**CLASSIFICATORE: KNearestNeighbour Custom**

Come classificatore semplice custom abbiamo deciso di utilizzare il classificatore KNearestNeighbour.

Il **KNN** fa parte dei classificatori **Instance-based** (noti anche come Lazy Learners) i quali sfruttano un apprendimento basato sulla memorizzazione dei dati di addestramento forniti, utilizzandoli come riferimento e confronto per la classificazione di nuove istanze.

Questo confronto utilizza una particolare **misura di prossimità**: usano la distanza o similarità tra i vari esempi.

Per creare e utilizzare un modello come il KNN per prima cosa è necessario memorizzare al suo interno l’intero training set (questa in pratica è la sua fase di addestramento) dopodichè per ciascun nuovo record da classificare viene calcolata la sua **distanza** da ciascun oggetto del training set e infine si individuano i k record più vicini: al nuovo record è assegnata la classe prevalente tra questi ultimi.

Abbiamo deciso di utilizzare questo classificatore perché:

* Non richiede l’induzione di un modello, con conseguente risparmio di risorse e **semplicità**.
* Non fa ipotesi sui dati (visto che si basa solo sulla misura di prossimità), di conseguenza è possibile utilizzarlo in un’ampia gamma di problemi, anche quando il dataset non è divisibile in maniera lineare (quindi è molto **flessibile**)
* Su dataset piccoli può comunque mantenere una certa accuratezza, che altri algoritmi invece fanno fatica a mantenere.

Ovviamente teniamo in considerazione anche i suoi lati negativi ovvero:

* Ogni volta occorre calcolare la prossimità tra il nuovo record e gli esempi del training set e ciò richiede più risorse computazionali rispetto ad altri modelli.
* Richiede molta memoria per memorizzare tutti i dati all’interno.
* Potrebbe ignorare o considerare meno attributi con una scala di valori inferiore, di conseguenza, è necessario quindi una normalizzazione o standardizzazione.

Il punto cruciale quando si vuole utilizzare un classificatore di questo tipo è quindi effettuare un adeguata normalizzazione degli attributi e individuare il valore di k e la misura di prossimità più adatti.

**REALIZZAZIONE E TEST CLASSIFICATORE**

Nella pratica tutto ciò è stato implementato in codice python senza l’utilizzo di alcuna funzione esterna e per gli iperparametri è stato scelto di ottimizzare il numero di elementi più vicini per determare la classe ‘k’, il tipo di distanza da utilizzare durante il calcolo delle distanze e infine se pesare o no le distanze (quindi il tipo di peso da utilizzare).

Abbiamo creato la classe **‘knnCustom’** che accetta in ingresso il training set, gli iperparametri del classificatore e il numero di fold con cui fare una cross validation. Dopodichè all’interno abbiamo:

* la funzione **‘calcolo\_distanza’** che accetta il training set, il test set e la distanza con cui fare il calcolo; restituisce una lista di liste con all’interno il valore della distanza tra ogni record del training set e quello del test set corrispondente.
* La funzione **‘knn’** che accetta il training set, i record del test set e gli iperparametri scelti; restituisce una lista con la predizione della classe per ogni record del test set.
* La funzione **‘calcolo\_accuratezza’** che accetta le predizioni del classificatore e le effettive classi dei record del test set; restituisce l’accuratezza del modello.
* La funzione **‘tuning\_con\_cross\_validation’** non riceve niente in ingresso e in pratica restituisce la media dell’accuratezza ottenuta durante un cross validation effettuata sul training set con i parametri ottimali individuati.

Quindi in pratica dopo aver creato il classificatore abbiamo effettuato il calcolo dell’accuratezza media, sul training set, ottenuta da un cross validation che utilizza i migliori iperparametri (con l’apposita funzione citata) dopodichè abbiamo creato un nuovo classificatore utilizzando proprio questi ultimi e effettuato le predizioni sul test set calcolandone l’accuratezza.

**RISULTATI**:



**ANALISI DEI RISULTATI:**

A quanto pare, i risultati sul test set risultano pessimi mentre quelli sul training set, nonostante non siano comunque ottimali, sembrano essere decisamente migliori. Il modello, quindi, non riesce a adattarsi completamente ai dati e decisamente fa fatica a predire le classi di nuovi record.

I migliori iperparametri durante le predizioni sono **k=1**, distanza utilizzata= **distanza euclidea** e peso utilizzato= **peso uniforme**. Quindi questo classificatore sembra performare meglio quando valuta un solo elemento vicino.

Questo però, come si può anche osservare poi nell’accuratezza sul test set, porta a **overfitting** perché il classificatore si è adattato troppo ai dati e non ha di conseguenza una buona capacità di generalizzazione.

Ricordiamo poi che il modello è addestrato su **dati non normalizzati** è ciò può aver diminuito considerevolmente le capacità predittive di questo tipo di classificatori. Tra gli attributi, infatti, c’è una grande differenza di scala di valori ed essendo il KNN un classificatore che si basa sulle distanze tra i punti dati ciò comporta che tali attributi dominino, in modo eccessivo, il calcolo delle distanze con conseguente diminuzione di importanza (nella fase predittiva) di attributi con scale inferiori. Quindi se questi ultimi attributi sono effettivamente efficaci nel discriminare i record, ora come ora non siamo in grado di sfruttare queste informazioni.

Un altro problema evidente è lo **sbilanciamento del dataset** che comporta appunto classi con pochi record. Il nostro classificatore potrebbe aver assegnato il record alla classe errata data proprio questa disparità di elementi. Comunque, avendo un k molto piccolo, dovremmo aver perlomeno diminuito tale possibilità. Sarà comunque necessario quindi un successivo bilanciamento del dataset.

Per quanto riguarda il numero di attributi, di sicuro con 12 attributi la dimensionality curse incide sulla performance, e oltre a quello c’è da contare che che molti di tali attributi sono estremamente correlati tra loro il che comporta che insieme abbiano un peso maggiore poi nella effettiva predizione (rispetto agli attributi non correlati).

Si suppone infine che le caratteristiche utilizzate potrebbero non essere sufficientemente distintive e quindi il classificatore non riesca a dividere i record adeguatamente (problematica che potrebbe comunque rimanere dopo tutte le fasi di preprocessing).

Vedremo quindi se con il preprocessing tutte queste problematiche potranno essere risolte o perlomeno gestite così da ottenere una performance migliore.