie:

* **Modelli basati su filtro**, che sfruttano un **criterio matematico** per valutare la qualità di una feature o di un sottoinsieme di esse.
* **Modelli wrapped**, che sfruttano un **algoritmo di classificazione** per valutare le performance del vero e proprio algoritmo di classificazione per uno specifico insieme di features, **per poi stabilire quale sia il sottoinsieme di features più utile** ai fini della vera e propria classificazione.
* **Modelli embedded**, che utilizza la **soluzione di un algoritmo di classificazione** per ricavare suggerimenti utili sulle features più rilevanti, che, una volta isolate, possono essere utilizzate per un nuovo addestramento del modello.

16.1.1 - Modelli Basati su Filtro

Come detto, questo tipo di modelli valuta le features in base a un criterio matematico e ha il vantaggio di **tener conto delle ridondanze tra di esse**.  
L'uso di tali metodi ha, però, un problema riguardo ai costi computazionali, dato che, per un problema d-dimensionale, esistono possibili sottoinsiemi di features di cui valutare la bontà. Nella pratica, pertanto, **si tende ad analizzare le features in modo indipendente**, per poi selezionare quelle più discriminanti.

Tra i modelli basati su filtro ricordiamo:

* **Gini Index** - usato per **attributi categorici** (adattabile anche per attributi numerici discreti). Supposti gli possibili valori di un attributo categorico e la frazione di punti contenente il valore dell'attributo e che appartengono alla classe , l'indice di Gini per il valore dell'attributo è definito come .   
  Come visto con il clustering, **valori piccoli** dell'indice di Gini indicano una **maggiore capacità discriminatoria** dell'attributo, dato che punti appartenenti alla stessa classe per lo stesso valore dell'attributo restituiscono .  
  È possibile ricavare un indice di Gini complessivo per tutti i valori dell'attributo attraverso la relazione , dove è il numero di punti che assumono il valore dell'attributo, da cui l'insieme dei punti .
* **Entropia** - utile in quanto sfrutta la **correlazione tra variazioni di entropia e contenuto informativo** tipici della teoria dell'informazione. Posta la frazione di punti contenente il valore dell'attributo e che appartengono alla classe , l'entropia è uguale a .  
  **Valori elevati** dell'entropia implicano **alta mescolanza tra classi diverse**, mentre **un'entropia a 0** indica la **perfetta separazione tra classi**, dunque un alto potere discriminatorio della feature.  
  Come per l'indice di Gini, è possibile definire un'entropia complessiva per l'attributo usando le supposizioni viste in precedenza, ed è pari a .
* **Fisher Score** - progettato per misurare il **rapporto tra separazione media inter-classe e intra-classe** di **attributi numerici**. **Più è alto** il Fisher Score, **più è alto il potere discriminatorio** dell'attributo. Viene definito come , dove e sono, rispettivamente, la media e la deviazione standard dei punti appartenenti alla classe , e dove è la frazione di punti appartenenti alla classe .  
  Il numeratore della formula rappresenta la separazione inter-classe media, mentre il denominatore la separazione intra-classe media.
* **Fisher Linear Discriminant** - può essere pensato come una **generalizzazione del Fisher Score per combinazioni lineari di features**. Lavorando in forma supervisionata, è un metodo che tende a trovare la direzione massima di variazione delle features e, per contro, l'iperpiano perpendicolare che separa meglio le classi rispetto alle features stesse, in modo da massimizzare il già citato rapporto inter-classe/intra-classe.

16.1.2 - Modelli Wrapped

Sono modelli caratterizzati da una **strategia generale**, che consiste nel **perfezionare iterativamente un insieme di features** aggiungendo una feature per volta.

L'algoritmo viene inizializzato con , e si compone di due passaggi:

1. Si aggiungono una o più features ad ;
2. Si utilizza un algoritmo di classificazione per valutare la precisione dell'insieme di features e, in caso negativo, si rigetta l'aggiunta.

Bisogna quindi definire le **possibili strategie** per aggiungere features ad :

* una **strategia greedy** può incrementare l'insieme ad ogni passo aggiungendo sempre la feature più discriminatoria rispetto ad un certo criterio
* usare **campionamento casuale** per aggiungere una feature ad .

Il secondo passo del processo iterativo evidenzia come il successo del metodo sia **dipendente anche dallo specifico algoritmo** che si utilizza.

16.2 - Alberi di Decisione

Si tratta di un metodo di classificazione basato su **decisioni gerarchiche sulle features**, disposte in una struttura ad albero.

È un **algoritmo supervisionato**, in quanto necessita che i dati abbiano affibbiate delle etichette, e fa in modo da suddividere gerarchicamente il dataset in modo che il livello di mescolamento delle features di classe in ogni ramo sia il minore possibile. La suddivisione operata dagli alberi di decisione **ricorda il funzionamento dell'algoritmo di clustering top-down**.

Gli alberi di decisione possono effettuare delle discriminazioni sia su singole features, per cui si parla di **split univariato del nodo**, o su un insieme di attributi, nel qual caso si parla di **split multivariato del nodo**.  
L'uso di **split multivariati** è più potente e **porta ad alberi meno profondi**.

Un algoritmo di induzione ha due tipi di nodo, i **nodi interni** e i **nodi foglia**. **Ogni nodo foglia è etichettato con la classe predominante nel nodo**, mentre un nodo interno speciale e la radice, che corrisponde all'intero spazio delle features.  
Durante la creazione dell'albero, **è possibile separare solo quei nodi che sono combinazione di due o più classi**, mentre essa giunge alla fine nel momento in cui l'algoritmo incontra un criterio di arresto (il più semplice è quello in cui tutti gli esempi in una foglia appartengono alla stessa classe).

La costruzione dell'albero, se troppo dettagliata, può portare ad una situazione di **overfitting**, in cui il modello aderisce esattamente agli esempi di addestramento ed è incapace di generalizzare. Per non incorrere in overfitting, **si può effettuare una potatura dei nodi che sono andati in overfitting** secondo un qualche criterio (es. si valida l'incremento di precisione su un insieme di validazione estratto dal training set).

16.3 - Random Forests

Sono una **generalizzazione degli alberi di decisione** e si tratta di classificatori di tipo ensemble, ossia è composto da più classificatori addestrati sullo stesso dataset che effettuano ognuno la propria predizione e il cui **risultato è una combinazione** di queste, atta a generare una predizione globale più robusta. Fanno uso della tecnica del **bagging**, ossia:

1. **Si crea un ensemble di classificatori indipendenti** e distribuiti identicamente, in modo da ridurre la varianza delle stime all'aumentare del numero di stime effettuate.
2. **Si generano dataset con campionamento con rimpiazzo**, così da poter generare dataset quanto più simili possibili.
3. **Si addestrano i classificatori** e la predizione della foresta è quella che ha ottenuto la maggioranza dei voti o la media delle predizioni espresse da ciascun classificatore su ogni campione.

Data questa cooperazione, la random forest è molto più potente di un singolo albero di decisione. Un'altra caratteristica interessante è che **i nodi più elevati della gerarchia sono**, di base, **invarianti**, in quanto la particolarizzazione degli alberi avviene nei nodi più interni, in modo da non ottenere solo copie dello stesso albero. Per ottenere questa diversificazione, le random forests fanno uso di **specifici algoritmi per lo split casuale** nei diversi alberi dell'ensemble. Tra i criteri di split abbiamo:

* **Forest-RI (Forest - Random Input)**, in cui viene estratto un sottoinsieme casuale di features , con che viene usato come regolatore della casualità introdotta nel criterio di split.  
  Lo split in ogni nodo è preceduto dalla selezione casuale di un sottoinsieme di attributi di dimensione , per cui gli split tengono in considerazione solo S. Più è piccolo il valore di , minore sarà la correlazione tra gli alberi, ma minore sarà la precisione dell'albero.  
  È stato dimostrato che, per dimensioni, il miglior compromesso è selezionare features.
* **Forest-RC (Forest - Random Combination)**, in cui viene estratto un sottoinsieme di features e si creano combinazioni lineari da usare come features multivariate per lo split.  
  Per ogni nodo le features vengono selezionate casualmente e combinate linearmente con coefficienti generati uniformemente nell'intervallo .

Usando algoritmi di random forest, non si fa uso di una strategia di potatura esplicita, per cui ogni albero cresce cercando di addestrarlo con dati di addestramento atti a massimizzare la riduzione di varianza. Tuttavia, a causa dei criteri limitati di random split degli alberi, i singoli costituenti potrebbero mostrare un bias maggiore, molto marcato nel momento in cui la frazione di features significative è molto piccola.

16.4 - Classificatori Probabilistici

Si tratta di un modello che quantifica la relazione tra le features e la classe obiettivo in termini di probabilità. I due modelli più diffusi sono:

* **Classificatori di Bayes**, in cui si usa la regola di Bayes per modellare la probabilità di ciascun valore della variabile target per features date. In genere, si assume che i punti all'interno di una classe siano generati secondo una specifica distribuzione di probabilità, come Bernoulli o quella multinomiale.  
  Tra questi classificatori, i **Bayes naive** assumono features condizionate dalla classe statisticamente indipendenti, in modo da semplificare la modellazione (anche se ciò non sempre corrisponde alla realtà).
* **Regressori logistici**, che assumono la variabile target come tratta da una distribuzione di Bernoulli la cui media è definita da una funzione parametrizzata sulle features.  
  Pertanto, la distribuzione di probabilità della variabile target è una funzione parametrizzata delle features.

Il primo tipo di classificatore prende il nome di **classificatore generativo**, mentre il secondo viene definito **classificatore discriminativo**.

16.4.1 - Bayes Naive

Sia la variabile aleatoria target che rappresenta la classe di un esempio di test, la cui feature d-dimensionale ha valori , potendo scrivere i valori sulla singola feature come ; la probabilità condizionata di appartenenza a una classe () dato a valori fissati è, secondo il teorema di Bayes

Per l'assunzione naive di questo classificatore, le componenti della feature sono statisticamente indipendenti, per cui vale

Da cui deriviamo che

In questo modo, si è semplificato il calcolo di una probabilità condizionata e congiunta al calcolo di più probabilità condizionate, che possono essere espresse in termine frequentista come .  
Il problema di quest'ultima relazione risiede nel fatto che, per piccole dimensioni del dataset, fa tendere l'intera probabilità del classificatore a 0, dunque bisogna introdurre un fattore di smoothing laplaciano, per cui la relazione diventa , dove è il parametro di smoothing, mentre è il numero di valori distinti della feature j-esima.  
Questa aggiunta fa in modo che, per valori nulli, il rapporto sia uguale a , una stima ragionevole in assenza di dati di addestramento sulla classe .

16.4.2 - Regressori Logistici

Nella forma più semplice, si assume che la variabile target sia binaria con valori .   
Sia un vettore di parametri -dimensionale, in cui è un **parametro di offset** e il **coefficiente relativo alla i-esima dimensione** nei dati. Per un record la probabilità che la variabile ,rappresentante la classe, abbia valore -1 o 1 è modellata dalla funzione logistica

e

Un regressore logistico **può essere visto come un classificatore lineare** in cui rappresenta il vettore dei coefficienti dell'iperpiano di separazione tra le due classi, laddove i punti che fanno assumere valori positivi all'equazione dell'iperpiano saranno assegnati alla classe positiva, e viceversa.

Per stimare i parametri più adatti del modello di regressione logistica viene utilizzato l'approccio a massima verosimiglianza.   
Siano e i segmenti di dati appartenenti alle classi positiva e negativa. Se indichiamo con il k-esimo punto, la funzione di verosimiglianza per il dataset viene definita come

Poiché l'obiettivo è **massimizzare la funzione di verosimiglianza**, dobbiamo lavorare con le derivate parziali della funzione, che altro non sono che i parametri , ed eseguire un algoritmo di ascesa del gradiente. Per semplificare i costi computazionali, conviene sempre **lavorare con la log-likelihood associata**, per cui abbiamo che, la derivata parziale di rispetto al parametro è data da

che altro non sono che le **somme degli errori di classificazione commessi** per le due classi.

La **condizione di aggiornamento** per il parametro è

dove è il passo di aggiornamento, determinabile usando una ricerca binaria per massimizzare il miglioramento della funzione obiettivo.

16.5 - Support Vector Machines

Si tratta di classificatori che permettono di **determinare l'iperpiano di separazione ottimale** tra due classi, ossia quello che massimizza il margine tra i due iperpiani paralleli che delimitano le classi, i cosiddetti **support vectors**.

Nel caso di classi linearmente separabili, è possibile determinare un iperpiano lineare che separi nettamente i dati delle due classi.

16.5.1 - SVM per Dati Linearmente Separabili

Di base, per dati linearmente separabili, è possibile individuare infiniti iperpiani separatori lineari. Lo scopo di una SVM è quello di determinare l'iperpiano **la cui distanza minima da entrambe le classi sia la massima possibile**, in quanto ciò genera un classificatore molto robusto. La distanza minima cui si fa riferimento è modellabile grazie al concetto di **margine** dell'iperpiano.

Consideriamo un iperpiano separatore tra due classi, il margine è definito come la **somma delle sue distanze dai punti più vicini** di ciascuna classe. Supponiamo, allora, di costruire due iperpiani paralleli che passano per i punti di addestramento di entrambe le classi (che, in questo senso, prendono il nome di vettori di supporto) e di definire la distanza tra di loro come margine: di fatto, **l'iperpiano ottimale si troverà esattamente al centro** tra i due iperpiani.

Per determinare l'iperpiano di margine massimo, possiamo massimizzare il margine espresso in termini di coefficienti dell'iperpiano di separazione. Se indichiamo gli punti del training set con , dove è un **vettore d-dimensionale corrispondente all'i-esimo punto** e è la **classe binaria associata all'i-esimo punto**, possiamo definire l'iperpiano di separazione con la forma , dove è un vettore d-dimensionale rappresentante la **direzione normale** al piano e uno scalare detto **bias**, con il primo che indica l'orientamento dell'iperpiano e il secondo che indica la sua distanza dall'origine.  
I coefficienti individuati nell'equazione dell'iperpiano devono essere appresi per **addestramento** in modo da **massimizzare il margine**. Se le classi sono linearmente separabili, l'iperpiano di separazione esiste certamente, e tutti gli associati a soddisferanno l'equazione , mentre quelli per cui soddisferanno l'equazione .  
Avendo detto che, se esiste, l'iperpiano si trova al centro tra i due vettori di supporto e ha equazione , possiamo esprimere i vettori di supporto in termine di un parametro che regola la distanza tra di essi, secondo le equazioni

e

che, con variabili opportunamente scalate, possono essere riscritte come

e

Quelle appena scritte sono le **equazioni dei vincoli di margine** e, avendo detto che le classi sono linearmente separabili, nello spazio tra di esse non vi sono dati con classificazione indecisa. Ciò permette di esprimere le condizioni di appartenenza a una o all'altra classe come

Queste condizioni possono essere scritte in **forma compatta** come .

Avendo ricavato i vettori di supporto, grazie all'algebra lineare, possiamo affermare che **la distanza tra di essi è** da cui, poiché la differenza tra i termini costanti delle due equazioni è 2, il margine è pari a , che altro non è che la funzione da massimizzare.

In realtà, massimizzare la forma equivale a minimizzare la funzione , una forma quadratica che può essere risolta con **rilassamento lagrangiano**, la cui risoluzione viene omessa, che portano a una definizione dei parametri direttori del piano e a una derivazione di da cui si ricava che **solo i vettori di supporto sono utilizzati per trovare la soluzione**.

Il duale lagrangiano può essere ottimizzato attraverso la tecnica di ascesa lungo il gradiente in termini del parametro vettoriale n-dimensionale

In modo analogo alla regressione logistica, **l'equazione di aggiornamento** basata sul gradiente è la seguente

dove è il **tasso di apprendimento**. Come soluzione iniziale si può considerare il vettore nullo, che rientra nello spazio delle soluzioni.

16.5.2 - SVM con Soft Margin per Dati non Linearmente Sep.

Nella maggior parte dei casi reali, i dataset contengono dati non linearmente separabili, ma ciò non vieta di poter classificare usando una SVM: è possibile, infatti, costruire un iperpiano di separazione che classifica ottimamente la stragrande maggioranza dei dati, a meno di poche eccezioni.

La costruzione di un iperpiano con SVM per classi non linearmente separabili fa uso di un **sistema di penalità** che compensano la violazione dei vincoli di margine. Solitamente, il livello di violazione di ciascun vincolo da parte del punto è dato da una **variabile di rilassamento** , per cui possiamo definire il nuovo **insieme di soft margin** sui vettori di supporto come

Indicati con e due parametri di rilassamento del modello, possiamo affermare che i punti che violano i vincoli **penalizzano il modello** con un fattore . Tipicamente si pone , per cui possiamo riformulare la funzione obiettivo **hinge loss** come , che è, ancora una volta, un problema risolvibile con rilassamento lagrangiano.

Si dimostra che **le soluzioni al problema** per e **non risentono dei fattori di penalizzazione** e sono, pertanto, identici al risultato con hard margin, così come il duale lagrangiano, in quanto **i termini lineari che coinvolgono** **sono nulli**.

16.6 - Valutazione della Bontà della Classificazione

Per la classificazione il concetto di precisione si può applicare a diversi indicatori, per cui dobbiamo dare due definizioni preliminari:

* Si definiscono **problemi metodologici** quelli **legati alla divisione dei dati** etichettati in segmenti di dati di addestramento e dati di test, e quindi legati alla metodologia usata per tale divisione, che impatta la valutazione.
* Si definiscono **problemi di quantificazione** quelli legati alla **scelta di una misura numerica** per valutare la bontà del metodo.

La misura più semplice è la **accuracy**, ossia una misura che esprime **quanto le classi predette sono uguali alle classi corrette** del dataset. Posta la classe predetta del dato e detto lo spazio delle classi, la accuracy si esprime come

, .

Per evitare che le classi risultino sbilanciate nella valutazione della accuracy, si può utilizzare la **versione bilanciata** della stessa, che viene espressa come

La bontà della predizione può essere valutata anche attraverso una **matrice di confusione**, che nel caso binario si configura come una matrice , così fatta

in cui le righe rappresentano la **classe reale** e le colonne fanno riferimento alla **classe predetta**. L'ideale sarebbe avere i valori fuori dalla diagonale uguali a 0, perché sarebbe sintomo di classificazione perfetta, posto che i positivi e che i negativi . Tra le misure che possiamo utilizzare abbiamo:

* **Precision** - rapporto tra e la somma . Misura **quanto l'algoritmo riesce ad individuare una certa classe**.
* **Recall** - rapporto tra e la somma . Misura **quanto l'algoritmo riesce a separare una classe dalle altre** (valori alti indicano l'incapacità dell'algoritmo di classificare altre classi in presenza di una certa classe).
* **F1-Score** - rapporto tra il prodotto di precision per recall e la loro somma, moltiplicato per 2.

Un'altra misura è l'area sotto la **curva AUC**, ovvero l**'area sottesa dalla curva ROC**, che illustra la capacità diagnostica di un sistema di classificazione binario al variare della sua soglia di discriminazione.  
**La ROC mette a confronto i tassi di falsi positivi e positivi reali**, laddove il tasso dei falsi positivi è pari alla differenza tra 1 e **precision**, mentre il tasso dei positivi reali è la quantità di veri positivi rispetto al totale, e coincide con il **recall**.