**­­­­­­LA PREVISIONE DEL DIABETE TRAMITE ALGORITMI DI MACHINE LEARNING**

Fumagalli Mattia, Morzenti Giacomo, Valsecchi Matteo

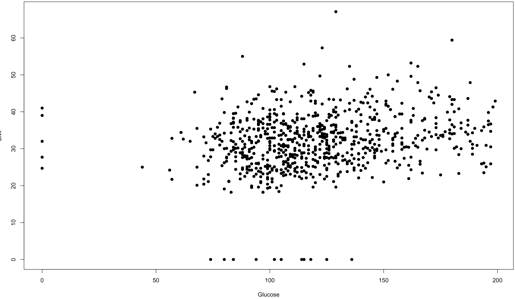
**INTRODUZIONE**

Il dataset utilizzato per le nostre analisi è denominato “Diabetes Dataset” ed è scaricabile al seguente link: “<https://www.kaggle.com/datasets/mathchi/diabetes-data-set>”. Questo insieme di dati da noi analizzato è composto da 768 osservazioni e 9 variabili, dove ogni osservazione rappresenta un individuo su cui sono state effettuate delle rilevazioni mediche per prevedere la positività o meno al diabete. Tutte le variabili sono state rese di tipo numerico. Questo insieme di dati è stato estratto da un data set più ampio realizzato dal “National Institute of Diabetes and Digestive and Kidney Diaseases” e in particolare si è voluto selezionare solamente le osservazioni relative a pazienti donna di almeno 21 anni e di origine indiana “Pima”.

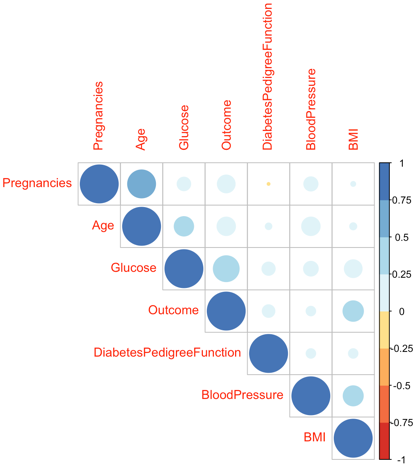
L’obiettivo delle analisi svolte a partire dall’insieme di dati indicato è quello di prevedere, sulla base di misurazioni diagnostiche relative ad alcuni parametri corporei e clinici, se un paziente è affetto dal diabete.

**PRE-PROCESSING**

Per le nostre analisi di Machine Learning abbiamo deciso di eliminare alcune features poco significative per i nostri scopi oppure che presentavano un gran numero di elementi mancanti: in questo dataset i missing values erano rappresentati da valori nulli per variabili del corpo umano in cui la quantità 0 non è realisticamente possibile, come per esempio il livello di insulina o lo spessore della cute del tricipite.



Come si osserva dallo scatterplot precedente, nel nostro insieme di dati si evidenziano delle osservazioni problematiche. Nelle nostre indagini abbiamo affrontato la presenza di missing values in due modi diversi: nel caso delle variabili *Insulin* e *SkinThickness* la proporzione di valori mancanti rispetto al numero di osservazioni era molto alta (52% e 32%), per cui abbiamo deciso di eliminare direttamente tali variabili; nel caso delle restanti variabili i missing values risultavano essere relativi a sole poche osservazioni, per cui abbiamo deciso di eliminare tali righe del dataset.



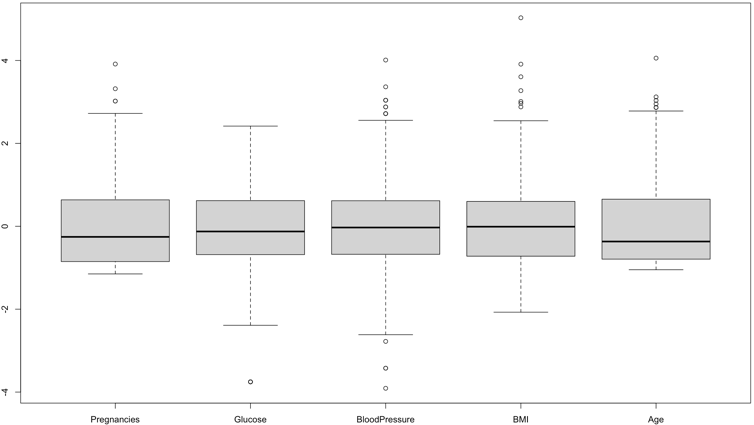
Grazie al corplot sopra rappresentato abbiamo inoltre optato per l’eliminazione della variabile *DiabetesPedigreeFunction*, siccome era poco correlata con la variabile risposta.

Terminate le attività di pulizia e conoscenza del nostro insieme di dati su cui operare, il nostro dataset presenta 724 osservazioni e 6 variabili. Ognuna delle 724 osservazioni rappresenta una singola donna di almeno 21 anni e di origine indiana “Pima” su cui sono stati rilevati dati corporei per osservare se è affetta da diabete oppure non lo è. Le 6 variabili, invece, sono descritte nel seguito:

* *“Pregnancies”* (gravidanza): indica il numero di volte in cui la donna è stata in gravidanza
* *“Glucose”* (glucosio): indica la concentrazione di glucosio nel Plasma due ore dopo un test di tolleranza al glucosio svolto in modo orale. È misurato in mg/dl e per la specie umana i soggetti sani hanno valori compresi tra 70 e 99. Un valore compreso tra 100 e 125 è indicativo di uno stato di pre-diabete, mentre un valore pari o superiore a 126 nella maggior parte dei casi è segno di diabete
* *“BloodPressure”* (pressione sanguigna): indica la pressione sanguigna diastolica, cioè il valore della pressione arteriosa nel momento in cui il cuore è in fase di rilassamento, e si misura in mm. Per l’individuo umano i valori di questo parametro devono essere normalmente compresi tra 80 e 84
* *“BMI”*:è l’indice di massa corporea, dato dal rapporto tra il peso in kilogrammi kg e il quadrato dell’altezza in metri m. Per le donne i valori ottimali di BMI sono nel range 18.7-23.8. Se il BMI è minore di 18.7 l’individuo femminile è sottopeso, se invece è superiore al range si verifica uno stato di sovrappeso che per valori molto elevati diventa obesità
* *“Age”* (età) misurata in anni
* *“Outcome”* (risultato): questa variabile rappresenta la variabile risposta, cioè la classe che viene associata al paziente sulla base dei diversi parametri clinici rilevati. Questa variabile classe assume valori 0 oppure 1: il livello 1 si interpreta come “paziente positivo al diabete” cioè l’individuo è affetto dalla malattia, mentre 0 indica “paziente negativo al diabete”

Per prima cosa osserviamo la distribuzione delle due classi nella variabile *Outcome*. Si nota che 475 osservazioni appartengono alla classe 0 e 249 istanze sono contenute nella classe 1: è possibile dunque definire che le classi sono abbastanza bilanciate.

Considerando che nel nostro lavoro di analisi si utilizzeranno metodi di clustering distance-based, è importante valutare la standardizzazione o la normalizzazione delle features incluse nel dataset. Le variabili riportate hanno infatti dei range molto diversi tra loro e conseguentemente avranno pesi differenti nei vari algoritmi applicati. Abbiamo deciso di applicare la normalizzazione e di seguito è stato riportato il boxplot delle variabili normalizzate, attraverso il quale si possono anche individuare eventuali outliers. Tale procedura di modifica dei valori non è stata applicata alla variabile risposta *Outcome.*



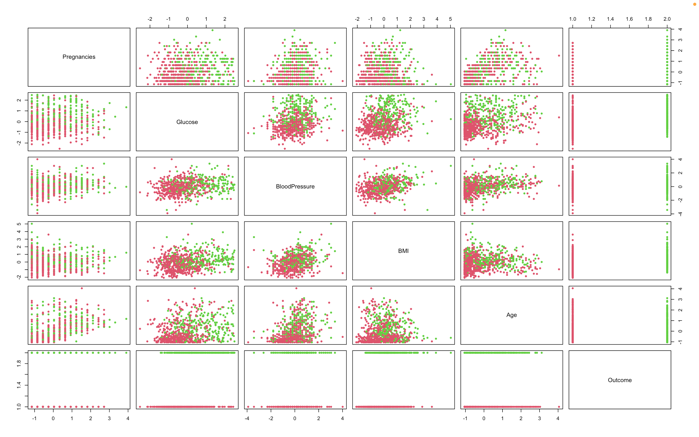
**CLUSTERING**

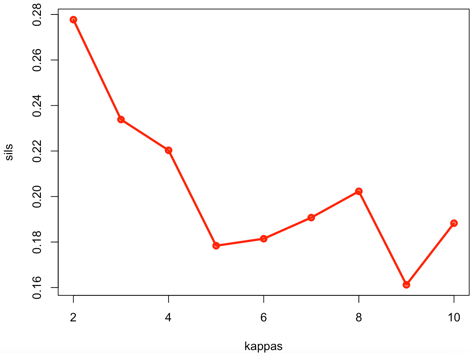
Nel nostro lavoro di analisi abbiamo eseguito le operazioni di clustering utilizzando i metodi K-Means, H-Clust, soft clustering e DbScan.

* **K-MEANS**

Per effettuare le operazioni di K-Means abbiamo utilizzato i dati normalizzati in quanto questo algoritmo si basa sulla misura di distanza Euclidea, per cui le variabili con elementi con varianze differenti potrebbero essere difficilmente paragonabili tra loro.

L’algoritmo K-Means produce due clusters di numerosità simile, per cui le istanze sono classificate in modo più omogeneo rispetto al raggruppamento reale. Dall’osservazione dei vari scatterplot per le diverse coppie di variabili, abbiamo optato per selezionare *BMI* e *Glucose* perché il grafico che coinvolge tali features è quello che rappresenta meglio la suddivisione tra i due gruppi.

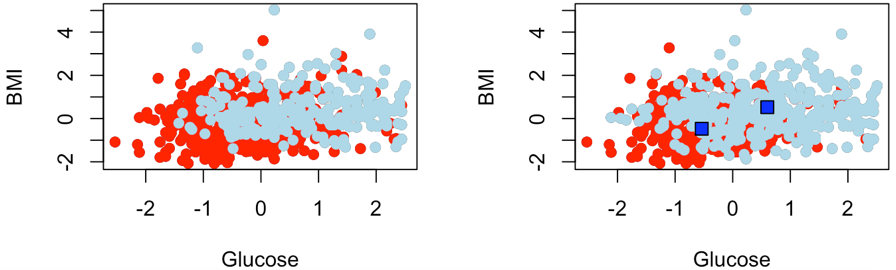


La prima valutazione sul nostro risultato di clustering è stata effettuata utilizzando un grafico con cui abbiamo confrontato l’andamento della varianza all’interno dei gruppi al variare del numero k di cluster in cui le osservazioni sono state divise. L’output presenta una curva in cui il più netto cambiamento di pendenza è in corrispondenza di k=3, per cui ci suggeriva di suddividere le istanze in 3 gruppi separati: ciò si scontrava, però, con l’effettiva classificazione dei punti, poiché i pazienti analizzati possono essere solamente ripartiti in positivi o negativi senza classi intermedie. L’andamento dalla silhouette invece evidenzia che il numero ottimo di gruppi in cui dividere le istanze è 2, in accordo con il raggruppamento reale:

Per valutare la precisione della classificazione abbiamo ricorso ad una matrice di confusione in cui si confrontano le classi previste con le corrispondenti classi reali date dalla variabile *Outcome.* Da questa confusion matrix possiamo ricavare tre indicatori utili per osservare la performance di clustering:

* Accuracy: è il numero di previsioni corrette sul totale delle osservazioni
* Sensitivity: è il numero di veri positivi (diabetici) previsti sul totale dei positivi
* Specificity: è il numero di veri negativi (non diabetici) previsti sul totale dei negativi

In questo caso l’accuracy è 0.70, la sensitivity è 0.75 e la specificity è 0.67. In base a questi risultati possiamo affermare che il nostro classificatore classifica meglio le donne positive al diabete: il numero di individui positivi previsti correttamente è infatti superiore al numero di positivi previsti in modo errato. Questo fenomeno si verifica anche per le etichette negative ma in proporzioni ridotte.



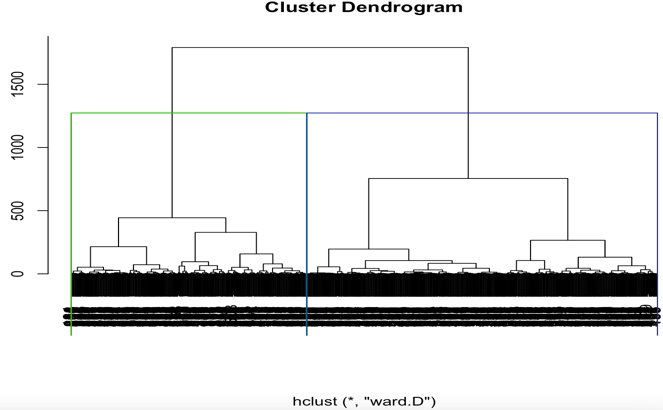
* **H-CLUST**

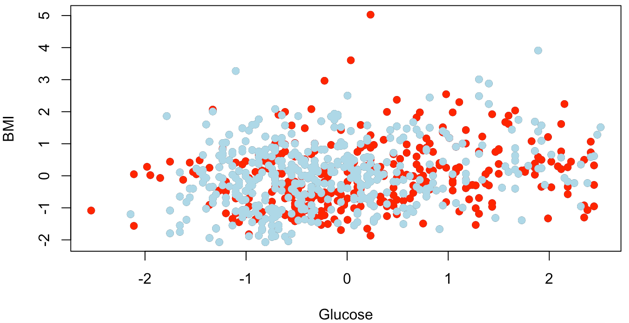
H-Clust, cioè l’algoritmo gerarchico, si basa sulla misura di distanza per cui anche in questo caso abbiamo lavorato con dati normalizzati.

L’algoritmo gerarchico agglomerativo unisce gli elementi con distanza inferiore. Come primo passaggio abbiamo calcolato la matrice di distanze. RStudio fornisce diversi metodi per valutare la distanza e nel nostro operato ne abbiamo utilizzati 6: “*complete”*, “*average*”, “*single*”, “*ward.D*”, “*ward.D2*” e “*mcquitty*”.

Il metodo che ci fornisce i risultati migliori è *ward.D* che ha un’accuracy pari a 0.66, una sensitivity di 0.89 e una specificity di 0.54. Questa opzione di clustering è stata proposta da Joe H.Ward: egli ha suggerito una procedura di raggruppamento gerarchico agglomerativo in cui il criterio per la scelta della coppia di cluster da unire ad ogni passaggio si basa sul valore ottimale di una funzione obiettivo, data dalla somma dell’errore dei quadrati. La differenza tra *ward.D* e *ward.D2* consiste nel fatto che la seconda opzione è l’implementazione diretta del metodo di Ward, mentre la prima deve prevedere l’elevamento al quadrato delle distanze euclidee prima di inserire il comando hclust(). Di conseguenza, entrambi i metodi producono lo stesso risultato essendo analoghi.

Di seguito sono riportati due grafici: il dendogramma che rappresenta i gruppi creati in modo agglomerativo e il plot che rappresenta la dispersione dei punti previsti dall’algoritmo

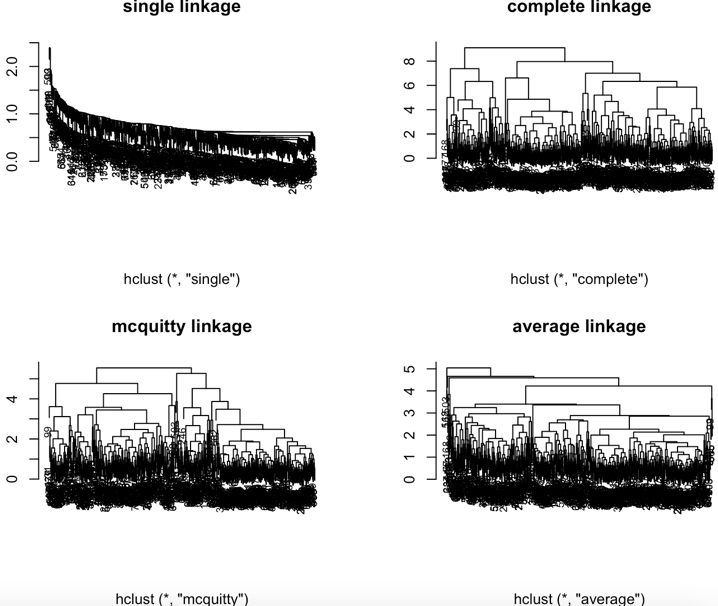




Dal dendogramma si nota una netta separazione tra due grandi clusters: *ward.D* e *ward.D2* sono gli unici metodi che riconoscono tale separazione in modo netto, mentre per tutte le altre opzioni provate si osserva un solo cluster oppure uno dei due gruppi risulta degenere.

Sono di seguito riportate le accuracy per i diversi metodi provati e i loro relativi dendogrammi:

|  |  |  |  |
| --- | --- | --- | --- |
| METODO H-CLUST | ACCURACY | SENSITIVITY | SPECIFICITY |
| *complete* | 0.56 | 0.92 | 0.37 |
| *average* | 0.34 | 0.99 | 0.002 |
| *single* | 0.34 | 1 | 0.002 |
| *ward.D* | 0.66 | 0.89 | 0.54 |
| *ward.D2* | 0.66 | 0.89 | 0.54 |
| *mcquitty* | 0.58 | 0.59 | 0.58 |



* **SOFT CLUSTERING**

L’attività di clustering prosegue con la parte relativa al soft clustering. Questo metodo è anche definito “mixture-based generative model” e prevede che ogni punto sia generato da k distribuzioni G1,…,Gk dove ogni distribuzione Gi rappresenta un cluster. La descrizione delle distribuzioni può essere fatta analizzando i parametri che le caratterizzano e tali misure si possono determinare con la stima di massima verosimiglianza. L’algoritmo iterativo adatto per fare questa operazione è chiamato EM, cioè expectation-maximization: l’output finale che si ottiene è il mixture-model ottimo che descrive i nostri dati, in cui ogni distribuzione che lo compone rappresenta un cluster.

In questo caso abbiamo utilizzato i dati non normalizzati perché si usa come criterio la massima verosimiglianza e non la distanza: di conseguenza, grandezze diverse delle varianze saranno adattate dalla procedura di ottimizzazione.

I cluster risultanti da questo algoritmo sono bilanciati e si distribuiscono molto più omogeneamente rispetto ai dati iniziali, perciò è difficile riconoscere due gruppi diversi. Otteniamo, infatti, un’accuratezza molto bassa e pari circa a 0.35, per cui questo metodo di clustering è poco efficace per i nostri dati.

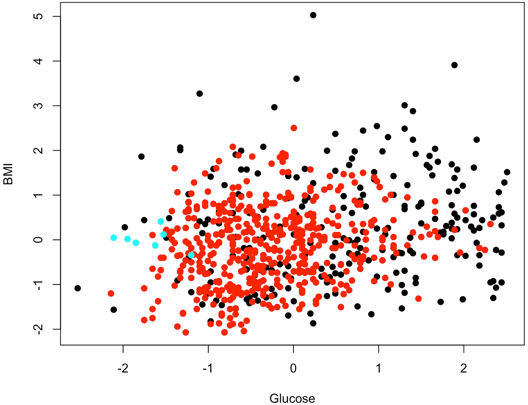
* **DBSCAN**

L’ultimo algoritmo di clustering che abbiamo impiegato è il DbScan. Questo metodo di raggruppamento, a differenza degli altri precedentemente analizzati, si basa sulla densità dei punti: un punto si dice denso se contiene un gran numero di punti all’interno di un suo intorno sferico di raggio definito. Definito un raggio ε e un valore soglia τ, DbScan prevede la classificazione dei punti nello spazio in tre categorie:

* Core point: punti che in un raggio ε hanno almeno τ punti
* Border point: punti che in un raggio ε hanno meno di τ punti ma contengono un core point
* Noise point: punti che in un raggio ε hanno meno di τ punti e non contengono un core point

Dopo aver classificato i punti, DbScan produce un clustering dove i cluster riflettono i core point e i relativi border point più vicini: i border point, infatti, sono associati al core point con cui hanno una distanza inferiore. I noise point, invece, rappresentano gli outlier: a differenza di tutti gli altri algoritmi DbScan è in grado di prevedere, gestire e trattare le anomalie.

Per questo algoritmo abbiamo impiegato i dati normalizzati. I due iper-parametri raggio e soglia devono essere inseriti in input nell’algoritmo e nelle nostre analisi abbiamo optato per impostare i valori di ε pari a 0.92 e di τ pari a 8, poiché erano i livelli che ci restituivano il numero relativamente più basso di noise point (in relazione ad altri valori degli iper-parametri). Questa combinazione di iper-parametri non riesce comunque a rappresentare bene i cluster in quanto genera un gruppo di numerosità minima (7). Oltre a tale cluster, si classificano 489 punti come appartenenti all’altro raggruppamento e un numero tuttavia consistente di istanze (228) come noise point/outlier.



Il precedente grafico riflette la dispersione dei punti classificati nei diversi clusters dal DbScan: i colorati in rosso sono appartenenti al gruppo più numeroso, quelli in colore azzurro appartengono all’insieme di dimensione minore e la moltitudine di punti neri sono i noise point osservati.

In conclusione, possiamo affermare che DbScan crea un gruppo di dimensione accettabile di cui però non possiamo valutarne bene l’accuratezza. Inoltre, il secondo gruppo generato ha dimensioni troppo piccole, a favore di un gran numero di punti che diventano outlier.

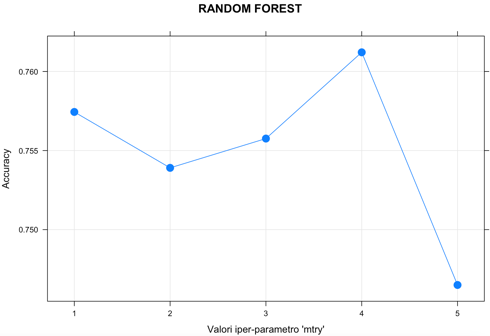
**CLASSIFICAZIONE**

Per la classificazione abbiamo utilizzato i metodi di Random Forest, Support Vector Machine, K-nearest neighbors e reti neurali. Lo scopo in tutti i casi era quello di ricercare gli iper-parametri in grado di minimizzare l’errore di generalizzazione.

Come passaggio iniziale abbiamo deciso di suddividere il dataset di partenza in due sottoinsiemi: un insieme più grande (75% del totale) utile per addestrare il modello e valutare i parametri ottimali, detto training set o insieme di stima, e una porzione più piccola (25% del totale) finalizzata alla misurazione dell’errore di classificazione, chiamata test set o insieme di verifica.

* **RANDOM FOREST**

Il primo metodo che abbiamo utilizzato per la classificazione è il Random Forest: questo è un metodo di ensemble in cui ogni componente è un modello generato tramite alberi decisionali. Random Forest coinvolge due iper-parametri: il numero di variabili da campionare ad ogni passo (*mtry*) e il numero di alberi che compongono la foresta (*ntree*). Impostando come metodo di valutazione dell’errore di generalizzazione la cross-validation e considerando un numero di alberi di default imposto da RStudio pari a 500, il livello ottimale di *mtry* è equivalente a 4. L’accuracy corrispondente sul training set è di 0.761: è più grande di tutte quelle osservate precedentemente nei metodi di clustering, per cui si può ipotizzare che la classificazione sembra essere più adatta per analizzare il nostro dataset rispetto al clustering.

Il seguente grafico rappresenta la variazione dell’accuratezza rispetto al numero di variabili da campionare ad ogni passo della Random Forest.

Dopo individuato il valore ottimo del parametro *mtry* tramite le analisi sul training set, abbiamo generato i valori previsti sul test set e con una matrice di confusione siamo stati in grado di confrontare tali previsioni con le etichette reali delle istanze appartenenti a questo insieme di verifica. I risultati finali riportano un’accuratezza di 0.755, una sensitivity di 0.564 e una specificity di 0.856. È interessante osservare come con la Random Forest l’accuracy sul training set e sul test set sono molto simili fra loro.

Nella Random Forest è presente un certo grado di arbitrarietà che per iterazioni diverse della procedura porta a determinazioni diversa dell’iper-parametro mtry: questa arbitrarietà si può risolvere con il metodo della “leave-one out cross validation” che in questo caso non utilizzeremo perché computazionalmente troppo costosa.

* **SUPPORT VECTOR MACHINE**

Nell’applicare la Support Vector Machine abbiamo deciso di affrontare tre diverse situazioni: support vector lineare, support vector machine non lineare a kernel “polinomiale” e support vector machine non lineare a kernel “radial basis function”.

Applicando la SVM lineare abbiamo ottenuto un’accuracy di 0.75.

Ci siamo poi spostati verso le trasformazioni kernel, secondo il principio per cui se un insieme di dati non è linearmente separabile nell’“input space” esisterà uno spazio a dimensioni aggiuntive, detto “features space”, in cui sarà linearmente separabile. Il metodo che ci consente di mappare i dati in uno spazio di dimensione maggiore tramite un funzione kernel è definito “kernel trick”.

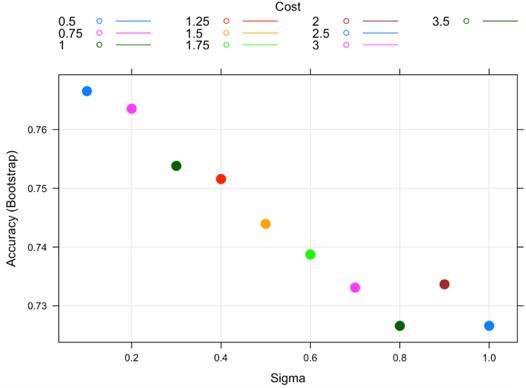
Nel caso del kernel “polinomiale” abbiamo fatto variare il parametro *C*: *C* è il parametro di penalità e vuole indicare quanto è concesso l’errore di classificazione:

* se C è elevato è più importante non effettuare l’errore di classificazione, per cui mi accontento di un margine di separazione meno massimizzato al fine di avere un classification error più minimizzato
* viceversa, se C non è elevato è meno importante l’errore di classificazione, per cui si massimizza il margine di separazione a costo di avere un errore meno minimizzato

Il valore ottimale di C trovato tramite la Support Vector Machine a cui è stato applicato il metodo di “bootstrap” è pari a 0.25. L’accuratezza corrispondente osservata sul training set è 0.766. Date questi risultati ottenuti sui parametri ottimali, procediamo valutando l’algoritmo sul test set. Creando una matrice di confusione che relazione le previsioni su tale insieme con le etichette reali delle istanze del test set, si ottiene che l’accuratezza sull’insieme di verifica è 0.794, la specificità è 0.932 e la sensitività è 0.532.

Nel caso del kernel “radial” è stato inserito anche il parametro *sigma*: *sigma* è un indice di complessità della funzione kernel e definisce la capacità di trasformare l’input space in uno spazio di features space. Tale valore indica l’influenza esistente tra i punti del piano e la retta di separazione. Anche in questa situazione è stato applicato il metodo “bootstrap” e i valori ottimi degli iper-parametri *C* e *sigma* sono rispettivamente 0.5 e 0.1. L’accuracy corrispondente misura sul training set è di 0.768.

Di seguito si può osservare un grafico che rappresenta il variare dell’accuratezza sul training set per la SVM radiale in relazione ai valori assunti dai due iper-parametri:

****

Successivamente si valuta anche in questo caso la performance dell’algoritmo sul test set tramite una nuova matrice di confusione, in cui ora sono inserite le predizioni date dal kernel “radiale”. Osserviamo ora un’accuracy a livello 0.805, specificity pari a 0.932 e sensitivity 0.564.

In conclusione, possiamo affermare che tra i due kernel usati non vi è una grande differenza, infatti l’accuracy è abbastanza simile. La SVM potrebbe inoltre rappresentare l’algoritmo di classificazione migliore per il nostro modello essendo infatti quello che riporta una percentuale di previsioni corrette maggiore tra tutti quelli provati. SVM classifica in modo moderatamente buono i positivi, ossia le donne con *Outcome* uguale a 1: circa 55% di previsioni corrette. Risulta sembrare di gran lunga superiore la capacità della Support Vector Machine di classificare i pazienti negativi: la percentuale di essi prevista correttamente è molto elevata e si attesta intorno al 93%.

* **KNN**

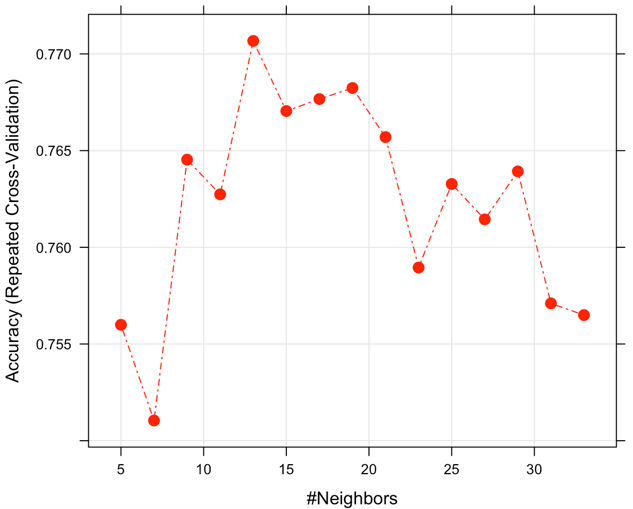
Il metodo KNN K-nearest neighbors non setta parametri e la sua procedura è divisa in tre fasi:

* si sceglie il punto per cui effettuare la previsione
* si determinano i k punti più simili (vicini) al punto su cui si vuole fare la previsione
* si effettua la previsione sul punto considerato basandosi sulle etichette di classe dei k-nearest neighbors relativi alla variabile target considerata

L’unico iper-parametro da determinare in questo caso è il *k*: esso è un intero positivo solitamente non molto elevato e che, in un contesto di classificazione binaria in cui sono presenti esclusivamente due classi, è opportuno scegliere dispari al fine di evitare di ritrovarsi in situazione di parità. In questo caso utilizziamo il metodo di “repeated cross-validation” per cui l’arbitrarietà non è del tutto superata. Effettuiamo un’indagine su 15 valori diversi assegnabili all’iper-parametro e consideriamo l’accuratezza più alta come criterio per determinare il valore ottimo. Imponiamo, inoltre, che il numero di ripetizioni della cross-validation è 3. Dalle nostre analisi risulta che k ottimo è pari a 13 e ciò porta ad un’accuratezza nella classificazione sul training set pari a 0.768. Il valore per il Cohen’s Kappa è di 0.46: non è molto elevato ma è comunque un livello moderatamente accettabile.

Valutando le previsioni sull’insieme di verifica (test set) ottenute tramite l’algoritmo KNN si può costruire una matrice di confusione con cui si confrontano tali previsioni e le etichette reali per le unità che compongono il test set. Si ricava che per tali istanze l’accuratezza vale 0.772, la sensitività 0.580 e la specificità 0.873. È possibile osservare la similarità dell’ammontare dell’accuracy per le analisi sui due insieme in cui il dataset è suddiviso.

Anche in questo caso, così come nella Random Forest, l’accuracy ha un valore superiore rispetto a tutte le procedure di clustering analizzate in precedenza: si potrebbe dunque ipotizzare che la classificazione è più utile per indagare il dataset Diabetes rispetto alle operazioni di ricerca dei cluster.

Il seguente plot rappresenta la variazione dell’accuratezza da noi ottenuta al variare del parametro k: si può individuare che il livello più alto raggiunto dall’accuracy è in corrispondenza di k=13, in accordo con quanto detto prima

**APPROCCIO CONNESSIONISTA: RETI NEURALI**

Come ultimo metodo di analisi dei nostri dati abbiamo utilizzato le reti neurali. In questo parte di lavoro abbiamo suddiviso il dataset in training set e in test set, per allenare il modello e successivamente la bontà sul test set.

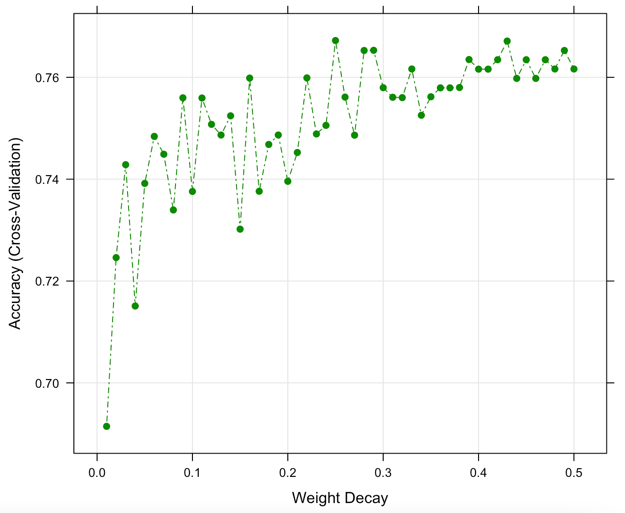
L’approccio connessionista tenta di emulare il modello celebrale dell’essere umano, per cui ciò che è importante non sono le unità fondamentali (neuroni) ma come esse sono collegate. Ogni neurone ha una funzione di attivazione e ne esistono diversi tipi. Ogni neurone riceve degli input da altri neuroni che non sono fissi ma variabili, quindi ad ognuno viene assegnato un peso diverso a seconda dell’importanza della connessione. Questa combinazione pesata viene passata attraverso la funzione di attivazione che produce un output. I singoli sono organizzati in vari layer, o strati. Esistono tre layer fondamentali:

* input layer, che ha un numero di neuroni pari al numero di features
* output layer, che ha un numero di neuroni pari al numero di output previsti
* hidden layer o strato nascosti: i neuroni degli hidden layer svolgono la vera attività computazionale creando le vere connessioni tra i diversi layer. All’aumentare del numero di neuroni in questi strati la rete coglie pattern più complessi ma aumenta il costo computazionale. Inoltre, in base al numero di layer nascosti la separazione dei punti nello spazio sarà più o meno diversa: con 1 hidden layer si suddividono dati linearmente separabili, con 2 si dividono dati in sottospazi convessi e con 3 in qualsiasi sottospazio.

Tramite le operazioni in R ci ponevano lo scopo di determinare gli iper-parametri *size* e *decay*. Il parametro *size* rappresenta il numero di neuroni per hidden layer: nel nostro caso abbiamo utilizzato un solo strato nascosto poiché, nonostante osservando i dati sarebbe stata più adatta una separazione non lineare, il costo computazionale non ci era accessibile. Invece *decay* indica il tasso di decadimento. I risultati affermano che, dopo aver fatto diverse simulazioni, i valori che ci danno l’accuratezza più alta sono *size*=15 e *decay*=0.25. Ciò significa che nell’hidden layer ci saranno 15 neuroni con un tasso di decadimento di 0.25.

L’accuracy osservata sul training set è di 0.767. Abbiamo in seguito previsto i dati sul test set e abbiamo creato una confusion matrix confrontando queste previsioni con le etichette reali delle istanze del test set. In questo caso l’accuratezza diminuisce a 0.75, la sensitività è di 0.580 e la specificità è 0.839.

Il seguente grafico riporta la variazione dell’accuratezza rispetto a vari valori del tasso di decadimento: per valori di *decay* compresi tra 0 e 0.2 si ha un aumento significativo dell’accuracy che poi sembra stabilizzarsi e presentare un picco in corrispondenza di *decay*=0.25.

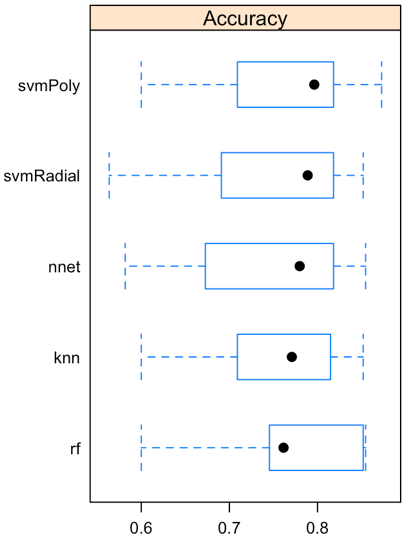
****

Un’osservazione importante che si può trarre da questi esiti è che i metodi del KNN e delle Reti Neurali hanno una sensitività maggiore rispetto agli altri algoritmi. È possibile dunque affermare che con procedimenti si può classificare e prevedere meglio le istanze positive, seppure con una probabilità di errore che resta importante.

**CONCLUSIONE**

Per concludere possiamo affermare che per trattare il dataset Diabetes sono più efficaci i metodi di classificazione rispetto ai metodi di clustering. In particolare, la SVM con kernel “polinomiale” o con kernel “radiale” sono le procedure che portano i risultati migliori a livello di performance sul test set: in entrambi i casi abbiamo una misura di accuratezza che si aggira intorno all’80%. Il valore che più ci ha sorpreso è stata la specificità che raggiunge addirittura un valore del 93%, mentre la sensitività è intorno al 55%. Con questo possiamo affermare che il Machine Learning applicato a questo insieme di dati permette di prevedere con grande efficacia i “non diabetici” mentre ha più difficolta nella previsione dei “diabetici”.

È possibile, inoltre, confrontare i diversi livelli di accuratezza osservati sul training set tramite i diversi algoritmi: seppur i valori appaiono molto simili tra loro sono comunque presenti minime differenze di pochi millesimi di unità. Per una maggiore coerenza tra i risultati, abbiamo impostato come metodo di valutazione la cross-validation per tutti gli algoritmi. La raffigurazione seguente mostra i diversi livelli di accuratezza misurate sul training set: dalla sua osservazione si può notare che le Support Vector Machine con kernel “polinomiale” o con kernel “radiale” sono i metodi che restituiscono un livello di accuracy maggiore anche per quanto riguarda il training set.

****

**BIBLIOGRAFIA E SITOGRAFIA**

* <https://elearning.unimib.it/course/view.php?id=38076>
* <https://it.wikipedia.org/wiki/Algoritmo_EM>
* <https://en.wikipedia.org/wiki/Ward%27s_method>
* <https://arxiv.org/pdf/1111.6285.pdf>
* <https://lexjansen.com/nesug/nesug10/hl/hl07.pdf>
* <http://topepo.github.io/caret/train-models-by-tag.html>
* <http://topepo.github.io/caret/available-models.html>
* <http://topepo.github.io/caret/using-your-own-model-in-train.html>
* <https://www.rdocumentation.org/packages/stats/versions/3.6.2/topics/rect.hclust>
* <https://it.wikipedia.org/wiki/K-nearest_neighbors>
* <https://cran.r-project.org/web/packages/caret/vignettes/caret.html>
* <https://www.rebeccabarter.com/blog/2017-11-17-caret_tutorial/>