**Machine Learning Concepts**

* Cos’è il Machine Learning?
* Supervised Learning
* Overfitting e Underfitting
* Evaluating Performance – Classification Error Metrics
* Evaluating Performance – Regression Error Metrics
* Unsupervised Learning

**Cos’è il Machine Learning?**

Il machine learning è un metodo di analisi dei dati che automatizza la costruzione di modelli analitici. La parola chiave in questo caso è **automatizza**.

Utilizza algoritmi che iterativamente imparano dai dati, il machine learning permette ai computer di cercare informazioni nascoste senza essere esplicitamente programmate nel dove cercare. Normalmente diresti al computer cosa e dove cercare i dati ma con il machine learning il computer segue un determinato set di regole e poi il programma troverà le informazioni nascoste nei dati.

Il machine learning viene usato in tantissimi ambiti: detenzione di frode, ricerche sul web, pubblicità in tempo reale, prevenzione di rottura di equipaggiamenti, nuovi modelli di prezzo, detenzioni intrusioni nei network, consiglio di motori, segmentazione dei clienti, analisi del sentiment, predizione fidelizzazione dei clienti, riconoscimento di sequenze e immagini, filtro di spam delle email eccetera.

Le reti neurali sono un modo di modellizare un sistema nervoso biologico matematicamente.

Queste reti posso essere usate per risolvere compiti che altri algoritmi non possono, per esempio la classificazione di immagini.

Il deep learning semplicemente riferisce a reti neurali con più di uno strato nascosto.

**Supervised Learning**

Gli algoritmi di apprendimento supervisionato (**supervised learning**), sono allenati usando degli esempi etichettati, come un input il quale desiderato è già conosciuto. La parola chiave è **etichettato**.

Per esempio, un segmento di testo potrebbe avere una categoria etichettata come: spam vs email leggittima oppure una recensione positiva vs una negativa.

Il network riceve un set di input insieme ai corrispondenti output corretti e l’algoritmo impara comparando il suo output insieme a quello corretto per trovare gli errori. Successivamente modifica il suo modello di conseguenza.

Il supervised learning è usato comunemente in applicazioni dove i dati storici prevedono la possibilità di eventi futuri.

Il **processo di machine learning** è suddiviso in diversi passaggi.

Si passa prima dall’acquisizione dei dati, per esempio dai clienti, sensori o altri metodi, poi si esegue la pulizia e ordine di essi (usando per lo più pandas) in modo che la rete neurale possa processarli, successivamente si posso testare subito i dati su un modello esistente oppure usarli per creare (building) o allenare (training) un modello, in entrambi i casi poi si passano i dati alla fase di testing di un modello e in base a i suoi risultati si decide se serve tornare alla fase di training e building per aggiustare i parametri del modello, quando la fase di testing è soddisfacente si passa finalmente alla distribuzione del modello.

Questo è un modello molto semplificato del supervised learing, infatti contiene un problema, nella divisione dei dati tra il test di essi e la creazione o allenamento di un modello, come facciamo a essere sicuri quali dati esatti influenzano la performance del modello? Dopotutto abbiamo la possibilità di aggiornare i parametri del modello più e più volte.

Per risolvere il problema i dati sono spesso divisi in **3 parti**: training data, usato per allenare i parametri di un modello, validation data, usati per determinare quali hyperparametri aggiustare, poi ci sono i test data, usati per ottenere una metrica finale dalla performance del modello, quest’ultimi tenderanno ad essere più diversi rispetto ai dati per il training e la validation visto che non potrai tornare a quelle fasi precedenti per essere sicuro della performance del modello. Di base il processo saraà questo: trovi dei dati adatti per la fase di training di un modello, poi con i validation data diversi da quelli del training vedrai come il tuo modello performa su dati non visti e nel caso torni indietro e aggiusti tuoi hyperparametri, ma quando ti serve una risposta definitiva sulla performance del modello, per esempio per renderla nota al tuo boss si passa alla **fase finale dei test** senza la possibilità di tornare indietro a fare modifiche, quella sarà la performance definitiva del tuo modello su dati davvero mai visti.

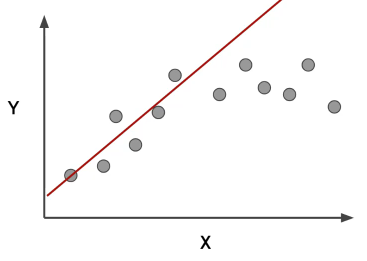
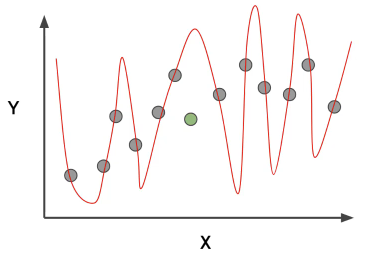
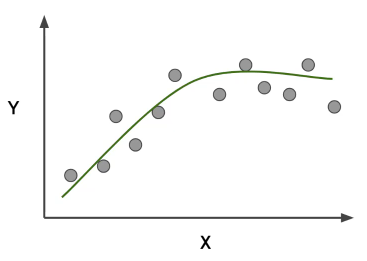
**Overfitting e Underfitting**

L’**overfitting** è quando il modello è troppo adatto ai **training data e fa pochi errori** ma poi nei **test e validitation data fa molti errori**.

Proprio per questo i validitation e test data servono, bisogna capire se il proprio modello non sta andando in overfitting abituandosi troppo a un certo tipo di dati ignorando altre possibilità e non riuscendo a soddisfare in un ambiente reale. Usando un piano cartesiano come dimostrazione dell’andamento dei modelli, un modello fato bene è ondulato in modo sereno,

Nel caso di overfitting il modello è ondulato in modo caotico passando da un estremo all’altro pur di risolvere i problemi.

L’**underfitting** invece è quando il modello non riesce a capire la direzione dei dati e i punti in comune per comprendere cosa deve analizzare, l’esatto contrario dell’overfitting risultando in un bassa variabilità e un alto bias, questo caso è spesso causato da un modello eccessivamente semplice. In un piano cartesiano l’andamento sarebbe direttamente proporzionale andando a ignorare completamente l’andamento dei test.



Modello regolare Overfitting Underfitting

Questo tipo di dati è facile da visualizzare, ma nel caso di dataset a più dimensioni come li visualizzi? Immaginiamo che stiamo allenando un modello e misuriamo i suoi errori col passare del tempo di allenamento.

Un modello fatto bene partirebbe da un asse delle y (errori) molto alto per poi avvicinarsi a quello delle x (epochs) e rimanendo costantemente vicino significando che da tanti errori arriva a farne molto pochi nel tempo.

Un modello fatto male invece già dall’inizio fa pochi errori ma poi nel tempo arriva a farne tanti partendo vicino all’asse x e poi allontanandosi costantemente.

In questo caso però le linee sono di più: una per il training set e l’altra per il testing set, un modello fatto bene ha la linea dei testing set molto vicina a quella dei training set, significa che sia in un ambiente controllato e uno non controllato la quantità di errori è molto simile tra di loro. Nell’overfitting i testing set all’inizio sono anche meglio di quelli del training set ma successivamente peggiorano costantemente allontanandosi dall’asse delle x e dal training set, ciò significa che il modello si è abituato troppo al training set e il modello non riesce a generalizzare dati mai visti prima e quindi dovresti smetterla con l’allenamento.

**Evaluating Performance –**

**Classification Error Metrics**

Ci sono diverse metriche di **classificazione** per misurare la performance di un modello a cui attenersi, essi sono: **accuratezza, richiamo, precisione e F1-Score**.

Di base il un compito il modello avrà due output: la previsione del modello era **corretta o incorretta**.

Fortunatamente corretto e incorretto si espande a situazioni dove avrai molteplici classi, come prevedere categorie maggiori a due. Per esempio avendo categorie A, B, C, D puoi sia aver indovinato quella giusta o sbagliata. Questo può esserre semplificato in ciò che è conosciuto come **classificazione binaria**, ossia una situazione in cui abbiamo solo due classi disponibili, questo può espandersi anche a maggiori classi.

Un esempio è provare a creare un modello che riconosca se un’immagine ha un cane o un gatto. Visto che questo sarà un modello supervisionato prima di **allena** un modello sui **training data**, poi si **testa** il modello con i **testing data**. Dopo che abbiamo le previsioni del modello dai dati del **test\_x** li paragoniamo alle risposte corrette di **valori corretti y**.

Per cui si parte da un modello allenato e ora tocca provare la performance del modello, prendiamo un immagine dal test\_x (per esempio la foto di un cane), e etichettiamo come valore del test\_y appunto come “cane”, il nostro modello farà una previsione che paragoniamo alla etichettatura del test\_y, il risultato per l’appunto o sarà corretto o sarà sbagliato visto che il modello sa riconoscere solo cani o gatti.

Questo processo viene ripetuto per tutte le immagini del test\_x, alla fine avremo un conteggio delle volte che era corretto e le volte in cui non lo era, il punto chiave che tocca fare è: **non tutti i valori corretti o incorretti hanno lo stesso valore**.

Nel mondo reale una singola metrica non specifica tutte le circostanze intorno ad essa, per questo servono delle metriche costanti che possano misurare le circostanze di tutti i casi, organizzeremo i nostri valori previsti con quelli reali in un **confusion matrix**.

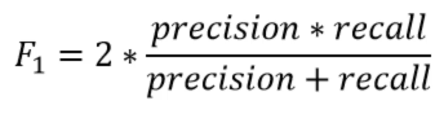
Partendo dall’**accuratezza (accuracy)**, essa è nella classificazione di problemi il **numero di previsioni corrette** fatte dal modello divise per il **numero totale di previsioni**, ossia successi diviso totale, è la metrica più comune e semplice da capire. Per esempio se test\_x **aveva 100** immagini e ne **indovina** **80**, allora il nostro rapporto è **80/100** per cui un’accuratezza dello **0.8 o 80%**. L’accuratezza è comoda quando i le classi bersaglio sono ben bilanciate, ossia una proporzione tra immagini di cani e quelle di gatti uguali o quasi. Ma nel caso di classi **non bilanciate** questo parametro non è una buona scelta, se si avessero 99 immagini di cani e solo 1 di gatti, se il modello indovina solo le immagini con **cani** allora la sua accuratezza sarebbe altissima del 99%. In questa situazione serve distinguere bene il **richiamo**  e la **precisione**.

Il **richiamo (recall)** è l’abilità del modello di riconoscere tutti i casi rilevanti in un dataset. La definizione precisa è il numero dei veri positivi **diviso per** il numero dei veri positivi **sommato ai falsi negativi**.

La **precisione (precision)** è l’abilità di classificazione del modelli di indentificare solo i data point rilevanti. Essa è definita con il numero dei veri positivi diviso il numero dei veri positivi **sommato ai falsi positivi**.

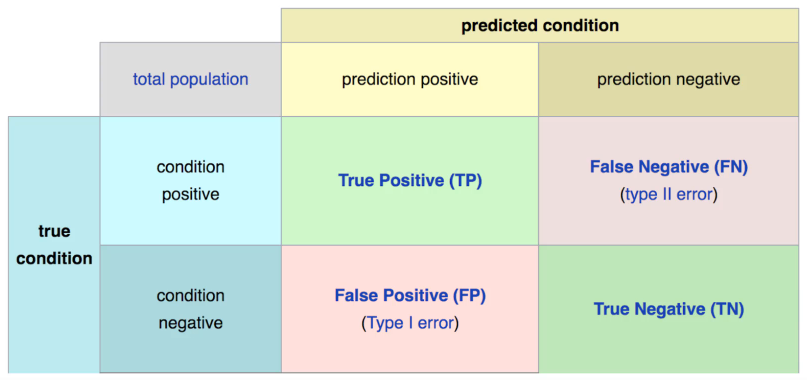
A volte si dovrà decidere quale tra recall e precision sia più importante visto che il recall è l’abilità di trovare tutte le istanze rilevanti in un dataset di contro la precision esprime la proporzione dei data point che il nostro modello dice essere rilevanti e sono davvero rilevanti, ossia recall per vedere quante volte **non riconosce qualcosa di giusto** e precision per vedere quante volte **da per giuste cose sbagliate**.

L’**F1-Score** è il caso dove vogliamo trovare un buon paragone tra recall e precision, possiamo combinare le due metri per ottenere il cosidetto F1-Score. Esso è la media armonica tra precision e recall prendendo entrambe le metriche nella seguente equazione:



Usiamo la media armonica invece della normale media perché punisce i valori estremi. Per esempio se hai un classificatore di precision 1.0 (un caso di precision perfetta) e un recall di 0.0 (un caso di recall pessima), una media normale sarebbe 0.5 mentre un F1 è 0, per cui ti rende più semplice capire quando considerare entrambe o solo una delle due.

Possiamo vedere anche tutte le classificazioni corrette contro tutte le classificazioni incorrette nella forma di un confusion matrix, per esempio:



Abbiamo le vere condizioni (che sono o positive o negative) e le condizioni previste (anche esse positive o negative) che si possono incrociare in condizione vera o falsa (T o F) e predizione positiva o negativa (P o N), da esse ricaviamo unendo le lettere in TP (true positive, vero positivo), FP (false positive, falso positivo), TN (true negative, vero negativo) e FN (false negative, falso negativo).

Per fare un esempio di ogni caso usando cani e gatti, un TP (true positive) sarebbe un caso in cui il modello dice che un cane è un cane, un FP (false positive) è un caso in cui un modello dice che un gatto è un cane, un TN (true negative) è quando il modello dice che un gatto non è un cane e infine FN (false negative) è quando il modello dice che un cane non è un cane, in sintesi quando è positivo il modello sta dicendo che c’è una corrispondenza, quando è negativo significa che non ci sta una corrispondenza, alla fine il risultato di queste previsioni è vero o falso. Nell’ambito della statistica quando il modello sbaglia la sua previsione si parla di Type I error per gli FP e Type II error per gli FN. In questa matrice puoi metterci come parametri anche l’accuratezza nella stessa categoria delle vere condizioni e la prevalenza nella categoria delle condizioni previste.

In sintesi il confusion matrix è un modo per paragonare le previsioni ai valori veri, ma una buona metrica dipende dalla situazione specifica, quindi anche se si ha una buona accuratezza (anche qui, cosa determina una buona accuratezza?) potrebbe non essere rilevante nel caso specifico di utilizzo, per esempio se ho creato un modello per prevedere la presenza di malattie la presenza di malattie è ben bilanciata nella popolazione generale? Probabilmente no, è difficile avere classi bilanciate in un setting reale.

Spesso i modelli sono usati come una diagnosi rapida **prima** di fare un test più elaborato, come un esame delle urine prima di quello della prostata.

Per cui tocca decidere se si preferisce un caso di un modello si focalizza nel riparare i casi di falsi positivi o falsi negativi, meglio riconoscere che qualcuno ha il cancro quando non lo ha, oppure non riconoscere che qualcuno ha il cancro quando lo ha? In ambito medico viene più sicuro dirsi di riconoscere bene i falsi negativi al costo di aumentare anche i falsi positivi. Ovviamente la risposta non è che in ogni caso serve dare più importanza ai falsi negativi, questo dipende da cosa ti dice l’esperto in quel campo a riguardo. **Dipende tutto dal contesto.**

**Evaluating Performance –**

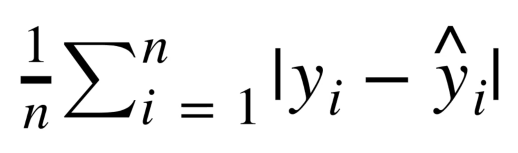
**Regression Error Metrics**

La **regressione** è un compito quando un modello cerca di prevere dei valori continui (quindi diversi dai valori categoriali i quali sono una classificazione), infatti le metriche viste prima come il recall o accuratezza non sono utili quando i valori sono **continui** e non classificati.

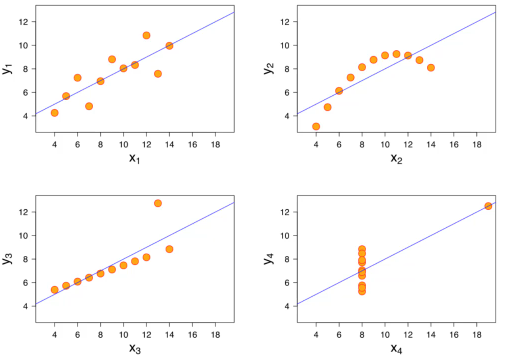
Per esempio prevedere il prezzo di una casa in base alle sue caratteristiche è un **compito di regressione (regression task)**, provare a prevedere il paese nel quale è presente una casa in base alle sue caratteristiche è un **compito di classificazione (classification task)**.

Le principali metriche di valutazione per la regression sono: **errore assoluto medio (mean absolute error), errore quadratico medio (mean squared error) e radice dell’errore quadratico medio (root mean square error).**

Il mean absolute error **(MAE)** come dice il nome è l’errore assoluto medio, in formula sarebbe:

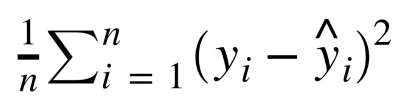


Si paragonano le previsioni con l’etichetta true\_y, ricorda che i valori sono continui per esempio paragonando il prezzo stimato di una casa con quello reale. Si prende la differenza tra il valore vero e quello previsto e da ciò ricaviamo il valore assoluto (se negativo verrà reso positivo), quindi la differenza positiva o negativa che sia, poi ci prendi la media da tutte le previsioni. Il lato negativo del MAE è che non punisce i grandi errori:



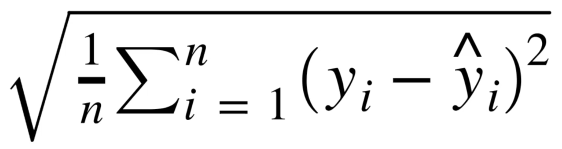
Come da grafici sopra la linea rimane direttamente proporzionale ignorando il pattern dei dati, nell’esempio di x3 la linea segue il pattern tranne per un punto, vogliamo che le nostre metriche considerino anche questi casi.

A questo aiuta di più il mean squared error **(MSE)**, ciò permette rispetto al MAE di calcolare anche gli errori grandi, rendendolo di fatto più popolare, la formula è la seguente:



Quindi si va a prendere la differenza tra il valore vero e quello previsto per poi fare il quadrato del risultato, in questo modo con il quadrato gli errori più grandi risaltano rispetto al MAE, ciò è più popolare perché punirai di più il tuo modello quando sbaglia e non ci serve neanche il valore assoluto visto che facciamo il quadrato anche di un numero negativo, ma resta comunque un problema, fare il quadrato dell’etichetta attuale meno la nostra previsione fa si che si faccia il quadrato di entrambe le unità, per cui il risultato sarà al quadrato rispetto a quello che vorremo.

Il modo di risolvere ciò è con il root mean square error **(RMSE)**, ossia la radice quadrata dell’MSE, la formula resta quasi invariata:



Ciò permette di avere le stesse proprietà del MSE e le stesse metriche all’interno andando a risolvere il problema del risultato al quadrato.

Come nella classificazione anche nella regressione il valore delle metriche **dipende dal contesto**, avere 10$ come RMSE per una casa sembra un affare visto e considerando il prezzo che tendono ad avere le case, ma se lo anche per una caramella c’è un problema.

Quindi serve paragonare la nostra metrica d’errore con il valore medio dell’etichetta nei propri data set per avere un intuizione se la performance generale è buona, anche la **conoscenza dell’ambito** ha un ruolo importante, se non conosco le dinamiche degli agenti immobiliari e l’ambito come faccio a capire se ho una metrica soddisfacente o non? Per cui il machine learning è un’attività anche di processo collaborativo.

**Unsupervised Learning**

Finora abbiamo visto esempi in ambiti tutti supervisionati e valutato la performance in essi nei compiti di classificazione e regressione. Ma in un ambito **non supervisionato**? Finora **le etichette erano conosciute** grazie ai **dati storici etichettati**, ma cosa succede quando non abbiamo storici etichettati? Per esempio abbiamo un set di immagini che non sono stati etichettati come cani o gatti.

Esistono alcuni compiti che ricadono nell’unsupervised learning: il **raggruppamento (clustering)**, il **rivelamento di anomalie (anomaly detection)** e la **riduzione della dimensionalità (dimensionality reduction).**

Il **clustering** consiste nel raggruppare insieme data points **non etichettati** dentro delle categorie o cluster, i data points vengono assegnati in base alla somiglianza, essendo comunque non supervisionato si rischia di avere corrispondenze con etichette che non avevamo in mente, per esempio se vogliamo dividere cani e gatti in due cluster diversi potrebbe accadere che in un cluster ci sono cani e gatti piccoli e nell’altro invece cani e gatti grandi, la nostra idea iniziale non è stata rispettata ma la divisione tra somiglianze (in questo caso di grandezza) è stata rispettata in modo corretto proprio perché non erano etichettati e non c’era una risposta giusta.

L’**anomaly detection** serve come dice il nome a trovare valori anomali nei dataset come per esempio le transizioni fraudolente di una carta di credito, in questo caso non hai uno storico che ti faccia da esempio per capire se un’azione è fraudolenta o no, per cui si cercano dei punti che sono molto dissimili tra loro.

Infine la **dimensionality reduction** sono tecniche che processano dati che riducono il numero di caratteristiche in un dataset, sia per compressione o per una maggior comprensione il trend all’interno del dataset.

Nell’unsupervised learning è importante notare che queste sono situazioni do noi **NON** abbiamo la risposta giusta ai dati storici, il che vuol dire che la valutazione dei dati è molto più difficile e fumosa, quindi tutte le metriche del supervised learning non sono affidabili visto che non si possono paragonare a niente, anche se tu avessi un etichetta dalla quale cominciare non sarebbe più unsupervised learning.

Il processo dei dati è il seguente: si aquisiscono i dati, li si pulisce, li si passa a un modello per allenarlo o costruirlo (non c’è più la fase di test dei dati visto che non c’è modo di avere la risposta corretta a cui paragonarli), poi si esegue una trasformazione sul dataset (anche qui, non si torna indietro rispetto a un ambiente supervisionato) come la dimensionality reduction e infine si distribuisce il modello.