1)Quali sono i principali paradigmi del machine learning? Se ne riporti una descrizione sintetica, chiarendo quali siano le principali differenze, con particolare enfasi per il caso del supervised learning. Si distinguano in particolare classificazione e regressione.

I principali paradigmi del machine learning sono: **Supervised Learning, Unsupervised Learning e Reinforcement Learning**.

- Il **supervised learning** agisce in base all'attività: l'algoritmo prevede il comportamento di un agente, utilizzando l'esperienza del passato tramite principalmente algoritmi di regressione e classificazione.
  L'obiettivo è che l'algoritmo fornisca una risposta corretta a ogni esempio nel dataset.
  Presuppone che i dati siano delle coppie (x,y) e ci sia una funzione di apprendimento che cattura le informazioni da ogni esempio.
  Quindi viene costruita una funzione h che data in input una x deve produrre il corrispettivo y correttamente. Quindi all'inizio deve mappare, dai dati forniti, possibilmente tutti i tipi di x con la corrispettiva y dal dataset fornito per il training.
  L'output è diverso in base all'algoritmo implementato che normalmente può essere un algoritmo di classificazione o regressione.
  L'esperienza fornita dal dataset da sola non ci permette di fare previsioni sulle istanze di dati non visti.
- Il **unsupervised learning** agisce in base ai dati: l'algoritmo individua similitudini e strutture nascoste all'interno dei dati (clustering).
  Per cui il goal è trovare la regolarità o patterns sui dati.
  Vengono forniti degli esempi x, e si cercano le regolarità su tutto il dominio dell'input
- Il **reinforcement learning** agisce in base all'ambiente: l'algoritmo impara a reagire all'ambiente e a tenere comportamenti intelligenti.
  La differenza tra i paradigmi precedenti è che c'è uno scenario di agenti autonomi, abbiamo uno scenario attivo e non invece solo un dataset fisso da cui apprendere in modo statico. Si osserva l'ambiente e si apprende da esso e da ciò che succede, si va a misurare come l'agente si muove nell'ambiente in modo corretto o sbagliato.
  L'obiettivo è ottimizzare la funzione, ovvero le task che deve produrre l'agente.
  L'agente ha degli stati in cui può essere e delle azioni che può eseguire nell'ambiente e ha delle "reward" in base alle azioni che esegue che possono essere positive, neutrali o negative da cui apprendere.

Un algoritmo di **classificazione** separa i dati in due o più classi. Quando fornisco un esempio al classificatore, l'algoritmo mi restituisce la classe a cui potrebbe appartenere. Gli output categorici sono chiamati labels o classi, e possiamo avere classificazioni che possono essere **binarie** o **multi-class** ovvero con 3 o più classi da individuare. Inoltre abbiamo classificatori **multi-label** in cui andiamo ad allenare un nostro classificatore a riconoscere un determinato tipo di classe **individualmente**. Per esempio da un immagine di una piazza, dobbiamo riconoscere in una unica immagine in comune diversi oggetti.
I classificatori si differenziano in:

- **Lineari:** sono semplici e veloci ma risento del problema dell' **underfiting;**
  Uno spazio, come per esempio un iperpiano che viene separato da una linea che indica il confine tra i due spazi.
- **Non lineari:** sono più precisi ma più lenti da elaborare e c'è sempre il rischio di cadere nell' **overfitting;**

Un algoritmo di **Regressione** si basa sull'interpolazione dei dati per associare tra loro due o più caratteristiche (**feature**). Quando fornisco all'algoritmo una caratteristica in input mi restituisce l'altra caratteristica e come per i classificatori, per i regressiori abbiamo regressori lineari e non lineari.
La regressione **lineare** è un approccio **lineare** che modella la relazione **tra** una variabile dipendente e una o più variabili indipendenti. Non è molto utile utilizzarlo per problemi di classificazione.
La regressione **non lineare** ha una accuratezza del modello previsionale più alta rispetto ai regressori lineari perché la stima è una curva o uno spazio curvo. Tuttavia si rischia di cadere in casi di **overfitting.**
Essa è definita dalla notazione: hteta(x) = theta * x = theta trasposto * x.
La regressione **logistica** è un modello **statistico** che predice la probabilità di un risultato che **può avere solo due valori.**
La funzione di predizione h(x) produce risultati compresi tra 0 e 1. Questo permette di considerare questo valore come una probabilità, tramite spesso una funzione come la sigmoide, che è ideale per task di classificazione.

2) Cosa si intende per "one learning algorithm hypothesis" e come tale ipotesi si relaziona con le reti neurali artificiali? Si fornisca inoltre una descrizione esaustiva degli elementi/ingredienti principali che permettono la definizione di una rete neurale multistrato.

Ci sono evidenze che il cervello umano utilizzi lo stesso algoritmo di apprendimento per processare diversi input, i quali possono essere per esempio la vista, l'udito e il tatto attraverso i neuroni.

L'idea è che se prendiamo l'area del cervello adibita all'orecchio, ovvero i suoi neuroni e i suoi collegamenti e la potessimo tagliare e ricollegare per esempio per usarla per un altro senso come la vista, il cervello col tempo si riconfigura, cercando di interpretare il nuovo segnale che gli arriva per svolgere il nuovo compito assegnato.

L'idea della **rete neurale** è questa, non c'è bisogno di creare 100 algoritmi per 100 compiti diversi, ma invece creare un **modello semplice** che poi verrà adattato al compito assegnato, ma ovviamente questa è un'idea filosofica.

La più piccola unità di una rete neurale è il **neurone**, esso ottiene molteplici dati in input e tramite una funzione matematica produce un output.

Il neurone è composto da: gli **input** che idealmente rappresentano i dentriti del neurone umano, l'elaborazione effettuata tramite funzioni matematiche che rappresentano il nucleo e l'output che corrisponde all'assone del neurone umano.

Per esempio, storicamente il primo esempio è stato il **percettrone** il quale tramite una combinazione lineare htheta(x) dava in output 1 se theta trasposto * x era maggiore di 0, 0 altrimenti. La funzione applicata ad x, può essere per esempio la **Logistic Unit**, ma anche la **Sigmoide** o la **Tahn,** ma anche tante altre.

Idealmente la rete neurale è un gruppo di diversi neuroni con forti collegamenti tra di essi. Tendenzialmente avremo 3 livelli: **input layer, hidden layer e output layer.**

L'**architettura** della rete neurale non è altro che una scelta dei 3 tipi di neuroni descritti sopra. Idealmente l'unica scelta architetturale importante è sull'hidden layer, poiché l'input, che sono le features dipendono dall'hardware disponibile e dalla complessità del problema e l'output normalmente è già stabilito a priori.

Nell'hidden layer scegliere quanti neuroni ha ogni strato, muoversi in profondità e quanti strati porre, ovvero muoversi in larghezza, significa apportare nuovi parametri e non è una scelta facile.

Idealmente ogni neurone è una funzione, la quale prende in input il risultato del neurone precedente, tranne per il primo strato che prende l'input, e lo processa e produce un output che sarà consegnato a un altro gruppo di neuroni. Per cui vi è una **gerarchia** ed è per questo che è utile avere milioni di neuroni, perché così l'input iniziale verrà processato idealmente milioni di volte prima di essere restituito dalla nostra rete neurale.

Riguardo all'input di un neurone/percettrone, oltre a quello gia descritto si ricorda che per tutti i neuroni oltre all'input ricevuto da altri neuroni o dall'input, vi è il **bias unit,** cioè un bias da applicare all'operazione che svolge il neurone. Per esempio, se il neurone svolge una sommatoria la quale serve a svolgere la operazione della porta logica & tra solo due input corrisponderà a:

Riceve da due input ognuno cosi: +20 se l'input è 1 o 0 se l'input è 0 e avrà come **bias unit** sempre in input ma non da un output di un altro neurone, il valore -30 il quale sommato ai due input ricevuti, se il risultato è >> 0 significa che ha ricevuto due 1 in input restituisce 1, se il risultato è << 0 significa che almeno un input è 0 e allora restituisce 0. Per cui questo **bias unit** serve a svolgere le operazioni dei neuroni.

Prendendo un esempio banale come l'operazione logica XNOR la quale corrisponde all'unione delle operazioni logiche & ,NOT e OR, essa suddivide il piano cartesiano in quattro parti, e questa suddivisione non può essere rappresentata da un singolo neurone con classificazione lineare poiché il singolo neurone non è capace di svolgere tutte queste operazioni logiche insieme.

La soluzione è **combinare** i neuroni con compiti diversi e l'output combinato dai due che effettuano due operazioni matematiche diverse rappresenterà l'output desiderato.

È importante sottolineare che generalmente, prendendo d'esempio un problema di classificazione non lineare e dei neuroni con classificazione lineare, se si prende individualmente il risultato dei singoli neuroni, la classificazione sarà terribile, ma collaborando insieme per risolvere un problema complesso, produrranno una suddivisione ottimale risolvendo, sempre per esempio, per un problema non lineare.

Per cui una rete neurale, non è altro che un insieme di tanti neuroni "stupidi" che svolgono task intermedie, che possono fare operazioni uguali ma con parametri diversi, e tutti gli output vengono sommati insieme per produrre una soluzione a un problema complesso.

3) Si descriva in modo dettagliato il modello di logistic regression (con regolarizzazione), le sue principali caratteristiche ed il contributo dei diversi elementi presenti nella funzione di costo. Si riporti infine una descrizione accurata delle differenze di tale modello rispetto ad un semplice classificatore lineare, anche mediante esempi qualitativi.

La regressione logistica è un algoritmo di apprendimento supervisionato che costruisce un modello probabilistico lineare di classificazione dei dati. E' usata nel machine learning per l'addestramento di un algoritmo nella classificazione supervisionata dei dati usato per stimare due risultati e solo due.

L'algoritmo stima la probabilità di occorrenza di un evento adattando i dati a una funzione logistica.

Essa è ottenuta dall'espressione:

z= theta trasposto * x

1 / (1+e^(-z))

La funzione **di costo** per la logistic regression è la seguente:

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \cdot \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \cdot \log(1 - h_\theta(x^{(i)}))\right]$$

Con regolarizzazione diventa:

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \cdot \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \cdot \log(1 - h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}$$

Nella fase di addestramento l'algoritmo di regressione logistica prende in input n esempi da un insieme di training e ogni singolo esempio è composto da m attributi e dalla classe corretta di appartenenza.

Durante l'addestramento l'algoritmo elabora una distribuzione di pesi (W) che permetta di classificare correttamente gli esempi con le classi corrette.

Poi viene calcolata la combinazione lineare z del vettore dei pesi W e degli attributi e questa combinazione lineare viene passata alla **sigmoide** che calcola la probabilità di appartenenza del campione alle classi del modello.

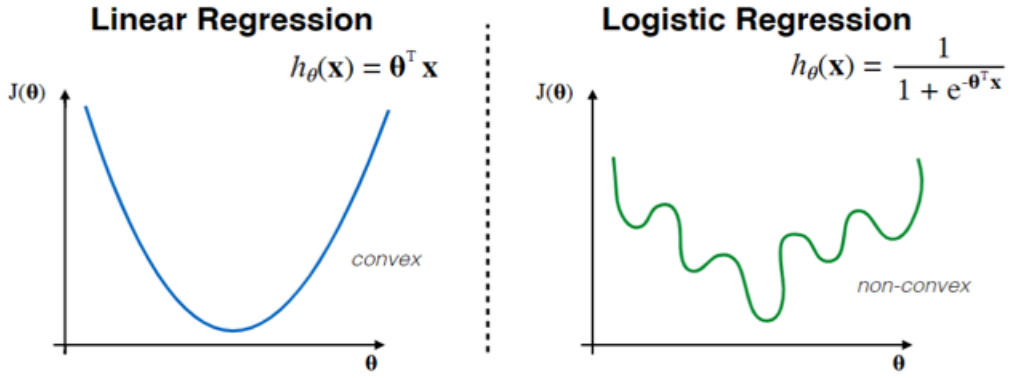
La **REGOLARIZZAZIONE** è la metodologia per mantenere un modello con un numero immenso di variabili e quindi un modello dettagliato, ma pesando diversamente le variabili durante la fase di dataset per calibrare i pesi e non cadere in casi di overfitting.

La regolarizzazione funziona aggiungendo la penalità associativa ai valori di coefficiente all'errore di ipotesi.

Il valore penalizzatore **lambda** viene deciso/calcolato tramite metodo di **gradient descent** il quale si aggiorna con una velocità decisa, fino a trovare il minimo locale in un determinato numero di iterazioni.

La differenza tra regressione logistica e lineare è ampia.

Come gia detto la regressione logistica è un modello di classificazione, mentre la regressione **lineare** è un algoritmo regressore che restituisce come risposta un altro valore tramite l'interpolazione dei dati.



L'update del gradiente è esattamente la stessa sia per la regressione lineare che per quella logistica.

Riguardo alla Linear Regression dato in esempo un prodotto cartesiano con dei punti disposti nel piano, la nostra linear regression produce un modello basato sul costo totale che è una linea, corretta ma inefficiente.

La logistic regression o **sigmoide** è un approccio decisamente migliore rispetto alla linear regression, nell'implementazione di classificatori in quanto essa è dotata di una peculiare forma (quella di una S) e la proprietà di avere il hteta(x) contenuto tra 0 e 1.

Nella quale theta(0) sarà il punto di flesso mentre theta(1) sarà la velocità (inclinazione) con la quale la funzione passerà da 0 (sinistra) ad 1 (destra).

Se i dati sono uniformemente distribuiti, e si sta utilizzando la tecnica dei **Least-squares** è possibile notare un comportamento molto simile alla funzione lineare, invece se si va a creare uno sbilanciamento nei dati, notiamo che la lineare(volendo essere troppo giusta) crea un errore, mentre la sigmoide non è affetta da questo problema grazie alle proprietà sopra descritte.

Il **difetto** della logistic regression è che la funzione non è convessa come quella lineare ma tende ad essere più ondulata.

4) Si descriva dettagliatamente la procedura di model selection(aiutandosi con un esempio concreto) e si fornisca una chiara giustificazione teorica/concettuale a tale procedura.

La selezione del modello è il compito di selezionare un modello statistico da un insieme di modelli candidati.

Per far ciò è noto usare l'**Hold Out**, ovvero dato il nostro dataset, esso viene suddiviso generalmente in **Training Set** per il 70%, **Validation Set** per un 15% e **Test Set** per il restante 15%, ovviamente queste percentuali possono differire, ma si consiglia una suddivisione tale.

Il **Training Set** sono i dati forniti alla macchina per allenare e addestrare modelli con diversi valori per i parametri.

Per cui i nostri modelli aggiornano i parametri in base ai dati forniti, e visto l'importanza, vi è la necessità che il Training Set sia di dimensione non piccola, sia diversificato equamente negli esempi forniti per non cadere in casi di overfitting e cercare di coprire in termini assoluti tutte le istanze possibili, anche se nella realtà è molto improbabile avere un dataset tale.

Il **Validation Set** è un set fittizio usato durante il processo di training, nel quale abbiamo esempi che il nostro modello non ha mai visto per vedere se il nostro modello in questione riesce a prevedere correttamente il compito che deve eseguire e aggiornare i parametri per migliorare l'errore dei modelli creati.

Questo Set ci fornisce gia il modello tra tutti quelli che disponiamo, il quale risponde meglio alle richieste, ma per non cadere in casi di overfitting sul Validation Test utilizziamo il 15% dei dati rimanenti per effettuare un altro Test.

Questi dati fanno parte del **Test Set,** i quali come gia descritto, determineranno con esempi mai visti dal modello, il modello migliore tra tutti e confermerà o meno ciò gia visto con il Validation Set.

Questi dati sono normalmente un indice di affidabilità dato che questi dati non sono solitamente a disposizione di chi crea il modello in quanto è un dataset usato solo per una verifica se il modello scelto è efficiente.

FORSE BISOGNA PARLARE DI:

Questa suddivisione, e questi Test fatti, cercano di simulare un problema noto e importante ovvero il problema dello **Scenario Vero.**

Ovvero noi alleniamo gli algoritmi tramite dei nostri dataset, ma concretamente non sappiamo come realmente i dati sono distribuiti.

Qui nasce l'**Empirical Risk Minimization** che definisce una famiglia di algoritmi apprenditivi e si danno dei limiti teorici di performance.

Questo perché prendendo in esempio un algoritmo per la valutazione delle case, se forniamo un dataset di un milione di case, queste saranno comunque una piccola parte rispetto alla realtà anche se il dataset in termini è grande.

Il **True Error** lo potremmo avere soltanto se il nostro dataset comprendesse tutti i casi possibili; quindi, nella realtà preso d'esempio il True Error come uno spazio nel piano cartesiano come un uovo, la nostra funzione h dovrà avere un proprio spazio nel piano più verosimile al True Errore, cosa comunque difficile se non impossibile da equiparare.

Un altro problema noto è che i dati che molte volte abbiamo a disposizione sono distribuiti a "coda lunga", ovvero avremo molti esempi per dei determinati tipi, e altri in cui ne avremo pochi, questo creerà un sbilanciamento nei dati che può portare il modello a preferire determinate categorie o esempi, rispetto ad altri in modo arbitrario e ingiusto. Un esempio di questo problema è il riconoscimento di una sposa, per il quale avremo molti esempi di spose col vestito bianco, quando per esempio in India le spose per usanza diversa, sono colorate e tendenzialmente avremo meno esempi e questo comporta al modello di non riconoscere le spose indiane se vengono forniti i dataset in modo bilanciato.

CURVE DI APPRENDIMENTO PER VALIDATION SET E TRAINING?

5) Spiegare in dettaglio gli elementi fondamentali del perceptrone, più in generale, delle reti neurali.

Si riporti inoltre una breve descrizione di come tale modello possa essere esteso mediante la realizzazione di un'architettura a più strati, fornendo un esempio che evidenzi le differenze/vantaggi di tale architettura.

La più piccola unità di una rete neurale è il **neurone**, esso ottiene molteplici dati in input e tramite una funzione matematica produce un output.

Il neurone è composto da: gli **input** che idealmente rappresentano i dentriti del neurone umano, l'elaborazione effettuata tramite funzioni matematiche che rappresentano il nucleo e l'output che corrisponde all'assone del neurone umano.

Per esempio storicamente il primo esempio è stato il **percettrone** il quale tramite una combinazione lineare htheta(x) dava in output 1 se theta trasposto * x era maggiore di 0, 0 altrimenti. La funzione applicata ad x, può essere per esempio la **Logistic Unit**, ma anche la **Sigmoide** o la **Tahn,** ma anche tante altre.

Idealmente la rete neurale è un gruppo di diversi neuroni con forti collegamenti tra di essi. Tendenzialmente avremo 3 livelli: **input layer, hidden layer e output layer.**

L'**architettura** della rete neurale non è altro che una scelta dei 3 tipi di neuroni descritti sopra. Idealmente l'unica scelta architetturale importante è sull'hidden layer, poiché l'input, che sono le features dipendono dall'hardware disponibile e dalla complessità del problema e l'output normalmente è già stabilito a priori.

Nell'hidden layer scegliere quanti neuroni ha ogni strato, muoversi in profondità e quanti strati porre, ovvero muoversi in larghezza, significa apportare nuovi parametri e non è una scelta facile.

Idealmente ogni neurone è una funzione, la quale prende in input il risultato del neurone precedente, tranne per il primo strato che prende l'input, e lo processa e produce un output che sarà consegnato a un altro gruppo di neuroni. Per cui vi è una **gerarchia** ed è per questo che è utile avere milioni di neuroni, perché così l'input iniziale verrà processato idealmente milioni di volte prima di essere restituito dalla nostra rete neurale.

Riguardo all'input di un neurone/percettrone, oltre a quello gia descritto si ricorda che per tutti i neuroni oltre all'input ricevuto da altri neuroni o dall'input, vi è il **bias unit,** cioè un bias da applicare all'operazione che svolge il neurone. Per esempio, se il neurone svolge una sommatoria la quale serve a svolgere la operazione della porta logica & tra solo due input corrisponderà a:

Riceve da due input ognuno cosi: +20 se l'input è 1 o 0 se l'input è 0 e avrà come **bias unit** sempre in input ma non da un output di un altro neurone, il valore -30 il quale sommato ai due input ricevuti, se il risultato è >> 0 significa che ha ricevuto due 1 in input restituisce 1, se il risultato è << 0 significa che almeno un input è 0 e allora restituisce 0. Per cui questo **bias unit** serve a svolgere le operazioni dei neuroni.

Prendendo un esempio banale come l'operazione logica XNOR la quale corrisponde all'unione delle operazioni logiche & ,NOT e OR, essa suddivide il piano cartesiano in quattro parti, e questa suddivisione non può essere rappresentata da un singolo neurone con classificazione lineare poiché il singolo neurone non è capace di svolgere tutte queste operazioni logiche insieme.

La soluzione è **combinare** i neuroni con compiti diversi e l'output combinato dai due che effettuano due operazioni matematiche diverse rappresenterà l'output desiderato.

È importante sottolineare che generalmente, prendendo d'esempio un problema di classificazione non lineare e dei neuroni con classificazione lineare, se si prende individualmente il risultato dei singoli neuroni, la classificazione sarà terribile, ma

collaborando insieme per risolvere un problema complesso, produrranno una suddivisione ottimale risolvendo, sempre per esempio, per un problema non lineare.

Per cui una rete neurale, non è altro che un insieme di tanti neuroni "stupidi" che svolgono task intermedie, che possono fare operazioni uguali ma con parametri diversi, e tutti gli output vengono sommati insieme per produrre una soluzione a un problema complesso.

6) Si descrivano nel modo più accurato possibile i concetti di bias e variance, il loro rapporto e come nella pratica possano essere affrontati e ridotti. A tal fine si riportino anche esempi concreti che aiutino a chiarire i diversi aspetti coinvolti.

Il **Bias** è l'errore prodotto da assunzioni sbagliate presenti nell'algoritmo di apprendimento. È un fenomeno che distorce il risultato di un algoritmo a favore o contro un'idea. La distorsione è considerata un errore sistematico che si verifica nel modello di apprendimento automatico stesso a causa di ipotesi errate nel processo di machine learning.

Un bias elevato può fare in modo che l'algoritmo non individui correttamente le relazioni tra le feature e le classi target, questo fenomeno è detto **underfitting.**

Ci sono vari tipi di bias:

Bias **INDUTTIVO** ovvero, la selezione di un algoritmo di apprendimento implica delle **assunzioni** rispetto allo spazio delle ipotesi H, ovvero sono tutte le assunzioni sulla natura del target della funzione e della sua selezione. Questo tipo di assunzioni vengono chiamate bias induttivo e possono essere di due tipi, **Restrittive** ovvero che limitano lo spazio delle ipotesi e **Di preferenza** ovvero che impongono un ordine sullo spazio delle ipotesi.

Per esempio, un algoritmo di regressione lineare assume una relazione lineare tra le features degli esempi. Un algoritmo nearest neighbor assume che gli esempi vicini tra loro condividono la stessa classe.

Bias **ALGORITMICO** sono errori sistematici e ripetuti in un sistema che generano risultati scorretti, per esempio privilegiando un gruppo arbitrario di utenti invece di un altro. Molte delle volte non è dovuta dall'algoritmo, ma dai dataset forniti con dati incompleti o focalizzati su un certo tipo rispetto che ad un altro.

Per esempio, un algoritmo di riconoscimento facciale, per le persone occidentali abbiamo una accuratezza del 99% rispetto ai caucasici che è del 70%, per le persone di colore del 60% arrivando in certi casi critici per le donne di colore.

La **Varianza** è un errore prodotto da una sensibilità eccessiva alle piccole fluttuazioni nei dati di training.

Una varianza alta può far si che un algoritmo modelli rumore causale nei dati di training invece che gli output desiderati, questo fenomeno è detto **overfitting.**

Per cui in casi di **UNDERFITTING** ci sono pochi parametri nel modello e un'elevata discrepanza nella classificazione (**high bias**). Il processo di apprendimento è troppo semplice.

In casi di **OVERFITTING** ci sono troppi parametri nel modello e un'elevata variabilità della classificazione. Il modello è troppo complesso e sensibile ai dati di training (**high variance**).

Nel modello ottimale cerchiamo di ottenere un **low bias** e una **low variance.**

Quello che possiamo fare in base al caso in cui il nostro modello si trova è:

In caso di **high variance** possiamo risolverla con:

- Aggiungere dati di training;
- Provare con un set minore di features;
- Incrementare il valore di lambda, il quale è una variabile usata per diminuire/o aumentare in base ai casi il peso delle features;

In caso di **high bias** possiamo:

- Usare un set maggiore di features;
- Aggiungere complessità al modello;
- Decrementare il valore di lambda;

7) Si descriva in modo accurato il modello di logistic regression, le sue principali caratteristiche ed il contributo dei diversi elementi presenti nella funzione di costo.
Si riporti inoltre una comparazione con il modello di classificazione lineare, evidenziando elementi in comune e differenze principali.
Infine, si descriva chiaramente la procedura di addestramento mediante l'applicazione di gradient descent.

La regressione logistica è un algoritmo di apprendimento supervisionato che costruisce un modello probabilistico lineare di classificazione dei dati. È usata nel machine learning per l'addestramento di un algoritmo nella classificazione supervisionata dei dati usato per stimare due risultati e solo due.

L'algoritmo stima la probabilità di occorrenza di un evento adattando i dati a una funzione logistica.

Essa è ottenuta dall'espressione:

z= theta trasposto * x

1 / (1+e^(-z))

La funzione **di costo** per la logistic regression è la seguente:

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \cdot \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \cdot \log(1 - h_\theta(x^{(i)}))\right]$$

Con regolarizzazione diventa:

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \cdot \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \cdot \log(1 - h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}$$

Nella fase di addestramento l'algoritmo di regressione logistica prende in input n esempi da un insieme di training e ogni singolo esempio è composto da m attributi e dalla classe corretta di appartenenza.

Durante l'addestramento l'algoritmo elabora una distribuzione di pesi (W) che permetta di classificare correttamente gli esempi con le classi corrette.

Poi viene calcolata la combinazione lineare z del vettore dei pesi W e degli attributi e questa combinazione lineare viene passata alla **sigmoide** che calcola la probabilità di appartenenza del campione alle classi del modello.

La **REGOLARIZZAZIONE** è la metodologia per mantenere un modello con un numero immenso di variabili e quindi un modello dettagliato, ma pesando diversamente le variabili durante la fase di dataset per calibrare i pesi e non cadere in casi di overfitting.

La regolarizzazione funziona aggiungendo la penalità associativa ai valori di coefficiente all'errore di ipotesi.

Il valore penalizzatore **lambda** viene deciso/calcolato tramite metodo di **gradient descent** il quale si aggiorna con una velocità decisa, fino a trovare il minimo locale in un determinato numero di iterazioni.

Il **Gradient Descent** è una tecnica che consente di ottenere i punti di minimo di una funzione.

La formula è:

$$\theta_{k+1} = \theta_k - \eta \nabla J(\theta_k)$$

Dove n è detto learning rate e rappresenta la lunghezza del "passo" che l'algoritmo compie nel verso della discesa più rapida. Un learning rate troppo elevato rischia di non far convergere l'algoritmo mentre uno troppo basso aumenta il tempo di convergenza.

Per calcolare il gradiente il costo viene calcolato su tutto il training set e dopo ogni aggiornamento il gradiente viene ricalcolato per il nuovo vettore theta(k+1). Questo metodo è chiamato **batch gradient descent.** Il risultato può variare in base alle condizioni iniziali dei parametri della funzione di costo in quanto c'è il rischio che la discesa di fermi in una zona di **minimo locale** non trovando i valori ottimali.

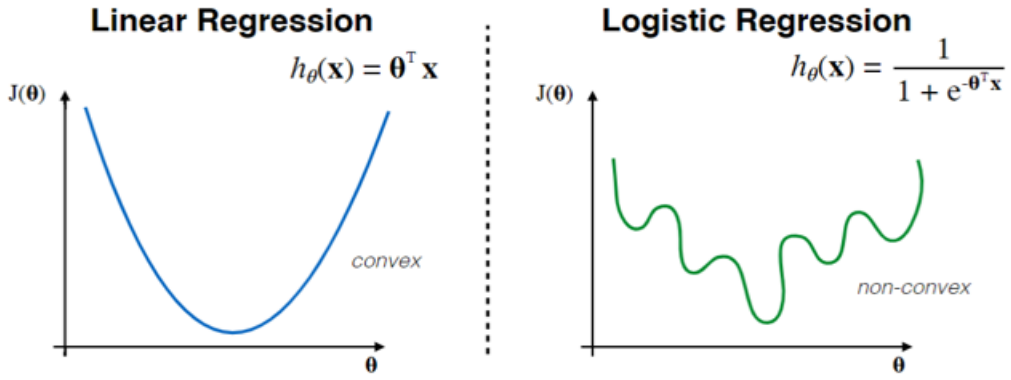
Per una corretta implementazione è necessario aggiornare tutti i parametri simultaneamente:

$\texttt{temp0} := \theta - \eta \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1);$
$\texttt{temp1} := \theta - \eta \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1);$
$\theta_0 := \texttt{temp0};$
$\theta_1 := \texttt{temp1};$

Dove h(x) è la **funzione di predizione** e J(theta) la **funzione di costo**.

La differenza tra regressione logistica e lineare è ampia.

Come gia detto la regressione logistica è un modello di classificazione, mentre la regressione **lineare** è un algoritmo regressore che restituisce come risposta un altro valore tramite l'interpolazione dei dati.



L'update del gradiente è esattamente la stessa sia per la regressione lineare che per quella logistica.

Riguardo alla Linear Regression dato in esempio un prodotto cartesiano con dei punti disposti nel piano, la nostra linear regression produce un modello basato sul costo totale che è una linea, corretta ma inefficiente.

La logistic regression o **sigmoide** è un approccio decisamente migliore rispetto alla linear regression, nell'implementazione di classificatori in quanto essa è dotata di una peculiare forma (quella di una S) e la proprietà di avere il hteta(x) contenuto tra 0 e 1.

Nella quale theta(0) sarà il punto di flesso mentre theta(1) sarà la velocità (inclinazione) con la quale la funzione passerà da 0 (sinistra) ad 1 (destra).

Se i dati sono uniformemente distribuiti, e si sta utilizzando la tecnica dei **Least-squares** è possibile notare un comportamento molto simile alla funzione lineare, invece se si va a creare uno sbilanciamento nei dati, notiamo che la lineare(volendo essere troppo giusta)

crea un errore, mentre la sigmoide non è affetta da questo problema grazie alle proprietà sopra descritte.

Il **difetto** della logistic regression è che la funzione non è convessa come quella lineare ma tende ad essere più ondulata.

8) Si descriva dettagliatamente la procedura di cross-validation, motivandone scopo ed utilità, e fornendo una chiara descrizione della (corretta) procedura di addestramento di un qualunque sistema di machine learning.

Rispetto al metodo dell'hold out, il metodo di cross validation è una tecnica dinamica che permette di usare in modo alternato i dati sia per il train che per il test.

Il metodo più comune è detto **k-fold cross validation** in cui il set di dati viene suddiviso in k partizioni uguali, l'algoritmo usa k- di esse per il training e la partizione rimanente per la validazione. Questo procedimento viene eseguito k volte, lasciando ogni volta fuori dal set di training una partizione diversa.

Infine viene fatta una **media dei vari risultati** ottenuti per calcolare la prestazione generale del modello. A questo punto si può fare una **valutazione finale** del modello risultante utilizzando il test set, che non è stato utilizzato all'interno della cross validation, per evitare di sovrastimare le prestazioni.

Un caso speciae del k-fold cross validation è detto **Leave-one-out** ed è quando il numero di partizioni scelte corrisponde al numero dei dati all'interno del set di training corrisponde al numero dei dati all'interno del set di training e quindi ad ogni iterazione il training set conterrà tutti i dati disponibili tranne uno.

Lo **scopo** della cross validation è quindi quello di cercare di far convergere, al crescere di k, l'errore medio che calcolano all'errore reale.

Ma al crescere di k, aumenta anche il costo computazionale poiché bisogna rifare k volte il training quindi solitamente è utilizzato un k relativamente piccolo, solitamente 10.

Rispetto al metodo **hold out** questo metodo non lavora in modo statico sui dati suddividendoli appunto staticamente, ma riutilizza i dati usati per il train anche per il validation, questo comporta a una migliore analisi del modello in cambio di una alta computazione rispetto al metodo di hold out.

9) Spiegare in dettaglio gli elementi fondamentali di SVM, in particolare
i) la sua interpretazione geometrica,
ii) la funzione di costo,
iii) le differenze/similitudini con altri modelli di ML.
Infine, si introduca brevemente l'estensione di SVM basata sul kernel trick.

Le macchine a vettori di supporto (**SVM** support vector machine) sono dei **modelli di apprendimento** associati ad algoritmi per la **regressione e classificazione** solitamente più per la classificazione. Possono risolvere problemi **lineari e non** e l'idea è quella di creare una linea o un iperpiano che separa i dati in classi.

Dato un insieme di esempi per l'addestramento, ognuno dei quali etichettato con la classe di appartenenza fra le due possibili classi, un algoritmo di addestramento per la SVM costruisce un modello che assegna i nuovi esempi ad una delle due classi, ottenendo quindi un **classificatore lineare binario non probabilistico**.

Un modello SVM è una rappresentazione degli esempi come **punti nello spazio** mappati in modo tale che gli esempi appartenenti alle due diverse categorie sono chiaramente **separati da uno spazio** più possibilmente ampio. I nuovi esempi sono quindi mappati nello stesso spazio e la predizione della categoria alla quale appartengono viene fatta sulla base del lato nel quale ricade.

Esiste una retta fittizia che divide le due classi ed essa è quella ottimale quando è disposta alla massima distanza dai punti di entrambe le classi. La distanza tra gli elementi e la retta viene chiamata **margine** per cui l'idea per trovare la retta separatrice ottimale, è quella di trovare la retta, la quale massimizza la distanza tra i punti di ambo le due classi, e per scegliere i punti scegliamo i punti più vicini all'altro gruppo dell'altra classe.

Per cui a differenza della regressione lineare in cui si cerca di trovare una linea che riduca al minimo la distanza dai punti dati, una SVM cerca di **massimizzare** la distanza.

Ci sono dati considerati come "anomalie" che attraversano idealmente il confine definito dal separatore, e se cercassimo di ottenere un separatore perfetto che comprenda anche queste anomalie, produrremmo un separatore super complesso il quale poi non si adatta al resto dei dati, per cui si è deciso di permettere alle anomalie di attraversare il separatore senza che influiscano sulla determinazione del separatore chiamando questi esempi come variabili **slack,** utilizzate in modo da ignorare il rumore e questo metodo si chiama **soft margin.**

La SVM utilizza la **hinge loss** che pone attenzione sui punti nel confine poiché:

Dato un punto sul piano

- se esso ha distanza di 1 o maggiore dalla retta la nostra perdita è 0 poiché il nostro punto si trova nella parte corretta e si trova lontano dal confine;
- se ha distanza compresa tra 0 e 1 allora il nostro punto è collocato nella parte corretta ma si trova vicino al confine generando quindi una **hinge loss**.
- se la distanza dal confine è 0 allora avremo una perdita di 1 poiché il nostro punto si trova sul confine ed è perfettamente sulla retta confinatrice.
- Se la distanza è negativa significa che il punto è collocato nella parte sbagliata e comporta a una perdita maggiore di 1

I punti classificati correttamente avranno una dimensione di perdita piccola o nessuna, mentre le istanze classificate in modo errato avranno una dimensione di perdita elevata, una distanza negativa dal confine comporta un'elevata perdita di cerniera (**hinge loss**) poiché si trova nella parte sbagliata della retta o iperpiano.

Hinge Loss
La funzione di loss delle SVM è detta Hinge Loss ed è definita nel modo seguente:

$$cost(h_\theta(x^{(i)}), y^{(i)}) = \begin{matrix} max(0, 1 - \theta^T x(i)) \text{ se } y^{(i)} = 1 \\ max(0, 1 + \theta^T x(i)) \text{ se } y^{(i)} = 0 \end{matrix}$$

Con questa funzione di loss verrà selezionato un margine equidistante dai vari esempi.
Possiamo riscrivere l'obiettivo di ottimizzazione nel seguente modo:

$$\begin{matrix} \min \frac{1}{2} \sum_{j=1}^{n} \theta_j^2 = \frac{1}{2} ||\theta||^2 \\ \text{s.t. } \theta^T x^{(i)} \geq 1 \text{ se } y^{(i)} = 1 \\ \theta^T x^{(i)} \leq -1 \text{ se } y^{(i)} = 0 \end{matrix}$$

La hinge loss è una funzione **convessa** e può essere ottimizzata tramite **gradient descent.**

La **funzione di costo** è:

$$\min_{\theta} \frac{1}{m} \left[\sum_{i=1}^{m} y^{(i)} \cdot cost_1(\theta^T x^{(i)}) + (1-y^{(i)}) \cdot cost_0(\theta^T x^{(i)}) \right] + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2$$

$$C\,A + B \quad , \quad C = \frac{1}{\lambda}$$

$$\Rightarrow \min_{\theta} C \sum_{i=1}^{m} \left[y^{(i)} \cdot (cost_1(\theta^T x^{(i)})) + (1-y^{(i)}) \cdot (cost_0(\theta^T x^{(i)})) \right] + \frac{1}{2} \sum_{j=1}^{n} \theta_j^2$$

Nel quale questa funzione calcola il costo in modo diverso a seconda della classe d'esempio i, se yi è pari a 1 o 0 cosi:

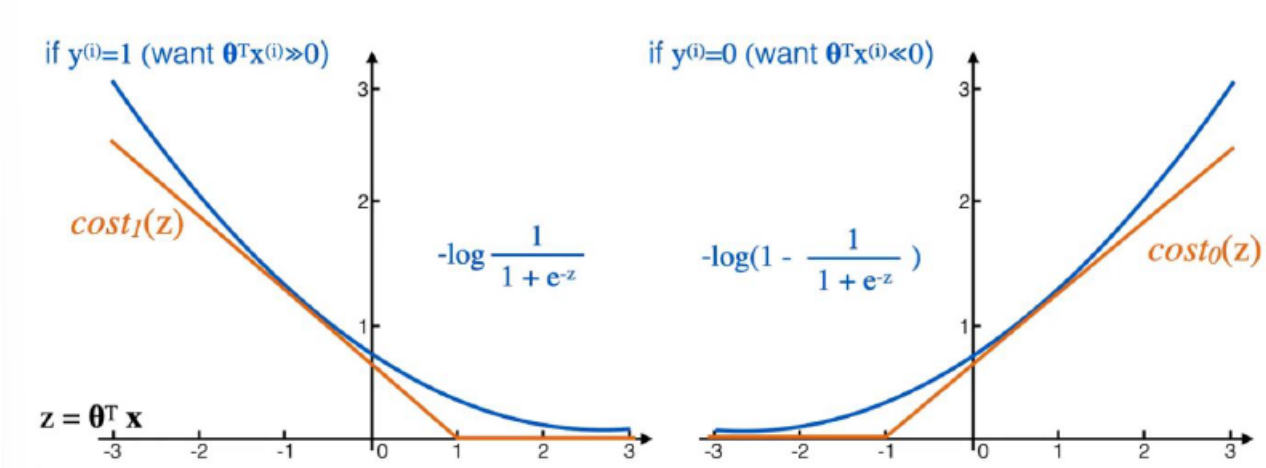
Questa funzione calcola il costo in modo diverso a seconda della classe dell'esempio i, se $y^{(i)}$ è pari a 1:

$$cost = -y^{(i)} \cdot \log(\frac{1}{1+e^{\theta^T x^{(i)}}})$$

invece, se $y^{(i)}$ è pari a 0:

$$cost = -(1 - y^{(i)}) \cdot \log(1 - \frac{1}{1+e^{\theta^T x^{(i)}}})$$

Nel caso in cui yi sia pari a 1 più il risultato di theta trasposto * xi sarà grande più il costo tenderà a 0 e più il risultato di theta trasposto*xi sarà piccolo più il costo tenderà ad infinito, viceversa per il caso in cui yi è pari a 0.



La funzione obiettivo di una SVM si può quindi scrivere nel seguente modo:

$$\min \frac{1}{\lambda} \sum_{i=1}^{m} [y^{(i)} \cdot cost_1(\theta^T x^{(i)}) + (1 - y^{(i)}) \cdot cost_0(\theta^T x^{(i)})] + \frac{1}{2} \sum_{j=i}^{n} \theta_j^2$$

La funzione **di costo della SVM** altro non è che una **approssimazione di quella logistica**. Al contrario della classificazione lineare la **mal distribuzione** delle classi negli esempi per ciascuna **non influisce** sulla bontà della divisione del piano, questo proprio perché non ci si concentra su tutti i dati ma solo sul vettore dei dati presenti sul margine.

Per la classificazione di dati non lineari con SVM si utilizza una pratica detta **kernel trick.**

Esso proietta i dati in un nuovo spazio tramite una funzione detta **kernel.** Partendo da un particolare spazio impongo gli stessi vincoli in uno spazio di dimensioni maggiori. La funzione kernel è una funzione che mappa i vettori nello spazio di partenza nello spazio con dimensioni maggiori.

I kernel sono anche detti prodotto vettoriale generalizzato in quanto:

Questo in sostanza viene effettuato in casi in cui dato un piano a n dimensioni con i punti assegnati nel piano, essi se non sono facilmente separabili, il **kernel trick** ci permette di

generare un piano con m dimensioni in cui m > n in cui i punti sono ricollocati nel modo corretto in cui generalmente è più facile una suddivisione corretta per la SVM.

10) Si descrivano nel modo più accurato possibile gli alberi di decisione, i loro vantaggi e svantaggi rispetto ad altri modelli (ad es. reti neurali) e si evidenzi il principale inductive bias di tale algoritmo. Si fornisca inoltre un semplice esempio di albero di decisione.

Gli alberi di decisione o anche detti **decision tree** è un sistema con n variabili in input e m variabili in output.

Le variabili in input (**features)** sono derivate dall'osservazione dell'ambiente. Le ultime variabili in output identificano la **decisione / azione / classi target** da interpretare.

Si nota che in alberi **molto profondi** le variabili in output intermedie, in uscita dai nodi genitori, coincidono con le variabili in input dei nodi figli. Le variabili in output intermedie condizionano il percorso verso la decisione finale.

Ogni nodo verifica una condizione (**test**) su una particolare proprietà dell'ambiente (**variabile**) e ha due o più diramazioni verso il basso in funzione.

Il processo consiste in una sequenza di test. **Comincia sempre dal nodo radice, e procede verso il basso.**

Man mano che il processo di selezione prosegue verso il basso, lo spazio delle ipotesi si riduce perché gran parte dei rami decisionali dell'albero sono eliminati.

La **decisione finale** si trova nei nodi **foglia terminali**, quelli più in basso, in questo modo dopo aver analizzato le varie condizioni, l'agente giunge alla decisione finale.

In un albero decisionale le **variabili** possono essere:

Variabili **Discrete**, le quali hanno valori numerici interi e in questo caso si parla di **classificazione**. La rappresentazione più semplice è quella booleana in cui le variabili hanno soltanto due valori, 0 e 1.

Variabili **Continue**, le quali hanno valori numerici reali. Dati due numeri reali qualsiasi, c'è sempre un altro numero intermedio tra questi, in questo caso si parla di **regressione.** La rappresentazione delle variabili continue è molto più complessa ma si adatta meglio a condizioni di incertezza, quando non esiste una distinzione netta tra il vero e il falso, tra un affermazione vera e falsa, ecc…

Il miglior modo per risolvere l'**inductive bias** di questo algoritmo e ottenere un albero efficiente è la preferenza di alberi corti rispetto ad alberi lunghi. Gli alberi con un elevato **information gain vicino o sulla radice** sono preferiti rispetto a quelli che non lo hanno.

I **VANTAGGI**

Gli alberi logici hanno l'indiscusso vantaggio della semplicità e rispetto alle reti neurali sono più facilmente comprensibili dagli esseri umani. Pertanto, l'uomo può **verificare come la macchina giunge alla decisione.**

Può lavorare su dati numerici e categorici contemporaneamente.

Non richiede moltissimi dati e performa bene sia su pochi dati e sia su grandi dataset.

Gli **SVANTAGGI**

La rappresentazione ad albero decisionale è poco adatta per i problemi complessi, perché lo spazio delle ipotesi diventa troppo grande.

La complessità spaziale dell'algoritmo potrebbe diventare proibitiva.

Non è un modello robusto poiché un piccolo cambiamento nel training può comportare un grandissimo cambiamento nella prediction.

Altissimo rischio di **overfitting.**

Esistono diversi algoritmi per costruire un albero. Uno degli algoritmi storici è l'**ID3 , Iterative Dichotomiser 3.** L'albero è costruito in maniera top-down usando la politica nota come **divide et impera** ed utilizza il concetto di **Entropia.**

Per ogni ricorsione l'algoritmo deve scegliere l'attributo che meglio classifica gli esempi. Per la scelta del miglior attributo e per la costruzione dell'albero decisionale più corretto e preciso, uno dei metodi più comunemente usati è l'**information gain,** che serve appunto a capire qual è l'attributo migliore**.** Viene calcolato utilizzando un valore chiamato **entropia.**

L'**entropia,** concetto usato in fisica e matematica, fu esteso anche alla statistica con la formula:

$$H(S) = -\sum_{c \in C} p(c) \log_2 p(c)$$

La formula è una sommatoria dove l'esponente **c** è il numero di classi o attributi e l'elemento **p(c)** è il rapporto tra il valore della classe i esima e la somma totale del valore delle classi.

In caso di una equa distribuzione delle classi e dei loro valori, otterremo una entropia tendente all' **1**, mentre se non vi è casualità poiché una classe detiene il totale del valore totale, l'entropia sarà tendente a **0.**

L'**Entropia** è la misura dell'incertezza di una variabile casuale, caratterizza l'impurità di una raccolta arbitraria di esempi. Maggiore è l'entropia maggiore è il contenuto informativo.

L'**Information Gain** è il grado di informazione associato ad ogni attributo e viene calcolato per ogni nodo poiché l'entropia degli attributi cambia in base al livello in cui ci troviamo.

S si riferisce all'intero insieme di esempi che abbiamo, **a** è l'attributo che vogliamo dividere, **modulo di S** è il numero di esempi.

Per cui è l'entropia del dataset – l'entropia del subset generato dall'attributo a.

Viene comunemente utilizzato nella costruzione di alberi decisionali da un set di dati di addestramento, valutando l'**information gain** per ciascuna variabile e selezionando la variabile che **massimizza** l'information gain, che a sua volta riduce al minimo l'entropia e **divide al meglio il set di dati** in gruppi per una classificazione efficace.

Un esempio di albero di decisione semplice è un albero binario che distingue tra limoni e arance in base alla lunghezza e larghezza.

Tramite **information gain** che usa l'entropia scelto il parametro che crea la divisione migliore, cioè la divisione maggiore del dataset, procedo in maniera ricorsiva nei figli fino a quando non ottengo una suddivisione ottimale.

11) Si descriva in modo accurato l'algoritmo k-NN, illustrando il ruolo dei principali iperparametri, i vantaggi e le debolezze del modello nei confronti di altri algoritmi affrontati nel corso, e si evidenzi il principale inductive bias di tale algoritmo.

L'algoritmo **k-nearest neighbors (KNN)** è un algoritmo di apprendimento automatico supervisionato usato per risolvere problemi di classificazione e regressione.

Questo algoritmo non include alcuna fase di apprendimento, esso non calcola il metodo predittivo come la regressione lineare o logistica, ma **trova le similitudini tra le features.**

Per poter spiegare il K-NN vi è la necessità di spiegare prima l'algoritmo **NN o nearest neighbors** il quale assume che dato un esempio da predire la classe di appartenenza e la sua rappresentazione nello spazio, nella maggior parte dei casi vi l'idea che appartenga alla stessa classe degli esempi vicini ad esso.

Il **problema di NN** è che non riesce a gestire in modo efficiente i dati **anomali** in quanto se per esempio dato un classificatore binario NN, e uno spazio nel piano in cui le due classi hanno grande distinzione in esso, un dato anomalo è un esempio di una classe, la quale rappresentazione nello spazio avviene all'interno o molto vicino alla zona di maggior concentrazione degli esempi dell'altra classe. Questo produrrà che dato un ipotetico nuovo esempio e la sua rappresentazione nello spazio, se esso avrà rappresentazione

molto vicina al dato anomalo e con classe da predire diversa da esso, il nostro modello predirà la classe del dato anomalo erroneamente.

Per cui per far fronte a questo problema, nasce il **K-NN** che è meno sensibile ai dati anomali in quanto dato un esempio da predire, decide la sua classe di appartenenza in base ai K elementi più vicini nello spazio ad esso.

Più piccolo è il **k** più il modello sarà sensibile ai dati anomali, ma se deciso un **k** troppo grande vi è il rischio che data la richiesta di predizione di un esempio, si andrà a verificare esempi troppo lontani da quello da esaminare e dato un dataset sbilanciato o comunque con aree non omogenee, si predirà in modo spesso non corretto.

Per esempio, se dato un dataset di 10 elementi e un modello con solo due possibili classi, blu e rosso. Se avremo un dataset sbilanciato come 8 elementi blu e 2 rossi, e un k = 5, ad ogni immissione di un dato da predire, il nostro modello predirà sempre che apparterrà alla classe blu, anche se il dato da predire avrà il suo spazio di rappresentazione molto vicino agli altri due esempi del rosso, poiché il nostro modello andando a cercare i 5 esempi più vicini a quello da predire, siccome di rossi ce ne sono solo 2, andrà sempre a prendere 3 elementi blu anche se molto distanti.

L'**inductive bias** di KNN è che si assume che nella maggior parte dei casi in piccolo quartiere nello spazio delle caratteristiche gli esempi appartengono alla stessa classe. Cioè il presupposto è che i casi vicini tra loro tendano alla stessa classe.

Esiste un **valore ottimale di k** che è uguale alla **radice del numero di esempi**.

I **VANTAGGI** sono:

è **facile da implementare** e allo stesso tempo semplice e accurato;

Si **adatta facilmente** poiché quando vengono aggiunti nuovi campioni di addestramento, l'algoritmo si adatta per tenere conto di eventuali nuovi dati poiché tutti i dati di addestramento vengono archiviati in memoria;

Pochi iperparametri poiché abbiamo un solo valore k e una metrica di distanza;

Funziona bene anche con dataset molto grandi;

GLI **SVANTAGGI** sono:

è **sensibile** ai disturbi delle classi, meno della NN ma comunque sensibile;

Non ha una buona scalabilità poiché è un algoritmo pigro, occupa molta più memoria e spazio di storage di dati rispetto ad altri classificatori rendendolo costoso in termini di tempo e denaro.

Le **distanze sono meno significative** quando abbiamo un valore di k alto.

12) Si descrivano nel modo più accurato possibile gli alberi di decisione, i loro vantaggi e svantaggi rispetto ad altri modelli (in particolare rispetto ad algoritmi/modelli parametrici),e si evidenzi il principale inductive bias di tale algoritmo. Infine, si illustri brevemente l'estensione di tale modello attraverso random forest.

Gli alberi di decisione o anche detti **decision tree** è un sistema con n variabili in input e m variabili in output.

Le variabili in input (**features)** sono derivate dall'osservazione dell'ambiente. Le ultime variabili in output identificano la **decisione / azione / classi target** da interpretare.

Si nota che in alberi **molto profondi** le variabili in output intermedie, in uscita dai nodi genitori, coincidono con le variabili in input dei nodi figli. Le variabili in output intermedie condizionano il percorso verso la decisione finale.

Ogni nodo verifica una condizione (**test**) su una particolare proprietà dell'ambiente (**variabile**) e ha due o più diramazioni verso il basso in funzione.

Il processo consiste in una sequenza di test. **Comincia sempre dal nodo radice, e procede verso il basso.**

Man mano che il processo di selezione prosegue verso il basso, lo spazio delle ipotesi si riduce perché gran parte dei rami decisionali dell'albero sono eliminati.

La **decisione finale** si trova nei nodi **foglia terminali**, quelli più in basso, in questo modo dopo aver analizzato le varie condizioni, l'agente giunge alla decisione finale.

In un albero decisionale le **variabili** possono essere:

Variabili **Discrete**, le quali hanno valori numerici interi e in questo caso si parla di **classificazione**. La rappresentazione più semplice è quella booleana in cui le variabili hanno soltanto due valori, 0 e 1.

Variabili **Continue**, le quali hanno valori numerici reali. Dati due numeri reali qualsiasi, c'è sempre un altro numero intermedio tra questi, in questo caso si parla di **regressione.** La rappresentazione delle variabili continue è molto più complessa ma si adatta meglio a condizioni di incertezza, quando non esiste una distinzione netta tra il vero e il falso, tra un affermazione vera e falsa, ecc…

Il miglior modo per risolvere l'**inductive bias** di questo algoritmo e ottenere un albero efficiente è la preferenza di alberi corti rispetto ad alberi lunghi. Gli alberi con un elevato **information gain vicino o sulla radice** sono preferiti rispetto a quelli che non lo hanno.

I **VANTAGGI**

Gli alberi logici hanno l'indiscusso vantaggio della semplicità e rispetto alle reti neurali sono più facilmente comprensibili dagli esseri umani. Pertanto, l'uomo può **verificare come la macchina giunge alla decisione.**

Può lavorare su dati numerici e categorici contemporaneamente.

Non richiede moltissimi dati e performa bene sia su pochi dati e sia su grandi dataset.

Gli **SVANTAGGI**

La rappresentazione ad albero decisionale è poco adatta per i problemi complessi, perché lo spazio delle ipotesi diventa troppo grande.

La complessità spaziale dell'algoritmo potrebbe diventare proibitiva.

Non è un modello robusto poiché un piccolo cambiamento nel training può comportare un grandissimo cambiamento nella prediction.

Altissimo rischio di **overfitting.**

Esistono diversi algoritmi per costruire un albero. Uno degli algoritmi storici è l'**ID3 , Iterative Dichotomiser 3.** L'albero è costruito in maniera top-down usando la politica nota come **divide et impera** ed utilizza il concetto di **Entropia.**

Per ogni ricorsione l'algoritmo deve scegliere l'attributo che meglio classifica gli esempi. Per la scelta del miglior attributo e per la costruzione dell'albero decisionale più corretto e preciso, uno dei metodi più comunemente usati è l'**information gain,** che serve appunto a capire qual è l'attributo migliore**.** Viene calcolato utilizzando un valore chiamato **entropia.**

L'**entropia,** concetto usato in fisica e matematica, fu esteso anche alla statistica con la formula:

$$H(S) = -\sum_{c \in C} p(c) \log_2 p(c)$$

La formula è una sommatoria dove l'esponente **c** è il numero di classi o attributi e l'elemento **p(c)** è il rapporto tra il valore della classe i esima e la somma totale del valore delle classi.

In caso di una equa distribuzione delle classi e dei loro valori, otterremo una entropia tendente all' **1**, mentre se non vi è casualità poiché una classe detiene il totale del valore totale, l'entropia sarà tendente a **0.**

L'**Entropia** è la misura dell'incertezza di una variabile casuale, caratterizza l'impurità di una raccolta arbitraria di esempi. Maggiore è l'entropia maggiore è il contenuto informativo.

L'**Information Gain** è il grado di informazione associato ad ogni attributo e viene calcolato per ogni nodo poiché l'entropia degli attributi cambia in base al livello in cui ci troviamo.

S si riferisce all'intero insieme di esempi che abbiamo, **a** è l'attributo che vogliamo dividere, **modulo di S** è il numero di esempi.

Per cui è l'entropia del dataset – l'entropia del subset generato dall'attributo a.

Viene comunemente utilizzato nella costruzione di alberi decisionali da un set di dati di addestramento, valutando l'**information gain** per ciascuna variabile e selezionando la variabile che **massimizza** l'information gain, che a sua volta riduce al minimo l'entropia e **divide al meglio il set di dati** in gruppi per una classificazione efficace.

Il concetto di **random forest** nasce per ridurre l'errore dovuto alla distorsione ovvero la **varianza** dei dati. Queste **random forest** mitigano bene questo problema poiché sono una **raccolta di alberi decisionali** i cui risultati sono aggregati in unico risultato finale. La loro capacità di **limitare l'overfitting** senza sostanzialmente avere l'errore dovuto dal bias.

Un modo in cui queste riducono la varianza è dovuto all'addestramento su diversi campioni di dati, un altro modo consiste nell'utilizzare un sottoinsieme casuale di funzionalità, per esempio se abbiamo 20 features, ogni foresta ne utilizzerà solo alcune ma quelle omesse potrebbero essere utili, quindi non è sempre conveniente ma se queste caratteristiche non fossero scelte casualmente, gli alberi diventerebbero particolarmente correlati, poiché alcune funzionalità potrebbero essere molto predittive e quindi le stesse caratteristiche verrebbero scelte in molti degli alberi di base.

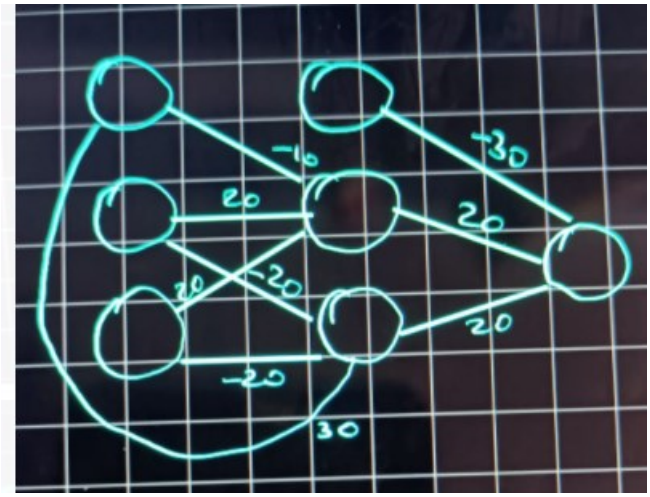
13) Si descriva accuratamente un esempio di rete neurale per la realizzazione di un operatore logico XOR (indicando valori dei parametri e funzione di attivazione prescelta) ; si fornisca inoltre l'analoga soluzione utilizzando un albero di decisione, e si discutano pro e contro delle due soluzioni.

Per poter realizzare l'operatore logico XOR tramite rete neurale, non vi è la possibilità di realizzazione tramite un singolo neurone/percettrone poiché è un operazione logica non lineare che suddivide il piano in 4 parti, e che quindi non è realizzabile tramite un singolo percettrone a combinazione lineare.

L'idea per realizzarlo è quella di costruire una rete neurale con 1 strato per l'hidden layer sui i quali verranno combinate insieme l'operazione AND e OR e NOT o come in questo caso una NAND , OR e una AND, dato che la XOR è la combinazione di esse.

Se usassimo l'output di un singolo percettrone, questo sarebbe completamente sbagliato, ma l'unione dei singoli output "stupidi" genera una suddivisione ottimale per il compito richiesto.

Una possibile implementazione di rete neurale per la XOR è la seguente:



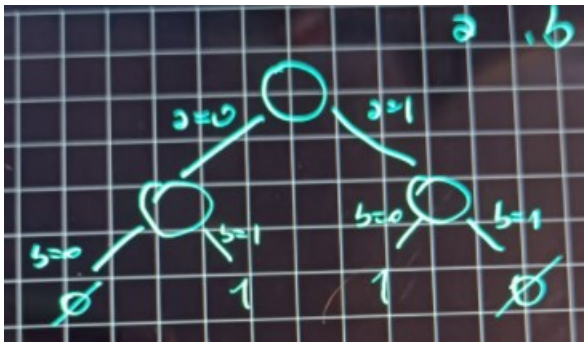
Innanzitutto, notiamo che i due neuroni sopra, nei primi due strati corrisponderanno al **bias unit** ovvero dei valori utilizzati per regolare l'output dei neuroni degli strati successivi, questi fondamentalmente vengono utilizzati per effettuare correttamente le operazioni di sommatoria dei valori.

Il secondo e terzo nodo della prima colonna sono gli input, per i quali possono valere 0 o 1 e questi valori verranno utilizzati e moltiplicati con due diverse operazioni e moltiplicazioni per lo strato successivo. Il secondo nodo è una OR logica il quale riceve un **bias unit** di -10 e i valori dei due input moltiplicati per 20 che possono rappresentare 0 o 20. Questo bias unit viene usato poiché nel caso in cui il risultato della sommatoria sia < 0 implicherà che l'input ricevuto siano due 0, mentre se > 0 allora almeno un input sia 1, poiché 20 è maggiore di 10, inoltre si nota che si usa -10 e si moltiplica per 20, poiché non si vuole mai ottenere come risultato della sommatoria 0, poiché è il punto di transizione dei due possibili risultati.

Invece il terzo nodo della seconda colonna rappresenta una NAND logica, la quale restituisce falso solo quando entrambi gli input valgono 1, poiché il **bias unit** è 30 e per rendere falso, ovvero il risultato della sommatoria negativo, entrambi gli input devono valere 1 che poi saranno moltiplicati per -20.

Il nodo di output è una AND, la quale ha un **bias unit** di -30 e per rendere la sommatoria positiva necessita che entrambi i valori dei due input intermedi sia 1 poiché sono moltiplicati per 20.

Una possibile implementazione tramite albero decisionale è realizzabile molto semplicemente cosi:



Dove a e b corrisponderanno ai due input, come possiamo vedere è stato deciso che prima verrà valutato a per poi valutare b in base alla valutazione di a.

Si noti che l'entropia di a e di b è uguale, per cui iniziare con a oppure b non migliora l'albero.

Per l'operazione logica XOR l'implementazione tramite albero sembra più adatta ma di ugual efficacia rispetto alla rete neurale.

Una rete neurale è molto utile per risolvere problemi complessi e ambigui ma in questo caso non è utile utilizzarla visto la sua complessità di base, mentre un decision tree è adatto a problemi come la XOR, peccando per problemi complessi in caso di esempi ambigui oppure dovuti alla sua facile voluminosità che può generare.

14) Si descrivano nel modo più accurato possibile i modelli di linear classification e logistic regression; si evidenzino differenze, vantaggi e svantaggi dell'uno rispetto all'altro. Si descriva infine il processo di apprendimento tramite gradient descent e le rispettive funzioni di costo, motivando in modo adeguato la particolare forma utilizzata in entrambi i casi.

Logisti regression e differenze con linear classification gia scritte ampiamente sopra.

Il processo di apprendimento ovvero il gradient descent pure, si noti che il procedimento è uguale per entrambe.

La **funzione di costo della linear regression** è la seguente:

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

La funzione di costo in questo caso è determinata dalla distanza del punto vero da calcolare a quello approssimato dalla funzione h per ogni punto del dataset.

Si noti che l'elevazione al quadrato è dovuta dal fatto che dobbiamo calcolare la distanza tra i due punti.

Il risultato della sommatoria viene diviso per 1/2m dove m sono il numero degli esempi, e viene fatto per stabilizzare il risultato della formula approssimando il costo dal totale a una media per ogni singolo punto.

15) Spiegare in dettaglio gli elementi fondamentali del perceptrone, più in generale, delle reti neurali multistrato, illustrando chiaramente le due fasi di feedforward e backpropagation. Si riporti inoltre un esempio di rete neurale per la realizzazione di un semplice operatore logico AND, ed uno per la funzione XOR.

Per il perceptrone e reti neurali spiegato sopra e anche XOR.

Il valore della predizione di htheta(x) viene calcolato tramite **feed forward** , ovvero data una rete neurale con idealmente diversi strati per l'hidden layer ma anche non, i neuroni sono collegati a quelli dello strato successivo generando diverse combinazioni da specifici neuroni. Questi collegamenti generalmente prendono l'input del neurone e lo moltiplicano per un determinato valore, questo implica che generando diverse combinazioni con valori moltiplicativi diversi e diverse combinazioni di neuroni, il nostro input verrà processato ad ogni iterazioni in diversi modi con diversi parametri generando molteplici output i quali verranno combinati insieme per produrre un output ottimale.

La **backpropagation** è la fase in cui data una rete neurale, viene effettuata un **gradient descent** per diminuire l'errore dato in output. Per cui si parte calcolando l'errore dato in output dalla rete, e si riporta indietro calcolando la stima della percentuale di errore dovuta da ogni singolo neurone, modificandone i parametri/pesi nella direzione del gradiente, ottimizzando la discesa del gradiente.

16) Si descriva dettagliatamente la procedura di cross-validation, motivandone scopo ed utilità, e fornendo una chiara descrizione della (corretta) procedura di addestramento di un qualunque sistema di machine learning. Si descrivano inoltre i concetti di true error e d'empirical error e se ne evidenzino le relazioni con la procedura di cross-validation.

La visione artificiale è un campo dell'intelligenza artificiale che permette ai computer e ai sistemi di ricavare informazioni significative da immagini digitali, video e altri input visivi e intraprendere azioni o formulare delle segnalazioni sulla base di tali informazioni. Se l'IA permette ai computer di pensare, la computer vision permette loro di vedere, osservare e capire.

La computer vision funziona più o meno come la vista umana, eccetto che gli umani hanno il vantaggio di disporre di anni e anni di esperienza in cui si è allenata a distinguere gli oggetti, quanto sono lontani, se si stanno muovendo e se c'è qualcosa di sbagliato in un immagine.

La computer vision permette alle macchine di svolgere queste funzioni, ma deve farlo in molto tempo con telecamere, dati e algoritmi piuttosto che con retine, nervi ottici e una corteccia visiva.

Dato che un sistema che è stato sviluppato per ispezionare i prodotti o guardare un asset di produzione può analizzare migliaia di prodotti o processi al minuto, notando difetti o problemi impercettibili, ed esso può superare rapidamente le capacità umane.