Supervisioned learning: scelta e valutazione di un modello applicato ad un classico problema di classificazione lineare

Fabio Proietti

Università di Bologna,
Dipartimento di Scienze dell'Informazione
fabio.proietti@studio.unibo.it

Abstract. La primaria difficoltà nell'utilizzo delle tecniche di machine learning è la totale incertezza su come esse agiscono rispetto al problema da risolvere. E' per questo necessario capire a priori che tipo di task si sta affrontando e quali approcci potrebbero essere utili alla causa. Lo scopo del progetto è quindi modellare una strategia che sia in grado di massimizzare sia l'apprendimento delle caratteristiche utilizzate per il learning, che la classificazione di nuovi elementi sulla base di quello che il modello ha appreso. L'elaborato dopo una breve introduzione, presenterà le caratteristiche del dataset utilizzato e la selezione di strategie per la modellazione. Attraverso i test effettuati saranno infine valutati i risultati ottenuti.

1 Introduzione

Il mondo del machine learning è suddiviso in tre parti [1]: esistono tecniche con un apprendimento di tipo supervisionato, non supervisionato o con un apprendimento definito per rinforzo.

Il *supervisioned learning* ha l'obiettivo di apprendere un modello a partire da un dataset composto da elementi categorizzati in classi. Un esempio tipico è la modellazione di un approccio che è in grado di capire se una mail appena ricevuta contiene o meno spam. La strategia quindi si basa su dataset in cui l'insieme delle caratteristiche di ogni mail sono classificate in due insiemi ben distinti.
Un altro task molto comune per un apprendimento con supervisione è la cosiddetta *regression analisys*; essa si basa su un certo numero di variabili utilizzate per il training dette *predictor* e su una variabile continua come risultato. Lo scopo della regressione è quindi capire la relazione che intercorre tra quelle variabili. In questo senso un tipico esempio è cercare di apprendere quanto l'ammontare dello studio per un esame da parte di uno studente ha influenzato il voto finale, così che poi si è in grado di predire l'esito dell'esame di un futuro studente in base al suo carico di studio.

Il tipo di apprendimento non supervisionato invece basa la creazione di un modello su dataset che non classificano le features che lo costituiscono. Nel caso

della classificazione quindi avremo un training set che non è in grado di distinguere se una mail è di spam oppure no, o nel caso della regressione avremo noto solo l'approccio di studio di uno studente e non il risultato dell'esame. Il workflow di questi algoritmi ha come scopo primario quello di cercare delle similitudini sui dati in possesso e categorizzarli, mentre la fase di testing è fondamentalmente la stessa rispetto al learning supervisionato.

Il reinforcement learning [2] mira alla generazione di un agente che apprende in base alle interazioni che esso avrà con l'ambiente. Nel momento in cui l'agente cambia di stato (ossia effettua una azione modificando l'ambiente) esso attende una risposta a cui può essere associata una ricompensa. I nuovi dati appresi dall'agente andranno perciò a modificare i comportamenti futuri dello stesso; un tipico esempio di applicazione può essere istruire un agente in modo da esplorare uno spazio (che può contenere ostacoli, altri agenti, ecc) così che sia in grado di raggiungere un nodo obiettivo con il minor numero di "mosse" (ossia il numero di cambiamento di stato) possibile.

Il problema studiato in questo progetto rientra nelle caratteristiche di un apprendimento supervisionato ed ha lo scopo di classificare i cittadini della città di Chicago in due range di reddito: sopra o sotto i 50 mila dollari annuali. Gli algoritmi di apprendimento utilizzati sono stati la Regressione Logistica, le Support Vector Machine con l'utilizzo di un kernel lineare e la tecnica dell'albero decisionale. Dopo la definizione del dataset utilizzato, le prossime sezioni mostreranno il workflow di modellazione di varie strategie che comprendono anche la combinazione di tutti i precedenti algoritmi (Ensemble Classification).

2 Analisi preliminare del task

2.1 Definizione del dataset

Il dataset utilizzato per questo scopo proviene dalla repository Uci machine learning, ed è gratuitamente fruibile da questo link. Il dataset descrive il risultato di un censimento effettuato nel 1994 ed ogni record rispetta le caratteristiche di un utente classificandolo in base al reddito annuo (>=50k $ o <50k $); i restanti attributi sono in totale 15 e sono sia di tipo numerico che testuale, con un insieme di dati continuo (età, ore lavorative per settimana, ecc.) o discreto (paese di origine, attestato di studi, ecc.). Il numero di entry è di circa 48 mila e al suo interno vi sono features sconosciute.

2.2 Model Flow

Il dataset scelto prevede la classificazione degli utenti suddividendoli in due categorie basate sul reddito annuo. Questo è quindi un classico problema di classificazione basata su apprendimento supervisionato (conosciamo le classi a priori). Date queste condizioni il processo di creazione di un modello in grado di risolvere il task si basa sui seguenti step (Fig.1):

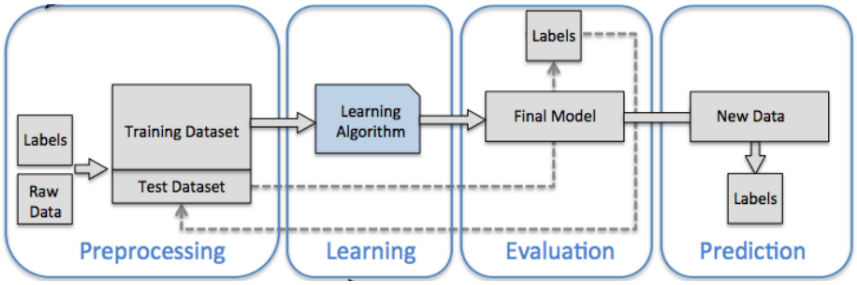


Fig. 1: Steps per la creazione di un modello che ha lo scopo di classficare nuovi dati.

- Preprocessing: Questa è una fase molto importante che riguarda l'uso di tecniche in grado di manipolare i dati a disposizione prima dell'applicazione dell'algoritmo di learning. E' necessario quindi suddividere il dataset in due sotto insiemi (training e test set) e quindi isolare l'attributo di classificazione (labels) dai restanti dati (raw data). Molto spesso queste suddivisioni però sono precedute dall'applicazione di tecniche che sono in grado di eliminare features ridondanti (feature selection); può anche essere possibile ricavare le cosiddette "meta-informazioni" attraverso tecniche di feature extraction o più semplicemente normalizzare e ridurre la dimensione del dataset stesso.
- Learning: A questo punto, dopo aver suddiviso il dataset, la fase di learning consiste nell'applicare un algoritmo di classificazione al nostro training set. Come detto non esiste un grado di superiorità di un algoritmo rispetto ad un altro, per questo è necessario effettuare una comparazione (o combinazione) tra quelli che sembrano più adatti allo scopo.
- Evaluation: Il metodo di valutazione consiste attraverso il test set creato nel primo di step, di valutare le prestazioni del modello utilizzando e cosiddette metriche di valutazione.La metrica che è stata usata in questo progetto è l'accuratezza e consiste nel calcolo della proporzione del numero delle istanze classificate correttamente.
- Prediction: Lo step finale si raggiunge quando il modello creato soddisfa le aspettative ed è pronto per essere utilizzato. E' infatti in questo momento che deve stabilire la classe per ogni nuovo record dato come input.

3 progettazione e valutazione del modello

3.1 Preprocessing

Per costruire un buon training set è necessario applicare diverse tecniche di learning [1][3] per far sì che il nostro modello di training non sia né troppo generale e nemmeno tanto specifico da non poter essere applicato ad un problema reale.

Imputing In ogni dataset la prima accortezza da avere è verificare se esso contiene o meno valori sconosciuti. Questa operazione si chiama *imputing* e consiste nell'effettuare delle scelte sul come gestire i cosiddetti "*missing values*". Lo stato dell'arte prevede varie vie di risoluzione: si possono eliminare tutti i record che contengono un *NaN value*, oppure si effettua la media dei valori in cui il valore conosciuto non è presente (un valore NaN viene considerato 0), o più semplicemente si inserisce il risultato calcolando il valore più ricorrente rispetto sempre l'attributo mancante.

Label Encoding A questo punto è importante determinare di che tipo sono le classi in cui si vogliono categorizzare i dati restanti. In generale la classificazione può essere binaria o multi classe ma soprattutto può essere rappresentata su scala numerica oppure in modo testuale. In questo ultimo caso è quindi necessario applicare la tecnica chiamata "*label encoding*" che semplicemente categorizza in scala numerica le classi riportate nel dataset. E' molto comune però avere a che fare con un insieme di dati che presentino più di un attributo che abbia valori discreti, per esempio in questo dataset sono attributi discreti lo stato di provenienza o lo stato relazionale, ed è quindi possibile applicare il *label encoding* anche a quelle features che non sono viste come lo scopo della predizione.

Features map Nel caso di features di tipo discreto è possibile anche applicare tecniche che oltre alla semplice traduzione numerale, siano in grado darci informazioni maggiori rispetto a quel specifico attributo. Ad esempio dato che un indumento può essere di varia grandezza, una feature nominata "taglia" oltre che a determinare la classe di appartenenza può anche definire che la taglia "M" è maggiore di della "S" ma minore della "L"; utilizzando un dizionario che adotti questa rappresentazione avremo che il valore assegnato alla lettera "M" sarà sicuramente maggiore di quello utilizzato per "S" ma minore di "L".

Feature Scaling Quando si decide di utilizzare certi estimatori, scalare il valore delle features può essere importante e tecniche che permettono di raggiungere questo risultato possono essere la *normalizzazione* o la *standardizzazione*. Per quanto riguarda la normalizzazione molto spesso si decide di scalare il valore delle features un intorno compreso tra $[0, 1]$ calcolando il nuovo $x^{(i)}_{norm}$ come:

$$x^{(i)}_{norm} = \frac{x^{(i)} - x_{min}}{x_{max} - x_{min}} \tag{1}$$

dove x_{min} e x_{max} sono i valori estremi dell'attributo e $x^{(i)}$ è l'*i-esimo* valore da scalare. Usando la standardizzazione invece si centrano i valori delle nuove features in modo da avere per ogni colonna media 0 e deviazione standard 1. La procedure può quindi essere espressa come:

$$x^{(i)}_{std} = \frac{x^{(i)} - \mu_x}{\sigma_x} \tag{2}$$

dove μ_x e σ_x sono rispettivamente la media e la deviazione standard di un particolare attributo.

Feature Selection Questo tipo di approccio viene utilizzato nel caso in cui un modello tende ad avere problemi di *overfitting* (o *underfitting*). Infatti nel caso in cui le performance del modello siano nettamente migliori nel training set rispetto che nel test set, avremo a che fare con una alta varianza e un basso bias (= overfitting). Una delle tecniche di feature selection utilizzata maggiormente per risolvere il problema è la *regolarizzazione* (che può essere di tipo L1 o L2). Questo approccio non fa altro che penalizzare i pesi delle features che hanno un grande valore. Per definire il range di pesi in cui si voglia che la tecnica adoperi, si utilizza un parametro λ: incrementando questo valore i pesi vengono tutti ristretti in un intorno vicino allo zero così che il nostro modello è meno dipendente a valori che prima assumevano un peso eccessivo. In pratica, il metodo di regolarizzazione L1 prevede che $\lambda||w||$ sia sostituito con $\lambda \sum_{j=1}^{k} |w_j|$, mentre in L2 avremo che $\lambda||w||_2^2$ sia uguale a $\lambda \sum_{j=1} k w_k^2$. Applicando queste formule si avrà che rispetto ad L2, L1 produrrà un vettore con *sparse features* dato che molti pesi saranno nulli. La "sparsità" delle features però potrebbe essere un fattore positivo nel caso in cui si è di fronte ad un dataset che ha una dimensionalità molto alta e quindi con molte features irrilevanti.

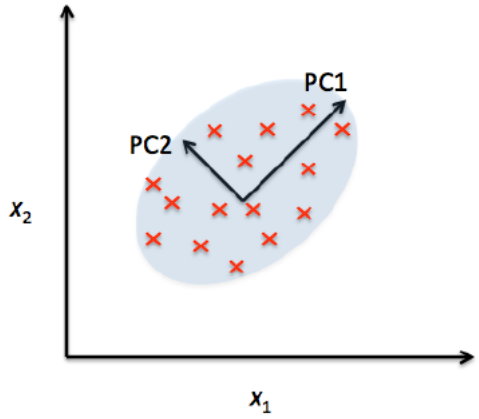


Fig. 2: "direzioni" della massima varianza in un dataset altamente dimensionale

Feature extraction Tecniche per l'estrazione di pattern basati sulle caratteristiche di un dataset sono molto spesso utilizzate per ridurre la dimensionalità dello stesso. Una tra i più importanti approcci prende il nome di *Principal Component Analysis (PCA)*; in poche parole il suo scopo è quello di trovare le "direzioni" della varianza massima di un dataset e proiettarle in un sotto spazio che abbia una dimensionalità al più uguale a quella di partenza (Fig.2). Analiticamente:

$$x = [x_1, x_2, \ldots, x_n], \mathbf{x} \in \Re^d$$
$$\Downarrow \mathbf{xW},\ \mathbf{W} \in \Re^{d \times k}$$
$$z = [z_1, z_2, \ldots, z_n], \mathbf{z}, \in \Re^k$$

in cui x e z sono i due vettori prima e dopo la riduzione della dimensione. Si avrà quindi che $k \ll d$.

Train Test split La suddivisione tra training e test set è infine molto importante perché l'apprendimento del dataset sia massima. Gli approcci possibili in questo caso sono tipicamente due: dividere in modo randomico l'insieme di dati in due sotto insiemi, uno per il train set (tipicamente il 70%) ed uno per il test set (restante 30%).

3.2 Learning

Una volta svolto il pre-processamento dei dati si passa alla fase di learning, la quale consiste nell'allenare gli algoritmi scelti per poi compararli e decretare nella fase di valutazione quale di questi è il più adatto da applicare al modello. Ovviamente dato che si tratta di un task di classificazione binaria con apprendimento supervisionato, il ventaglio di algoritmi si riduce a quelli adatti a problemi come questo. Anche se ciò è solo un sottoinsieme dei task possibili nel mondo del machine learning, fare una panoramica di tutti gli algoritmi di classificazione supervisionata richiederebbe troppo tempo, per cui rimanendo in linea con quanto detto fin'ora, nelle seguenti sezioni saranno mostrati solo gli approcci utilizzati per la costruzione del modello basato sul dataset precedentemente descritto. Più specificatamente gli algoritmi presi in esame saranno: L'Albero decisionale, la regressione logistica e le Support Vector Machine unicamente con kernel lineare.

Decision Tree Classifier Il metodo usato da questo algoritmo è quello di predire le classi dei samples attraverso l'uso di domande estrapolate dall'analisi del training set. Partendo dalla radice si inizia quindi a suddividere il learning set isolando le features che assumono il maggior *information gain*, ossia:

$$IG(D_p, f) = I(D_p) - \sum_{j=1}^{m} \frac{N_j}{N_p} I(D_j) \tag{3}$$

dove D_p corrisponde al parent set, f all'insieme delle features che devono essere suddivise, N_p è il numero totale delle features, I corrisponde ad una misura di impurità e per finire D_j corrisponde al j-esimo nodo figlio. In altre parole l'Information Gain è la differenza tra la misura di impurità del dataset genitore e la somma delle impurità dei nodi figli. La precedente formula (3) assume che la formazione dell'albero decisionale non sia binario. Molte implementazioni

però, per varie ragioni limitano il numero di child sets a 2, assumendo quindi la formazione di un albero binario. In questo caso la formula (3) diventa:

$$IG(D_p, f) = I(D_p) - \frac{N_{left}}{N_p} I(D_{left}) - \frac{N_{right}}{N_p} I(D_{right}) \tag{4}$$

La misura di impurità è un valore utile per ridurre il margine di errore durante la classificazione dei samples; i criteri maggiormente usati a questo scopo sono tre: *Gini index*, *Entropy*, *Classification error*.

L'indice di Gini ha lo scopo di ridurre al minimo la probabilità di errore durante la classificazione e questo è maggiormente possibile se le classi sono perfettamente mixate. Il Gini index quindi è dato da:

$$I_G(t) = \sum_{i=1}^{c} p(i|t)(-p(i|t)) = 1 - \sum_{i=1}^{c} p(i|t)^2 \tag{5}$$

nel caso di un albero binario avremo che l'impurità di Gini sarà:

$$I_G(t) = 1 - \sum_{i=1}^{c} 0.5^2 = 0.5$$

Anche utilizzando il criterio dell'entropia si tendono ad avere risultati molto simili a quelli dati dall'indice di Gini.

$$I_H(t) = -\sum_{i=1}^{c} p(i|t) \log_2(p(i|t)) \tag{6}$$

Come nel Gini index $p(i|t)$ corrisponde alla probabilità che i samples appartengano alla classe c dato un certo nodo t. Per questo l'entropia è 0 se tutti i nodi fanno parte della stessa classe, mentre è massima se sono uniformemente distribuiti.

La classificazione dell'errore infine è definita come:

$$I_E(t) = 1 - \max\{p(i|t)\} \tag{7}$$

Questo criterio è molto utile per sfoltire il nostro albero decisionale, ma poco efficace se lo spazio dei dati (e quindi dell'albero) tende a crescere. Questo perché aumentando il numero di elementi, le probabilità che essi appartengono a quella determinata classe è destinata a cambiare.

Logistic Regression La regressione logistica è un approccio molto usato perché semplice da implementare e molto ben performante nel caso della risoluzione di problemi lineari. Per capirne il funzionamento è utile partire dal concetto di *odds ratio*, che rappresenta la probabilità che un evento positivo possa accadere, e può essere scritto come $\frac{p}{1-p}$. In altre parole p è la probabilità che un certo evento appartenga ad una determinata classe. Con $logit(p)$ intendiamo invece il logaritmo dell'odds ratio, quindi assumiamo che

$$logit(p) = \log \frac{p}{1-p} \tag{8}$$

ossia vengono presi come input valori compresi nell'intervallo $[0, 1]$ così da essere trasformati in valori reali ed usati per esprimere una relazione lineare tra i valori delle features e quelli del *log-odds* (8). Numericamente parlando quindi abbiamo:

$$logit(p(y = 1|x)) = w_0x_0 + x_1x_1 + \cdots + w_kx_k = \sum_{j=0}^{k} w_jx_j = w^Tx \tag{9}$$

con $p(y = 1|x)$ la probabilità condizionata in cui un certo elemento appartenga alla classe y data la feature x con peso w. Quello che noi a questo punto vogliamo ottenere, è predire se un dato evento appartiene o meno ad una certa classe e per questo è necessario calcolare l'inverso del logaritmo precedentemente definito. Questa funzione è detta *logistica* o *sigmoidale* e può essere espressa come segue:

$$\phi(z) = \frac{1}{1 - e^{-z}}, z = w^Tx \tag{10}$$

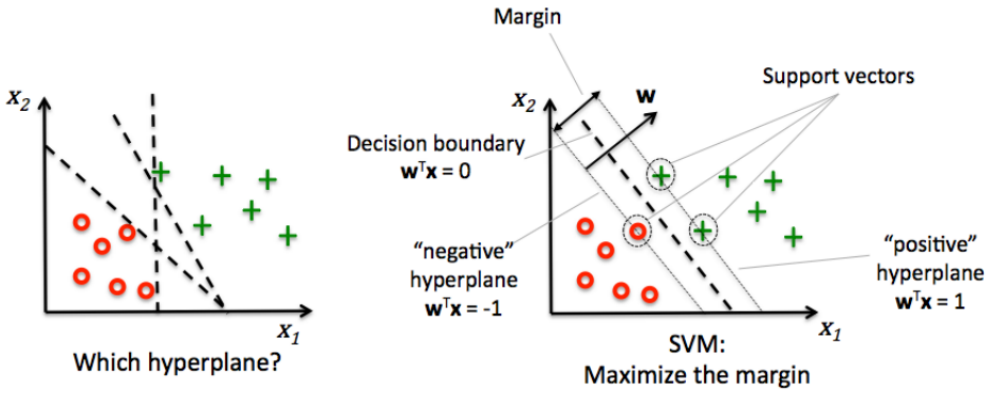


Fig. 3: Support vector Classification

SVM L'obiettivo di classificatore è quello di massimizzare il margine di confine di ogni classe del task. Come mostrato in Fig.3 il margine non è altro che la linea di confine posta in prossimità dei samples, i quali prendono il nome di support vectors. I support vectors danno origine alla creazione di iperpiani che sono paralleli rispetto al confine decisionale delle classi e nel caso in cui il margine sia vicino allo zero avremmo un modello incline all'overfitting, altrimenti si potrebbe avere il problema di un basso errore di generalizzazione. Dato che gli iperpiani sono paralleli tra loro possono essere espressi in questo modo:

$$\begin{aligned} w_0 + w^Tx_{pos} &= 1 \\ w_0 + w^Tx_{neg} &= -1 \end{aligned} \tag{11}$$

dove la prima espressione rappresenta l'iperpiano positivo, mentre la seconda quello negativo. Come detto le SVM mirano a massimizzare il margine, perciò è necessario prima conoscere la distanza tra le classi che è possibile calcolare come segue:

$$w^T(x_{pos} - x_{neg}) = 2 \tag{12}$$

$$\frac{w^T(x_{pos} - x_{neg})}{||w||} = \frac{2}{||w||}, ||w|| = \sqrt{\sum_{j=1}^{m} w_j^2} \tag{13}$$

sottraendo le due precedenti equazioni (12) e dividendo il risultato per la lunghezza del vettore w è possibile utilizzare il primo membro dell'equazione (13) per poter calcolare il margine. Calcolata la distanza si passa alla sua massimizzazione ($\frac{2}{||w||}$), ovviamente assumendo come vincolo che le nuove features siano classificate sempre nel modo corretto; riprendendo le equazioni degli iperpiani precedenti (11) si può assumere:

$$\begin{aligned} w_0 + w^T x^i \geq 1, \forall y^{(i)} = 1 - \epsilon^{(i)} \\ w_0 + w^T x^i < -1, \forall y^{(i)} = 1 + \epsilon^{(i)} \end{aligned} \tag{14}$$

tutti i samples negativi dovrebbero cadere nella parte negativa e tutti i samples positivi in quella positiva; ciò può essere descritto più sinteticamente come:

$$\frac{1}{2}||w||^2 + C\left(\sum_i \epsilon^{(i)}\right) \tag{15}$$

Usando la variabile C si può controllare la penalità per ogni mancata classificazione, ciò vuol dire che C corrisponde alla grandezza del margine tra gli iperpiani ed è perciò motivo del tradeoff bias-varianza che è in grado di eliminare problemi di over/underfitting; questo concetto viene detto regolarizzazione.

3.3 Evaluation

Dopo aver processato il dataset, e dopo aver scelto gli algoritmi di learning è necessario verificare qual è il modello che si adatta meglio al task. Questa fase è molto importante poiché permette di capire quante e quali modifiche è necessario apportare affinché si avvisi un aumento delle prestazioni. Spesso però questo incremento è imputabile anche (ma non solo) alla suddivisione del dataset stesso; precedentemente si è parlato della tecnica *Train test split* che, definita una percentuale di grandezza di uno dei due sottoinsiemi, la funzione ha lo scopo di dividere il dataset randomicamente in train e test set. Il primo verrà utilizzato per la fase di learning, mentre il secondo per la fase di valutazione. Questo tipo di suddivisione però non ha alcun tipo di fondamento logico e potrebbe non essere la migliore soluzione per far apprendere al modello i pattern presenti insiti nel

dataset. E' infatti possibili eliminare eventuali problemi di overfitting o underfitting semplicemente variando le dimensioni della suddivisione o aggiungendo nuove features al dataset stesso.

Per aumentare l'efficienza di questa fase esistono quindi dei metodi che permettono di ottimizzare la ripartizione del dataset di partenza, ma anche per combinare tutte le tecniche mostrate nelle sezioni modificandone anche gli attributi di funzione.

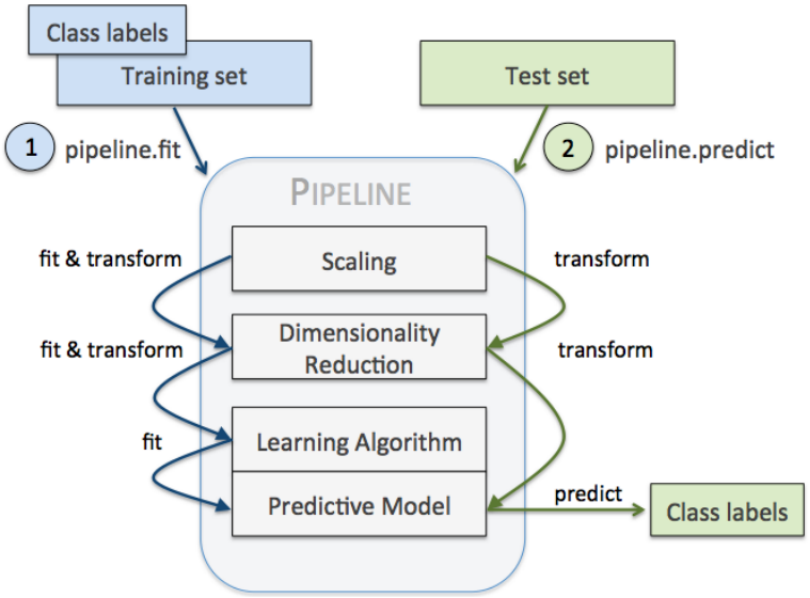


Fig. 4: Esempio di tecniche utilizzate in una pipeline

Pipeline La pipeline è uno strumento molto utile per combinare tecniche di trasformazione dei dati con gli estimatori. Essa infatti prende come oggetto una lista di tuple corrispondenti alle funzioni scelte; per prima cosa effettua la trasformazione dei dati del train set e li fa apprendere dall'estimatore, quindi trasforma i dati del test set e ne predice le classi (Fig.4).

GridSearch In ogni estimatore è possibile manipolare i parametri dei suoi attributi in base alle necessità. Ad esempio è possibile scegliere il tipo di regolarizzazione da usare nella regressione logistica o il modello di impurità per quanto riguarda il Decision Tree Classifier. Per questo in aggiunta alla pipeline è possibile utilizzare questa strategia che, definite delle liste di alternative come input, è in grado di mescolare i valori tra loro delineando la combinazione che riporta il risultato migliore rispetto la metrica utilizzata.

Fig. 5: Esempio della tecnica K-Fold Cross-validation

Cross-validation In aggiunta la Grid Search utilizza una tecnica chiamata cross-validation che, secondo certe regole, è in grado di suddividere il dataset così da incrementare il learning ed evitare possibili problemi di fitting causati da una cattiva suddivione del set di informazioni. Tra le tecniche la più utilizzata è la *K-fold cross validation* (Fig. 5), la quale consiste nell'suddivere l'intero training set in k parti, di cui $k-1$ sono utilizzate per il training set e la restante come test set. La particolarità di questa tecnica prevede che vengano fatte al più k operazioni in modo tale che il testing set sia assunto sempre da una diversa sezione delle k parti definite. Ad ogni iterazione quindi viene svolta la valutazione del modello secondo la suddivisione effettuata, per cui la stima finale sarà la media aritmetica delle stime calcolate ad ogni iterazione:

$$E = \frac{1}{10} \sum_{i=1}^{k} E_i \tag{16}$$

Learning Curves Oltre alle metriche di base, come si vedrà poi nei test, per valutare un modello può tornare molto utile l'applicazione di tecniche usate per il debugging degli estimatori come le learning curves. Esse sono comode per verificare graficamente se un determinato modello soffre di overfitting, e quindi alta varianza, o underfitting (alto bias). Quindi in aggiunta alla cross-validation e al GridSearch, queste ci permettono di capire, calcolando anche la deviazione standard di una stima del modello, come agire per evitare i due problemi sopra citati. Molto spesso infatti le soluzioni adottate in questi casi sono l'incremento dei samples per il tranining set, una nuova distribuzione dei pesi delle features oppure l'aumento dei parametri di regolarizzazione.

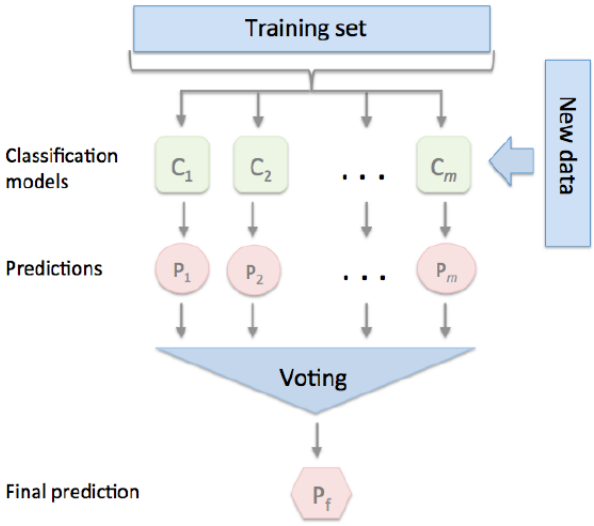


Fig. 6: Ensemble modeling con sistema di votazione

4 Ensemble modeling

Questo tipo di modellazione ha come approccio fondamentale quello di mettere insieme più modelli così da costruire una serie di classificatori che possono essere in grado di raggiungere risultati migliori rispetto a modelli singoli. Le tecniche utilizzate da questa strategia sono generalmente tre: definire il modello migliore attraverso uno schema che prevede un sistema di voto maggioritario; combinare randomicamente i modelli così da ridurre overfitting e underfitting oppure utilizzare più modelli "deboli" che sono in grado di incrementare le stime di valutazione apprendendo le debolezze degli altri. Come si nota nella figura 6, nel sistema di predizione a voto maggioritario, utilizzando il training set si effettua il learning degli n classificatori prima e la valutazione poi. per stabilire la classe di un determinato elemento, semplicemente si uniscono tutte le predizioni e si sceglie la classe che appare più frequentemente.

5 Tests

Dopo aver introdotto la teoria che regola la modellazione di un estimatore in questa sezione si passa ad applicare praticamente questi concetti. Gli esperimenti svolti sono tutti basati sul dataset precedentemente descritto.

5.1 Tools utilizzati

L'intero progetto è stato sviluppato in Python utilizzando le seguenti librerie:

– *Scikit-learn*[4] Per quanto riguarda l'applicazione delle tecniche utili alla costruzione del modello (Preprocessing, estimatori, ensemble modeling, ecc.);
– *Pandas*[5] per la gestione del dataset (unknown values, unione di subsets, ecc.);
– *Numpy*[6] come supporto a Pandas (nella gestione del tipo dei dati, nel calcolo di medie, deviazioni standard, ecc.);
– *Pyplot*[7], un modulo di *MatPlotLib* per la creazione e gestione di grafici.

5.2 Test 1: Random Model

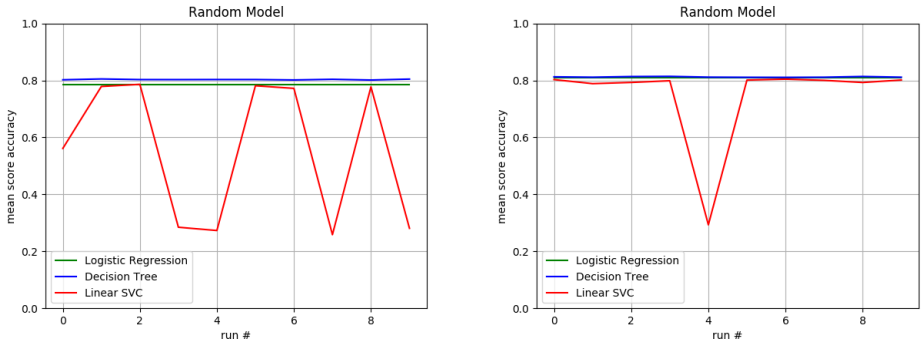
in questo primo test si è voluto far notare come le strategie di modellazione di un task siano importanti. Le uniche tecniche di preprocessing applicate sono la *label encoding* e la gestione dei valori sconosciuti; nella prima parte del test essi sono stati rimossi, mentre successivamente sono stati sostituiti con il valore più presente nella colonna corrispondente al dato mancante. Come si nota nella tabella 1 gli estimatori utilizzati sono stati la Logistic Regression, il Decision Tree e la linearSVC che altro non è che una SVM adatta a risolvere problemi lineari. Gli Hyperparameters dei classificatori adottati sono i valori di default assegnati dalla libreria. Il dataset è stato suddiviso con la tecnica Train Test Split, assegnando il 70% dei records al train set e il 30% al test set. I risultati si basano sulla media di 10 runs, mentre ome metrica è stata utilizzata l'accuratezza di predizione con l'aggiunta della deviazione standard.

Classifier	Train Accuracy (Mean)	Eval Accuracy (Mean)	removal unknown
Logistic Regression	0.791 +/- 0.000	0.786 +/- 0.000	yes
Decision Tree Classifier	1.000 +/- 0.000	0.803 +/- 0.001	yes
LinearSVC	0.555 +/- 0.247	0.555 +/- 0.238	yes
Logistic Regression	0.805 +/- 0.000	0.809 +/- 0.000	no
Decision Tree Classifier	1.000 +/- 0.000	0.812 +/- 0.001	no
LinearSVC	0.743 +/- 0.147	0.748 +/- 0.152	no

Table 1: Test 1

I grafici risultanti al test (Fig.7) riportano la valutazione del modello ad ogni run. E' subito possibile notare come sia la regressione logistica che il l'albero decisionale non subiscono la differenza rispetto al modo di gestione dei valori sconosciuti; cosa non vera per quanto riguarda LinearSVC, la quale, oltre ad avere nel primo caso valori molto altalenanti, tende ad avere prestazioni nel complesso più basse rispetto agli altri due. Questo ovviamente può essere dovuto anche ad altri fattori come la suddivisione del dataset o ai parametri assegnati.

E' infine importante notare come in questo caso la presenza di overfitting nel Decision Tree è possibile notarla solamente comparando la media degli score rispetto al training e al test set nella tabella 1.



(a) removal unknown: yes (b) removal unknown: no

Fig. 7: Random Model

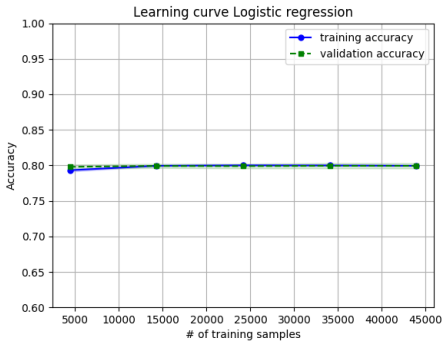
5.3 Test 2: Pipe Model

Il secondo test svolto riguarda invece la creazione di un modello facendo molta attenzione anche alla fase di pre-processamento del dataset. Come si nota nella tabella 2, per quanto riguarda il test con gli estimatori Logistic Regression e Linear SVC i dati sono stati manipolati applicando la tecnica di standardizzazione e riduzione della dimensionalità del dataset (PCA). Per il Decision Tree si è invece cercato di risolvere il problema dell'alta varianza riscontrato nel primo test, applicando una regolarizzazione di tipo "L2". Ovviamente, come nel primo test sono state applicate le tecniche Label Encoding e imputing rispettivamente per codificare gli attributi con spazio discreto e per gestire il problema di valori mancanti. La valutazione dei modelli è stata possibile grazie alla applicazione della tecnica di debugging "Learning Curve" insieme alla K-Fold cross_validation, con K = 10. I parametri degli estimatori non sono stati modificati.

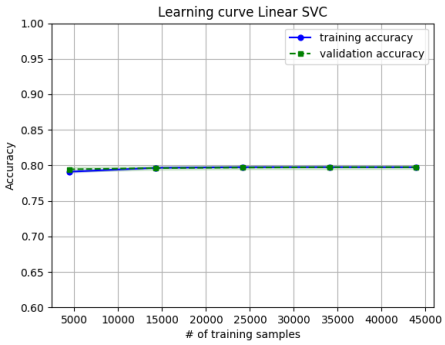
Classifier	Train Accuracy	Test Accuracy	preprocessing	removal
Logistic Regression	0.789 +/- 0.001	0.790 +/- 0.001	StandardScaler, PCA	yes
Decision Tree Classifier	0.836 +/- 0.004	0.806 +/- 0.006	regularization 'l2'	yes
LinearSVC	0.787 +/- 0.000	0.787 +/- 0.000	StandardScaler, PCA	yes
Logistic Regression	0.798 +/- 0.001	0.799 +/- 0.000	StandardScaler, PCA	no
Decision Tree Classifier	0.846 +/- 0.002	0.798 +/- 0.006	regularization 'l2'	no
LinearSVC	0.796 +/- 0.000	0.796 +/- 0.000	StandardScaler, PCA	no

Table 2: Pipe Model

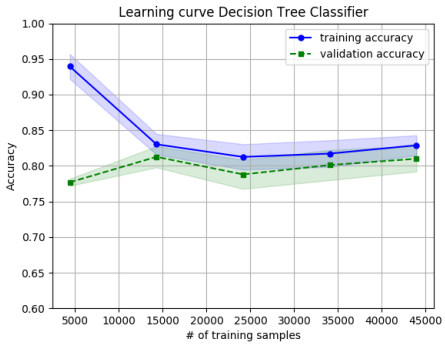
Comparando i risultati raggiunti (Fig.8), in questo caso rispetto al test precedente possiamo effettivamente notare che ponendo principalmente attenzione alla



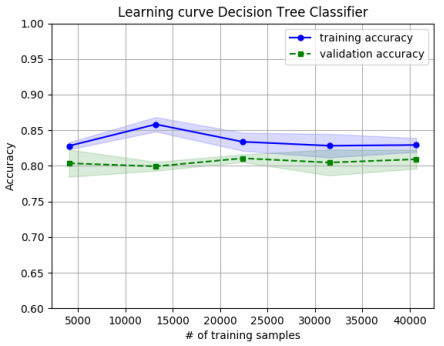
(a) removal: no



(b) removal: no



(c) removal: no



(d) removal: yes

Fig. 8: Pipe Model

fase di preprocessing, è possibile gestire casi di under/overfitting come nel caso dell'estimatore Decision Tree Classifier (Fig.8 (c)(d)).

5.4 Test 3: Grid Search

Fino ad ora la modellazione ha posto l'attenzione soprattutto sulla fase di processamento e valutazione mantenendo invariati i parametri degli estimatori. In questo caso utilizzando la GridSearch cross validation è possibile effettuare una valutazione più profonda dell'intero modello modificando anche quelli che sono i parametri degli estimatori.

Classifier	Accuracy (cross_val)	gridSearch best Params	removal
Logistic Regression	0.790 +/- 0.003	{'lr___solver': 'sag', 'lr___C': 1.0}	yes
Decision Tree Classifier	0.815 +/- 0.004	{'dTree___splitter': 'best', 'dTree___max_depth': 5, 'dTree___criterion': 'gini'}	yes
LinearSVC	0.788 +/- 0.002	{'lsvc___C': 1.0, 'lsvc___penalty': 'l2'}	yes
Logistic Regression	0.799 +/- 0.003	{'lr___solver': 'liblinear', 'lr___C': 0.1}	no
Decision Tree Classifier	0.807 +/- 0.017	{'dTree___splitter': 'best', 'dTree___max_depth': 5, 'dTree___criterion': 'entropy'}	no
LinearSVC	0.797 +/- 0.002	{'lsvc___C': 0.1, 'lsvc___penalty': 'l2'}	no

Table 3: Grid Seatch Cross Validation

Come detto, questa tecnica a forza bruta prende in input una serie di possibili alternative per ogni estimatore, le combina e ne calcola la stima utilizzando la k-fold cross validation. In questo caso per valutare l'accuratezza finale del modello, per la fase di learning ed evaluation è stata utilizzata una tecnica chiamata 5x2 Cross Validation, che altro non è che applicare la cross validation nel training set creato nella gridSearchCV. In questo modo le stime finali pervenute oltre che essere più precise permettono al modello di avere una fase di learning molto più forte. I risultati mostrati nella tabella 3 mostrano la media delle stime per il modello di quello specifico estimatore e la migliore combinazione dei parametri che ne è derivata. La fase di preprocessing invece è la medesima di quella del test precedente.

5.5 Test 4: Ensemble Modeling

Per ultimo si è scelto di combinare i precedenti tre modelli attraverso la tecnica della classificazione in base al voto. Quindi una volta che ogni estimatore ha pre-

detto la sua categoria per un determinato dato, esso viene assegnato alla classe che compare più volte tra quelle stimate. In questo caso è possibile utilizzare questa metrica perché sono presenti 2 classi e 3 estimatori, per cui è impossibile che si formino situazione di parità.

Classifier	Accuracy (crossVal)	removal
Voting Classifier	0.792 +/- 0.003	yes
Voting Classifier	0.801 +/- 0.003	no

Table 4: Ensemble Modeling: Mayority Vote

Conclusioni

Con questo elaborato si è voluto affrontare ed approfondire quella branchia del machine learning che ha come scopo di predire la classe di nuovi samples basandosi sull'apprendimento di dati pre-esistenti. Gli estimatori appartenenti al supervisioned learning sono quindi in grado di estrapolare pattern da dataset precedentemente generati. Come si è visto però questa fase di apprendimento affinché sia efficace necessita di uno studio approfondito del problema: è quindi prima di tutto importante capire che tipo di estimatore potrebbe raggiungere le performance maggiori, ma anche progettare in maniera consona una fase di preprocessing che permetta una visuale quanto più chiara e generale del dataset stesso. Per questi motivi la vera difficoltà nella costruzione di un modello di predizione sta nel determinare che tipo di operazioni sono necessarie.

Nei test precedentemente effettuati è emerso proprio questo: nel primo caso applicando gli estimatori dopo un preprocessing limitato sono sorti problemi che non permettevano di costruire un modello affidale. Il secondo e il terzo test effettuato invece ha dato la prova di come il porre l'attenzione sia sul dataset che sulle proprietà degli estimatori hanno fatto sì che i modelli risultanti siano molto più solidi rispetto al primo test. L'ultimo esperimento invece riguardava l'utilizzo di una tecnica che basava la sua stima sulla combinazione di più modelli: in questo caso, anche se le performance generali sono molto buone, non si notano significativi aumenti di prestazioni rispetto ai due precedenti test.

Gli sviluppi successivi di questo elaborato potrebbero quindi essere quelli di porre l'attenzione sullo studio e sulla modellazione di strategie che vedono l'utilizzo di estimatori diversi rispetto a quelli utilizzati, oltre che all'applicazione di diverse tecniche per la combinazione di modelli per quanto riguarda l'ensemble modeling.

References

1. Complementi di Linguaggi Di programmazione - Slides del corso
2. Kaelbling, Leslie Pack, Michael L. Littman, and Andrew W. Moore. "Reinforcement learning: A survey." Journal of artificial intelligence research 4 (1996): 237-285.
3. Python Machine Learning - Sebastian Raschka
4. Scikit-Learn Documentation
5. Pandas documentation
6. Numpy documentation
7. MatPlotlib documentation