APPRENDIMENTO SUPERVISIONATO

Cosa è il ML? E' la capacità di una macchina attraverso un algoritmo di imparare dai dati (esperienza) ➔ I dati che gli diamo e che analizza gli insegnano dei pattern. Non ne ha conoscenza, ne ha cognizione; il ML non insegna al computer cosa è una cosa, ma come riconoscerlo. Il ML è l'insieme delle tecniche che ci permette di insegnare alle macchine tramite i dati: automatizza l'insegnamento alla machina di determinati pattern.

Esempio: con il ML abbiamo insegnato ad una macchina a riconoscere i tumori, ma in ogni foto scientifica che gli avevano fornito c'era un righello, quindi riconosceva il righello, ma non sapeva cosa era un tumore.

Il ML si divide in due categorie: ML supervisionato e Non supervisionato. Terza branca: reinforcement learning.

Il Supervisionato gli da sia i dati, sia a cosa corrispondono. Il Non supervisionato, solo i dati.

APPRENDIMENTO SUPERVISIONATO

Paradigma di ML in cui il modello impara a prevedere un'etichetta a partire da dati etichettati. Insegna alla macchina a riconoscere quel pattern. Ogni riga di dati contiene un input (feature) e un output (target), che indica cosa rappresenta il dato. Esempio: feature è cane o gatto, l'output è o cane o gatto.

Regressione e classificazione sono due famiglie di modelli di ML supervisionato. La regressione ci dice come cambiano i dati nel tempo, la classificazione divide i dati in gruppi classificabili.

Il ML è alla base di qualsiasi processo di AI. L'AI non è un ambito: è una disciplina scientifica che usa ML come sua base.

ALBERI DECISIONALI

Alberi di scelta che usano le macchine per andare a comprendere la scelta migliore: divido su un albero tutti i dati in modo da avere una precisa collocazione dei loro valori. Dividono lo spazio dei dati in modo gerarchico, dai dati che hanno più valore a quelli che ne hanno di meno. Ogni nodo rappresenta una decisione rispetto ad una soglia. Ogni volta che si crea una previsione concreta si crea una foglia. Gli elementi vengono raggruppati per similarità!

Vantaggi:

1. alta interpretabilità ➔ si può visualizzare con plot_tree
2. Supporta sia classificazione sia regressione
3. Funziona con feature numeriche e categoriche (non numeriche)

Svantaggi:

1. Overfitting: quando cambio i dati, possono esserci problemi, perché tendono ad adattarsi troppo ai dati.
2. Meno performanti dei modelli di ensamble learning (random forest, ecc.)

N.B. di solito abbiamo fra le 5 e 10 foglie.

RETI NEURALI (MULTILAYER PERCEPTRON: MLP)

Sono un tipo di reti neurali. Se il ML è un modello che applica un'analisi, la rete neurale è un insieme di modelli legati fra loro che arrivano ad un obiettivo. Ad esempio, un decision tree può essere uno dei nodi della rete neurale, ma possiamo mischiare anche ML supervisionato e non supervisionato.

MLP sono modelli non lineari composti da strati di neuroni artificiali. Sono capaci di comprendere relazioni complesse fra i dati. La rete neurale quindi serve a montare insieme i pattern, cioè quello che ha capito, per crearne un valore reazionale. Simula un cervello.

Vantaggi:

1. Flessibili
2. Adatti a pattern non lineari
3. Buoni per grandi dataset

Svantaggi:

1. Poco interpretabili ➔ è una black box
2. Richiedono più dati e risorse computazionali
3. Più sensibili alla scelta degli iperparametri: parametri col peso maggiore che vengono scelti automaticamente.

N.B. Sta intorno alle 100 iterazioni! Non abbiamo tanta RAM

Standard scalar: lo utilizzo su x train e x test, scalato rispetto a x train. Y train non si tocca, che sarebbero le etichette (target)! Stai attento agli outlier, eliminali prima.

METRICHE

METRICHE SINGOLE PER CLASSE

- **Precision** = veri positivi rispetto a tutti i positivi predetti (veri o falsi); quando il modello predice questa classe, quante volte ha ragione? Quante volte ha ragione in questa predizione? E' l'indice per il quale sappiamo quante volte il modello ci prende. In

percentuale, rispetto alle volte che ha predetto un positivo, quante volte ci ha preso? Quante volte predice correttamente? Trova l'indice di positività
N.B. La macchina approssima verso il basso (0).

- **Recall** = veri positivi rispetto ai positivi reali (TP / TP + FN). Fra tutti gli esempi, quanti ne ha riconosciuti correttamente? Non dico quante volte ci ha preso, ma quante volte li ha riconosciuti? Rispetto a tutti i veri esempi di quella classe, quanti esempi di quella classe il modello riconosce correttamente? Quante volte riconosce correttamente?
  N.B. La macchina approssima verso l'alto, cioè verso 1.
- **F-1 Score**: media armonica fra precisione e recall. Utile quando le classi sono sbilanciate, o quando vogliamo un compromesso fra precision e recall senza modificare troppo gli algoritmi. L'F-1 score subisce grandi variazioni con piccole modifiche.
  N.B. Abbiamo media sbilanciata verso l'1 (?)
- **Support** = numero di casi reali di quella classe nel test set. Mi aiuta a capire se le classi sono sbilanciate, e quindi a gestire meglio precision e recall. Dà relatività alle metriche.

  Support

  È il numero di **casi reali** di quella classe nel test set.

  - Classe `0` : **15.161 campioni**
  - Classe `1` : **15.158 campioni**

  Qui le classi sono perfettamente bilanciate (buono per l'interpretazione delle metriche).

  N.B. Le metriche sotto lo 0.6 non sono accettabili! Va modificato il modello!

METRICHE COMPLESSIVE

- **Accuratezza**: predizioni corrette rispetto ai campioni totali. E' la metrica globale, ma non affidabile se le classi sono squilibrate o se applico ulteriori modelli in cascata ➔ se creo una pipeline, posso avere 3 molto imprecisi, me ne basta 1 che mi aggiusta le previsioni per farmi dire che il modello è giusto, quando non è vero. Mi dice quante previsioni totali vengono fatte correttamente.
  N.B. Non è affidabile come singola metrica del modello!!

  L'accuratezza ha due categorie:
  - *Macro Avg*: Media semplice tra le metriche delle due classi (ogni classe pesa uguale). Se ha un buon valore, le prestazioni sono uniformi fra le classi.
  - *Weighted Avg*: è la media pesata in base al numero di campioni per classe; coincide con la Macro Avg quando le classi sono bilanciate, e quindi hanno lo stesso numero di esempi

COME INTERPRETARE IL MODELLO

Come capisco se il modello è corretto?

- Verifico l'**equilibrio** fra le classi. Un modello è equilibrato quando le prestazioni fra classi sono simili. Per capirlo è utile guardare il Supporto: supporti simili indicano equilibrio, perché le classi sono bilanciate (ma non è una verità assoluta!). Inoltre altro fattore di equilibrio sono valori simili.
- **Recall,** leggermente più alto sulla classe 1
- **F-1 Score** vicino a 0.78 è un buon compromesso, dare priorità alla metrica della classe più importante da riconoscere (es. devo riconoscere un tumore).
- **Accuratezza** sopra il 60% va bene. Se è sopra il 90%, si dice che l'accuratezza è "troppo affidabile", cioè sta andando in overfitting. Ottimo risultato fra 65% e 89%.

CLASSIFICAZIONE SBILANCIATA

Che succede se abbiamo un dataset sbilanciato?

Un problema di classificazione è sbilanciato quando una delle classi è più rappresentata delle altre. Se alleniamo un modello in questo modo questo imparerà a predire sempre la classe più comune, con alta accuratezza ma sarebbe inutile.

Perché è un problema?

1. Il modello fatica a generalizzare per la classe minoritaria, che di solito è la piu importante (es. voglio classificare anomalie, guasti, frodi..). I dati divergenti non sono sempre rumore!
2. Le metriche per classe, come precision, recall e F-1 score diventano più importanti: se ho una rete neurale con molti strati, se mi devo basare sugli elementi singoli di una rete per fare valutazioni, non ne esci fuori.
3. Il piu grave: alta accuratezza può nascondere un modello inutile!

Cosa possiamo fare? Utilizziamo tecniche di **SAMPLING** = capacità di ridurre o aumentare dei campioni artificialmente.

- **Undersampling** = riduco il numero di esempi della classe maggioritaria, li riporta allo stesso numero di esempi di quella minoritaria. E' molto semplice, ma facendo così però rischio di perdere informazione. Sconsigliato, soprattutto nei dataset molto piccoli. Utile se stiamo usando vettori e valori numerici.

- **Oversampling** = aumenta i campioni della classe minoritaria. Utile perché non perdo informazioni, ma posso introdurre rumori, dati finti, utenti o classi che si sovrappongono uno a uno.

  L'Oversampling posso farlo in due modi:
  - **Random Oversampling**: duplica dati della classe minoritaria. Utile se stiamo calcolando cose "umane", semplici.
  - **SMOTE** = genera nuovi dati sintetici della classe minoritaria tramite interpolazione, quindi unendo un dato da uno e un dato dall'altro.

  Duplicare Vs Repiclare:

- **Class weighting** = alcuni modelli (es. random forest) possono dare un peso maggiore agli errori sulla classe minoritaria per bilanciare i pesi sulle classi. Quindi avremo una parità di peso artificiale. E' un'opzione molto efficacie, ma non adattabile a tutti i contesti.

Quando scegliere ogni tecnica?

- Pochi dati ➔ SMOTE, mi serve aumentare almeno un po' i dati. Oppure anche class_weight, ma meglio SMOTE.
- Tanti dati ➔ Undersampling. Prima fare pulizia dei dati per capire il range; ad es. limare in un certo range. Ad esempio se ho dati da 0 a 1000, elimino da 500 a 1000. Se non dico quali eliminare, li elimina a caso.
- Modelli semplici ➔ Class_weight
- Pipeline complessa con ensamble ➔ SMOTE, perché la pipeline deve essere coerente, non posso eliminare dati che mi servono dopo.
- Dataset rumoroso ➔ class_weight. Non posso "pulire" il dataset, se no vado in overfitting. Quindi bilanco i pesi: gli elementi rumorosi rimangono, ma avranno un peso così basso da essere poco influenti sul peso totale.

REGRESSIONE LOGISTICA

E' un modello LINEARE, che va a fare una classificazione binaria. E' un modello che calcola la probabilità che un'osservazione appartenga alla classe 1 tramite la funzione logistica (sigmoide). Più dati analizziamo, piu possiamo prevedere a lungo termine se un certo dato sarà 0 o 1. L'output è una probabilità, che ha una soglia di decisione a 0.5 per assegnare la classe. Questa soglia può essere modificata.

La regressione va a prevedere nel tempo i valori dei prossimi dati in una retta. Quindi c'è una separazione lineare tra le classi ➔ C'è una linea, o sei sopra, o sei sotto. E' l'unico modello che considera rumore elementi pari a 0.5, in generale alla mediana.

Quale è il suo limite? Deve essere per forza riprodotta su un piano 2D o 3D. Se non può essere descritta su un piano lineare, non può essere utilizzata.

RANDOM FOREST

Modello di **ensamble di alberi.** Il Forest costruisce tanti alberi, ognuno su un sottocampione dei dati, e ne media le previsioni. L'albero finale che decide se ogni elemento è 0 o 1 è scelto con voto di maggioranza, quindi non è preciso.

Quindi se ho molti dati, il random forest lavora bene; se ho pochi dati, lavorerà male, a meno che non andiamo a bilanciare i pesi ➔ è sensibile allo sbilanciamento delle classi.

Ogni decision tree cerca di essere il piu omogeneo possibile, feature e dati simili fra loro ➔ Ogni albero si distanzierà poco dal margine della foresta, fino all'ultimo albero che sarà totalmente distante dal primo.

Random Forest riduce l'overfitting degli alberi singoli, ma aumenta l'ambiguità.

La frontiera decisionale del random forest è frastagliata ➔ segue la forma dei dati. E' una forma NON lineare, ma a gradini, perché ogni albero crea regole di soglia. E' l'esasperazione dell'albero decisionale: io mescolo tutte le feature per trovare il target corretto di ogni singolo valore, per voto di maggioranza.

Come funziona?

Il Random Forest costruisce molti alberi, ognuno che si basa su un sottoinsieme casuale di dati (bagging) e un sottoinsieme casuale delle feature, tramite feature sampling.

Poi ogni albero viene addestrato indipendentemente con il suo sottoinsieme casuale di dati e di feature.

Quando arriviamo alla predizione finale:

- Classificazione: maggioranza degli elementi totali
- Regressione: media dei valori predetti

Perchè funziona?

1. Perchè gli alberi singoli hanno alta varianza e quindi tendono all'overfitting. Mescolando invece gli alberi con caratteristiche randomiche e dati randomici, la foresta non avrà overfitting, perché sto mescolando i dati.
2. Aggregazione di alberi diversi riduce varianza e generalizzazione del codice: se ripeto con dati diversi, miglioro come funziona la singola foresta
3. Il risultato è un modello piu stabile accurato e robusto rispetto ai singoli alberi

XGBOOST (EXTREME GRADIENT BOOSTING)

E' un ensamble di alberi, ma li costruisce in sequenza ➔ ogni nuovo albero tenta di correggere l'errore di quello prima, ottimizzando una funzione obiettivo tramite il gradiente discendente.

La predizione finale è una somma pesata di tutti gli alberi: anche se l'ultimo albero è il migliore, somma l'output pesato di tutti gli alberi.

Il codice è molto complesso. Inoltre non riporta il migliore degli alberi, ma la somma: quindi se sto bilanciando i pesi è inutile. Tuttavia, è molto adattabile a tutti i contesti e può essere usato in combinazione di altri modelli. La frontiera è più liscia ➔ anche al cambiare dei dati, il risultante è circa similare, non ha variazione frastagliata per adattamento ai dati.

Limite: nonostante crei l'albero migliore, lui riporta alla media fra tutti gli altri ➔ blackbox.

Differenze sintetiche

Modello	Lineare	Interpretabile	Overfitting	Richiede tuning	Flessibilità
Logistic Regression	Sì	Alta	Basso	Basso	Bassa
Random Forest	No	Media	Medio	Basso	Alta
XGBoost	No	Bassa	Medio-Basso	Alto	Molto alta

Grafico dei vari modelli:

- **Logistic Regression** separa i punti con una **retta pulita**
- **Random Forest** crea zone quadrate o rettangolari (regole a soglia)
- **XGBoost** disegna una frontiera più **curva e complessa**, seguendo meglio i dati

SPLIT DEL DATASET IN TRAIN, VALIDATION E TEST

Abbiamo tre parti, di solito divisi il 70%, 15%, 15%

1) Train = per allenare il modello
2) Validation = sceglie gli iperparametri e confronta i modelli
3) Test = valuta il modello finale su dati mai visti

Perché è importante?

1. E' uno dei pochi modi per evitare l'overfitting sulle metriche (non sul modello! Il modello potrebbe essere in overfitting un po'). Ma sulle metriche, potremmo ottimizzare il modello su quei dati; tramite il validation, evitiamo che le metriche mentano.

2. Permette confronti equi ➔ il test set rimane congelato! Non viene utilizzato durante il tuning, ma viene usato solo alla fine per la verifica.

TECNICHE DI VALIDAZIONE

K-FOLD CROSS VALIDATION

Tecnica per stimare le prestazioni di un modello utilizzando tutti i dati disponibili al modello. E' molto più efficiente rispetto al semplice split precedente, ma è molto piu dispendioso.

Di solito si divide il dataset in k-parti di uguale dimensione, chiamate fold. Per ogni iterazione, noi abbiamo il primo fold per il train e il secondo per il validation, poi si va avanti, finchè tutti i dati sono stati incrociati. A quel punto calcoliamo le metriche di ogni iterazione, e facciamo la media di tutte le metriche ottenute ad ogni validazione.

K-FOLD STRATIFICATO

Si usa nella classificazione, in modo da mantenere la proporzione originale delle classi. Il dataset quindi sarà sbilanciato, ma questa tecnica garantisce che in ogni fold ci sia un po' di ogni classe. Fondamentale quando si lavora con target binari o multi-classe.

Vantaggi: usa tutti i dati, è più stabile rispetto alla suddivisione train validation e test, e riduce la varianza nella stima delle prestazioni.

Svantaggi: più costoso computazionalmente, perché alleno il modello k volte, che è la divisione totale dei dati.

CURVA ROC-AUC

Curva che rappresenta il trade off fra True Positive e False Positive del modello, che quindi mi dice come sta lavorando nella distinzione fra le classi.

L'AUC (Area sottesa alla curva) è un numero che misura quanto bene il modello distingue fra le classi ➔ da 0.5 a 1 è un modello casuale, sotto 0.5 è casuale o invertito, 1 è il modello perfetto.