Contents

1 Introduzione

Il linguaggio di programmazione R ha come suo punto di forza il grande numero di librerie, più di 20k, a sua disposizione tramite Cran [1]. La florida disponibilità di librerie ha come svantaggio la mancanza di una standardizzazione della sintassi e delle funzioni offerte dai pacchetti, tale per cui anche pacchetti che offrono

funzioni complementari, finiscono per avere grammatiche e comandi diversi, e richiedere al programmatore un maggiore dispendio di tempo.

In particolare nei modelli di machine learning, dato che la pipeline di analisi dei dati, dall'esplorazione iniziale, al preprocessing, alla selezione e valutazione di un modello, e all'integrazione di una vasta scelta di modelli sia di regressione, che di classificazione, con tecniche di resampling e parallelizzazione, la necessità di un'unico pacchetto che gestisca queste funzioni è evidente.

In questo lavoro introduciamo Caret (Classification And REgression Training) [2], un pacchetto che mira a coprire tutti gli aspetti della pipeline di produzione di modelli di machine learning, e che sta venendo sfidato solo recentemente da Tidy-Models [3]. Le funzionalità di Caret vengono illustrate in questo lavoro tramite i dataset 'Titanic' [4] e 'House Prices' [5], rispettivamente di regressione e di classificazione.

Nella sezione 2 esaminiamo le funzioni di visualizzazione, e analisi esplorativa dei dati offerte da Caret e nella sezione 3 illustriamo le principali funzioni di preprocessing, dall'individuazione dei predittori a varianza zero, alle trasformazioni dei dati come centramento o PCA [6], e procediamo con il partizionamento di un dataset in train e testset.

Nella sezione 4 discutiamo le funzionalità principali di Caret, riguardanti l'addestramento dei modelli e il tuning dei parametri attraverso un'unica funzione standardizzata per tutti i modelli, oltre che a discutere la parallelizzazione delle tecniche di resampling.

Infine, nelle sezioni 5 e 6, esaminiamo rispettivamente le tecniche di selezione delle features offerte da Caret, e le metriche per il confronto di modelli diversi.

2 Visualizzazione dei dati

In questa sezione descriviamo i due dataset, 'House Prices' [5] e 'Titanic' [4], rispettivamente di regressione e di classificazione, analizzati in questo lavoro tramite le funzionalità di Caret.

2.1 Pulizia dei dati nei dataset House Prices e Titanic

Il dataset 'House Prices' contiene 80 variabili esplicative, mentre il dataset 'Titanic' ne contiene 11, sia categoriche che numeriche per entrambi.

Entrambi i dataset necessitano di pulizia, conversione di variabili numeriche a fattori e rimozione di valori nulli, per la quale Caret non offre alcuna funzionalità,

e che è stata fatta utilizzando le funzioni base di *R*, in particolare, le variabili contenenti stringhe non convertibili in fattori sono state rimosse, i valori nulli sono stati sostituiti con la media (o con la mediana nel caso categorico), dei valori non nulli delle realizzazioni della stessa variabile e le variabili categoriche sono state trasformate in fattori.

2.2 Analisi esplorativa dei dati

Caret mette a disposizione la funzione 'featurePlot' per l'analisi esplorativa dei dati, la quale offre un'interfaccia semplificata volta a produrre dei grafici elementari, sia per regressione che classificazione, al costo di una limitata personalizzazione dei grafici, che viene invece garantita molto ampiamente da librerie esterne come GGally [7] o ggplot2 [8].

Nella figura 1, confrontiamo uno scatterplot prodotto dalla funzione 'featurePlot' di Caret, con analoghe funzioni di altri pacchetti di GGally e ggplot2, e concludiamo che per eseguire un'analisi esplorativa dei dati su dataset complessi, le funzionalità di visualizzazione dei dati di Caret non diamo abbastanza controllo, a differenza degli altri pacchetti citati, sopratutto il pacchetto ggplot2.

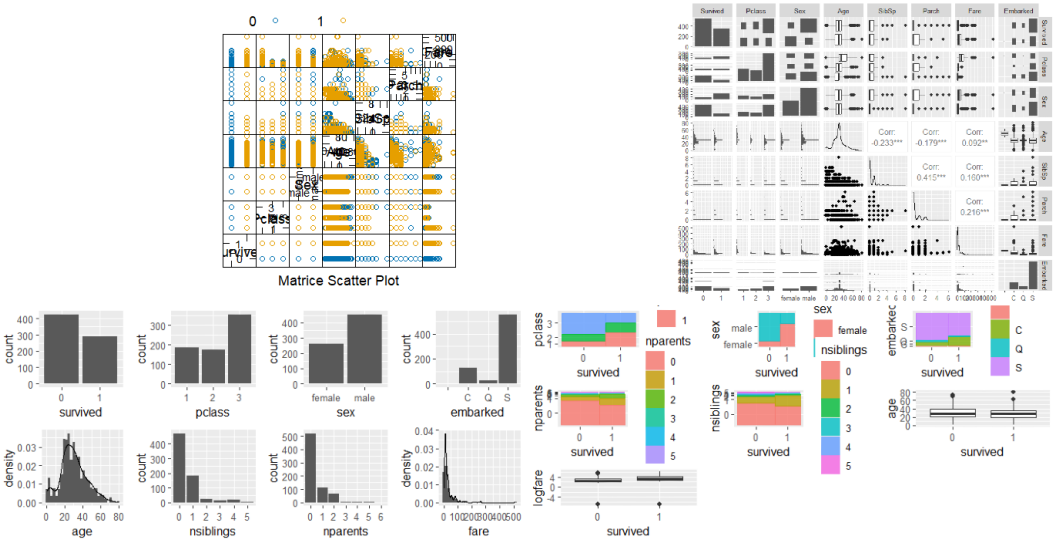

Figure 1: In alto a sinistra il grafico prodotto da Caret che produce uno scatterplot tra tutte le variabili, senza tenere in considerazione quali siano dei fattori, mentre in alto a destra il grafico prodotto da GGally, il quale discrimina tra variabili categoriche e numeriche. Entrambe le funzioni offrono una scelta limitata di grafici, in basso due grafici che analizzano rispettivamente la distribuzione di probabilità di ogni variabile esplicativa a destra, e la dipendenza di ogni variabile esplicativa con la variabile risposta a sinistra.

Utilizzando i risultati dell'analisi esplorativa possiamo eliminare alcune variabili o applicare semplici trasformazioni (per esempio applicare il logaritmo ad una variabile). Nella prossima sezione vediamo come applicare trasformazioni più complicate attraverso Caret.

3 Pre-processing e Partizionamento dei dati

In questa sezione analizziamo il pre-processing dei dati, principalmente illustrando le funzionalità della funzione "PreProcess" di Caret, nel centramento e scaling dei dati, e nell'applicazione di tecniche della riduzione della dimensionalità, oltre che all'individuazione dei predittori con Zero-variance e Near-zero variance tramite la funzione 'nearZeroVar'.

3.1 Predittori zero-variance

I predittori Zero-variance (ZVP), sono delle variabili esplicative, sia categoriche che numeriche, che comunemente si trovano in grandi dataset, e che sono definite dall'avere lo stesso valore ripetuto in tutte le realizzazioni del dataset (si dice a varianza zero, perchè la varianza delle realizzazioni di quel predittore è zero, essendo tutti gli elementi uguali), e che è buona prassi eliminare dal dataset, poichè poco informative [2, 9, 10].

Una seconda tipologia di predittori che bisogna tenere inconsiderazione sono i predittori Near-Zero Variance (NZVP), i quali sono definiti dall'avere un singolo valore ripetuto la maggior parte delle volte, e di conseguenza una varianza vicina a zero. L'eliminazione dei NZVP non è consigliata, visto che un NZVP potrebbe essere comunque informativo a differenza di un ZVP (pensiamo a una variabile categorica binaria che è quasi sempre della classe $c1$, quelle volte che capita $c2$ lo vogliamo sapere, e quindi la variabile è informativa anche se è una NZVP)[10].

Caret offre la funzione 'nearZeroVar' [2] per individuare sia i ZVP, che i NZVP, la quale prende in input un dataset, e ritorna gli indici dei ZVP e NZVP.

3.2 Centramento, Scale e riduzione della dimensionalità

La trasformazione dei dati è eseguita da una serie di tecniche volte ad ottenere un dataset che permetta di rispettare dei prerequisiti di un modello, o a diminuirne l'overfitting e i tempi di addestramento.

Tra le trasformazioni dei dati più comuni troviamo il centramento e scaling, oppure le tecniche di riduzione della dimensionalità, come il Principal Component Analysis (PCA), un algoritmo che, catturando la varianza delle variabili numeriche, crea delle componenti principali a cui è assegnato uno score (eigenvalore), tale per cui le componenti principali con uno score basso possono essere scartate, e di conseguenza è possibile ridurre il dataset, perdendo un'informazione minima [6].

Il pacchetto Caret offre la funzione 'preProcess' [2], la quale, variando l'argomento 'method' della funzione 'train', applica un insieme di tipologie di preprocessing, e ritorna un oggetto contenente il dataset trasformato. Se si specifica 'method = center', si portano i dati ad avere media zero, con 'method = scale', si portano i dati ad avere varianza 1, infine, specificando 'method = range' si normalizzano i dati.

La funzione 'preProcess' permette anche di applicare il Principal Component

Analysis (PCA) a tutte le variabili numeriche, specificando 'method=pca', i dati vengono scalati e centrati automaticamente prima della sua applicazione, come possiamo vedere nell'algoritmo 1. È possibile specificare un ulteriore argomento quando si utilizza PCA, 'thresh= X', dove $X \in [0,1]$ (di default $X = 0.95$), indica la quantità di informazione che vogliamo preservare dopo la trasformazione, e di conseguenza l'eliminazione automatica delle componenti principali che contengono la percentuale $1 - X$, meno informativa.

Allo scopo di convertire le variabili categoriche in dummies, Caret offre la funzione 'dummyVars', la quale converte le variabili categoriche, mantenendo invariate le variabili numeriche, come possiamo vedere nell'algoritmo 1.

Algorithm 1 Applicazione delle funzioni di pre-processing di Caret al dataset D HousePredictions

Require: dataset D, contiene variabili miste

1: dummy = dummyVars(SalePrice $\sim$., data = D)
2: D_{dummy} = predict(dummy, newdata = D)
3: PCA = preProcess(D_{dummy}, method="pca", thresh=0.95)
4: D_{pca} = predict(PCA, newdata = D_{dummy})

3.3 Partizionamento dei dati

Caret offre la funzione 'createDataPartition' che partiziona un dataset D, in due dataset distinti, D_{train} e D_{test}, in base ad un argomento $p = val, val \in [0,1]$, come possiamo vedere nell'algoritmo 2. È buona prassi accompagnare una funzione di partizionamento dei dati ad una funzione di mescolamento casuale del dataset, in modo da assicurarsi che il training, e il test set siano rappresentativi.

Algorithm 2 Applicazione di data splitting al dataset D HousePredictions tramite Caret

Require: dataset D_{pca}, contiene variabili miste

1: set.seed(2321)
2: D_{pca} = D_{pca}[sample(1:nrow(D_{pca}), replace = TRUE)]
3: idx = createDataPartition(D_{pca}\$SalePrice, $p = 0.85$, list = FALSE)
4: D_{train} = dataset[idx,]
5: D_{test} = dataset[-idx,]

4 Scelta dei parametri di un modello tramite resampling

Caret è un pacchetto particolarmente efficace nel raggruppare modelli provenienti da diversi pacchetti, ed offre un'unica funzione 'train' che gestisce non solo molti algoritmi di machine learning con parametri standardizzati, ma anche metriche e tecniche di resampling.

In questa sezione illustriamo la funzione 'train' e le sue funzioni ausiliarie, attraverso esempi dal dataset 'HousePrices' [5].

4.1 Addestramento dei modelli

Caret raggruppa diversi pacchetti di machine learning, come 'ranger' [11] o 'randomForest' [12] per i Random Forest, 'kernlab' [13] per SVM, 'naivebayes' e 'bnclassify' [14] per i classificatori bayesiani, e molti altri, una lista completa dei metodi forniti da Caret è disponibile nella repository di Caret [2].

La funzione 'train', permette di specificare uno qualsiasi di questi modelli come argomento, come si vede negli algoritmi 3 e 4.

4.2 Tecniche di resampling e tuning dei modelli

Oltre a raggruppare molti modelli di machine learning in un unico pacchetto, il maggiore punto di forza di Caret riguarda la sua capacità di poter applicare più tecniche di resampling e fare tuning dei parametri in modo trasversale per tutti i modelli, attraverso la stessa funzione 'train', usata per specificare il modello.

Caret offre la possibilità di trovare i parametri migliori specificando una matrice che contenga una lista dei parametri per il modello selezionato tramite il parametro tuneGrid, come illustrato nell'algoritmo 3, in alternativa è possibile utilizzare il parametro $tuneLength = n, n \in \mathbb{N}$, con il quale viene generata una lista di n combinazioni di parametri di tuning che Caret pensa siano migliori per quel modello, come possiamo vedere nell'algoritmo 4.

In entrambi i casi, ogni combinazione di parametri viene valutata dalla funzione 'train' secondo una metrica specificata come argomento, e il modello migliore è contenuto nell'oggetto ritornato dalla funzione 'train', nella figura 2 vediamo il risultato del tuning dei modelli Random Forest e SVM radiale sul dataset 'HousePrices' [5].

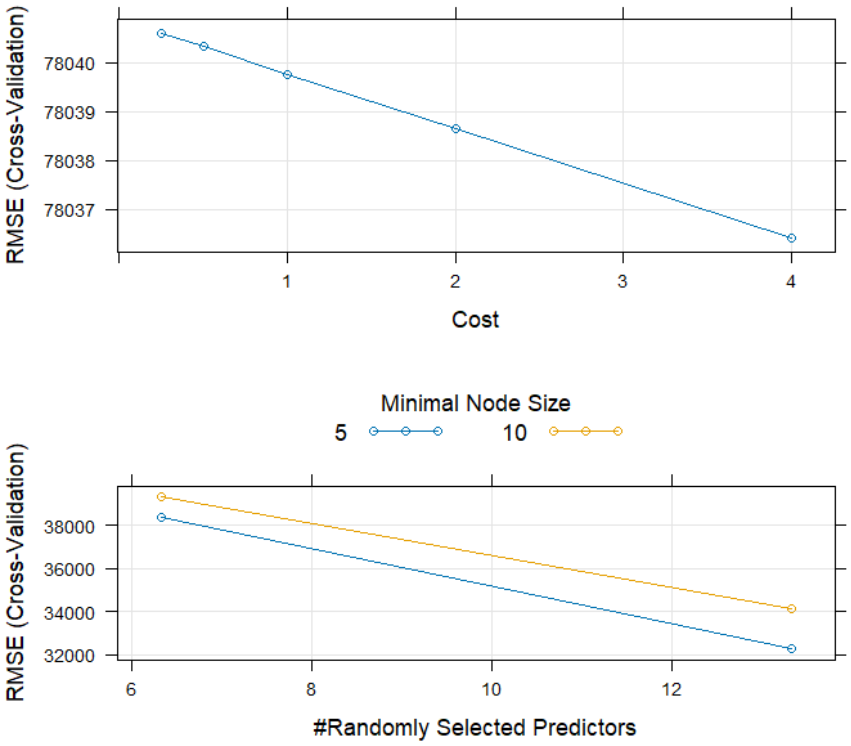

Figure 2: In alto il grafico che rappresenta il variare della metrica RMSE, al variare del parametro di tuning 'Cost' per il modello SVM radiale, mentre in basso il grafico rappresenta il variare della stessa metrica, al variare di due parametri di tuning, 'mtry' e 'quantità di esempi minimi per il taglio', per il modello Random Forest. Entrambi i modelli sono stati addestrati con il dataset 'HousePrices'

L'ultimo argomento rilevante offerto dalla funzione 'train' di Caret riguarda una selezione di tecniche di resampling, tra le quali troviamo cross validation, cross validation con ripetizione e bootstrapping. La funzione 'train' prende come argomento opzionale un oggetto di tipo 'trainControl', generato dall'omonima funzione ausiliaria di Caret, e che permette di specificare il metodo di resampling, e gli eventuali parametri che esso richiede, vedi gli algoritmi 3 e 4, applicati al dataset 'HousePrices'.

Algorithm 3 Addestramento di un modello Random Forest con cross validation come tecnica di resampling

Require: trainset D

1: control = trainControl(method="cv", number=10)
2: grid = expand.grid(mtry=c(ncol(D)/3, sqrt(ncol(D))), splitrule="variance", min.node.size = c(5,10))
3: model.rf = train(SalePrice∼., data=D, method="ranger", metric = "RMSE", tuneGrid = grid, trControl=control, importance="impurity")

Algorithm 4 Addestramento di un modello SVM radiale con bootstrapping come tecnica di resampling

Require: trainset D

1: control = trainControl(method="boot", number=25)
2: model.svm = train(SalePrice∼., data=D, method="svmRadial", tuneLength=5, trControl = control, importance = TRUE)

4.3 Parallelizzazione

La computazione parallela consiste nell'esecuzione parallela di processi non sequenziali, come i metodi di resampling, su differenti unità di calcolo.

In Caret, il costo maggiore in termini computazionali avviene durante il tuning dei parametri del modello, nel quale si usano tecniche di resampling come cross-validation o bootstrapping, le quali addestrano lo stesso modello con diversi parametri e set di dati, e dunque la parallelizzazione risulta particolarmente vantaggiosa, sia a livello teorico che sperimentale [15]. Come vediamo nella figura 3, in cui un modello 'boosted tree' con cross-validation e varie combinazioni di parametri di tuning è stato testato su macchine diverse sia con calcolo parallelo che sequenziale, il valore di speedup aumenta considerevolmente all'aumentare delle unità di computazione disponibili.

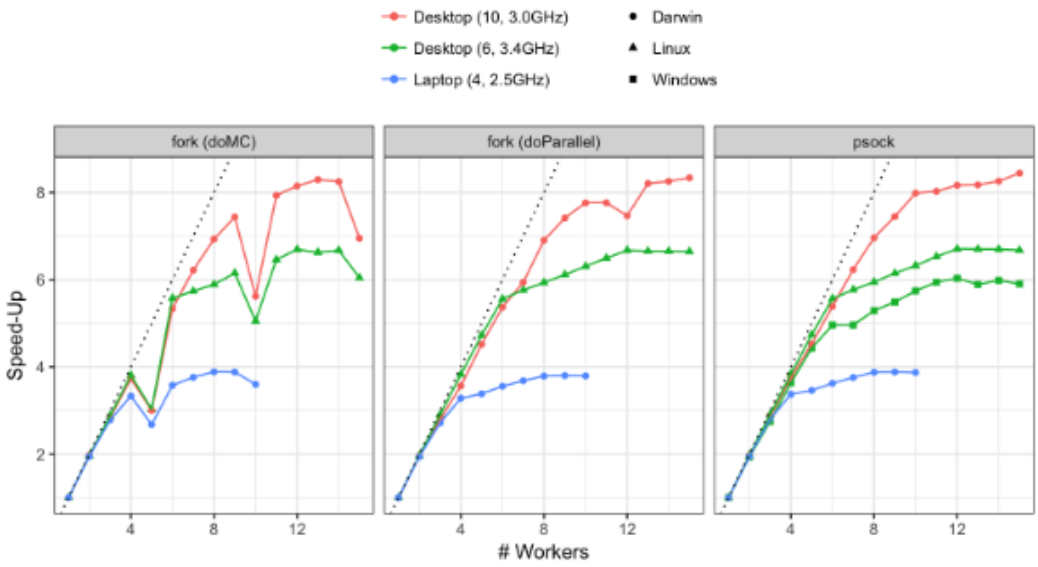

Figure 3: Nella figura a sinistra lo speedup, un valore che indica la riduzione dei tempi di esecuzione dati dal parallelismo, rispetto al calcolo sequenziale (uno speedup di tre, significa che l'algoritmo parallelo è tre volte più veloce di quello sequenziale), per tre macchine che hanno a disposizione rispettivamente quattro, sei e dieci unità di calcolo (rappresentate dai colori). I tempi di esecuzione nei grafici misurano lo speedup di un modello boosted tree con Caret con cross-valition di 10 split, e con una griglia di parametri di tuning per un totale di 1250 modelli da addestrare, i quali possono essere addestrati in parallelo. I grafici centrale e di destra eseguono le stesse misurazioni del grafico di sinistra con librerie di computazione parallela diverse [15].

Per parallelizzare il calcolo dei modelli durante il resampling, Caret necessita di librerie esterne. Come possiamo vedere nell'algoritmo 5, noi utilizziamo la libreria 'doParallel'.

Algorithm 5 Addestramento di un modello Naive Bayes con cross validation 10 splits in modo parallelo

Require: trainset D

```
cl = makePSOCKcluster(5)                                   ▷ Register parallelism
registerDoParallel(cl)
control = trainControl(method="cv", number=10)
model.bayes = train(Survived∼ ., data = D, method="naive_bayes", tuneLength = 3, trControl = control, importance = TRUE)
registerDoSEQ()                                            ▷ Deregister parallelism
stopCluster(cl)
```

Alcuni modelli, come Random Forest, sono predisposti per construzione al parallelismo non solo a livello di resampling, ma anche all'interno di un singolo modello. Questo tipo di parallelizzazione non è gestita direttamente da Caret, ma dai pacchetti che esso include, per esempio, nella funzione 'train', se sispecifica il metodo 'ranger', otteniamo random forest parallelizzati, mentre specificando il metodo 'random_forest', otteniamo una random forest non parallelizzata [11, 12].

5 Valore di importanza delle variabili

Il valore di importanza delle variabili di un dataset D, con cui è stato addestrato un modello M, indica, per ogni variabile, quanto quella variabile abbia contribuito ad nell'ottenere predizioni accurate, ed è calcolato dalla funzione 'varImp' di Caret in base al modello fornito. Per esempio, nei modelli lineari si usa il valore assoluto delle t-statistiche, mentre per i Random Forest si usano i dati OutOfBag (OOB) [2], come possiamo vedere nella figura 4. Inoltre, per i modelli che prevedono tecniche specifiche per il modello per la stima dell'importanza (come i modelli lineari e i random forest), è necessario specificare il parametro 'importance' nella funzione 'train', come possiamo vedere negli algoritmi 3 e 4.

Se non è possibile computare l'importanza con metodi specifici per il modello, dobbiamo utilizzare delle tecniche indipendenti dal modello [2]. Caret implementa queste tecniche, che variano in base che al tipo di analisi sia di regressione, oppure di classificazione.

```
> varImp(model.rf)
ranger variable importance

  only 20 most important variables shown (out of 197)

                    Overall
ExterQualTA          100.00
KitchenQualTA         87.77
GarageFinishUnf       66.40
BsmtQualTA            61.99
NeighborhoodNoRidge   60.72
FoundationPConc       60.15
ExterQualGd           51.52
GarageTypeDetchd      50.95
BsmtFinType1GLQ       49.29
KitchenQualGd         45.99
NeighborhoodNridgHt   44.17
RoofStyleHip          37.57
RoofStyleGable        37.21
BsmtQualGd            35.14
Id                    33.83
MasVnrTypeNone        27.20
FoundationCBlock      22.78
HouseStyle2Story      22.64
SaleTypeNew           22.04
Exterior2ndImStucc    21.14
```

Figure 4: Il valore di importanza delle 20 variabili più importanti del modello Random Forest, addestrato per la classificazione usando il dataset 'Titanic' [4]. Dati gli score di importanza, è possibile eliminare le variabili con score minore, addestrare e valutare un nuovo modello (feature selection)

La stima dell'importanza delle variabili di un modello può essere usata per eliminare le variabili meno importanti, e di conseguenza ridurre l'overfitting e il tempo di addestramento del modello, e aumentare l'accuratezza [16].

6 Predizioni e analisi delle prestazioni

6.1 Funzione 'predict'

Caret crea un wrapper attorno alla funzione 'predict' di *R*, e richiede di specificare sia il modello addestrato, che il testset, come illustrato nell'algoritmo 6. Nel caso

si abbia un modello di classificazione, se si vogliono ritornare le predizioni sotto forma di probabilità, bisogna specificarlo sia nella funzione 'train' che in quella di predizione, come illustrato nell'algoritmo 6.

Algorithm 6

Require: trainset T

1: control = trainControl(method="cv", number=10, **classProbs=TRUE**, savePredictions = TRUE)
2: model.svm = train(Survived ., data = T, method="svmRadial", tuneLength = 10, trControl = control)
3: predictions = predict(model.svm, newdata = testset, **type = "prob"**) ▷ Predizioni sotto forma di probabilità
4: predictions = predict(model.svm, newdata = testset) ▷ Predizioni sotto forma di classi

6.2 Valutazione dei modelli di regressione: RMSE e R^2 aggiustato

Root mean squared error (RMSE) e R^2 aggiustato sono delle metriche che valutano l'efficacia del modello nella predizione dei valori osservati, e vengono utilizzate per confrontare diversi modelli addestrati, come possiamo vedere nella figura 5, dove si confrontano i modelli Random Forest e SVM radiale applicati al dataset 'HouseSales'.

Caret permette di applicare queste metriche sia nella funzione 'train', allo scopo di selezionare la combinazione di parametri di tuning migliore per il modello durante il resampling tramite l'argomento 'metric', come possiamo vedere nell'algoritmo 3, che nella funzione 'postResample', che prende gli stessi argomenti della funzione 'predict' e ritorna le metrice RMSE e R^2, allo scopo di confrontare le predizioni, con le osservazioni di un insieme di dati su cui il modello non sia stato addestrato.

```
> Predictions with Random Forest
        RMSE     Rsquared          MAE
31080.526693     0.839445 19637.141919
> Predictions with SVM with radial kernel (for regression)
        RMSE     Rsquared          MAE
7.285115e+04 5.779602e-01 5.183092e+04
```

Figure 5: Sopra la funzione 'postResample' applicata al modello Random Forest scelto tramite resampling, sotto lo stesso per il modello SVM radiale. Dai risultati, vediamo che il modello Random Forest è migliore del metodo SVM radiale.

6.3 Valutazione dei modelli di classificazione: matrice di confusione, accuratezza e Kappa di Cohen

L'accuratezza e la Kappa di Cohen sono metriche usate per valutare un modello, e utili al fine di confrontare una serie di modelli addestrati e alla selezione del modello migliore, come possiamo vedere nella figura 6, dove si confrontano i modelli di classificazione addestrati sul dataset 'Titanic'.

Entrambe le metriche vengono calcolate a partire dalla matrice di confusione, una matrice che nel caso di una variabile risposta categoriale a due fattori presenta quattro entrate, veri positivi (TP), falsi positivi (FP), veri negativi (TN) e falsi negativi (FN), ma che può essere generalizzata a variabili categoriali a più di due fattori. Tramite la matrice di confusione, oltre alle metriche di accuratezza e Kappa di Cohen, è possibile anche calcolare la sensibilità e la specificità, che possono essere più informative dell'accuratezza nel caso sia più importante minimizzare i FP, o i FN.

```
> SVM
> confusionMatrix(predictions, observations)
          Reference
Prediction   c1   c2
        c1 57.5 12.5
        c2  4.1 25.9
 Accuracy (average) : 0.8338

> NAIVE BAYES
> confusionMatrix(predictions, observations)
          Reference
Prediction    0    1
         0 52.2 14.0
         1  9.4 24.4

 Accuracy (average) : 0.7665

> RANDOM FOREST
> confusionMatrix(predictions, observations)
          Reference
Prediction   c1   c2
        c1 57.8 13.2
        c2  3.8 25.2
 Accuracy (average) : 0.8298
```

Figure 6: Sopra la matrice di confusione e l'accuratezza calcolate per il modello SVM radiale, in mezzo lo stesso per il modello naive bayes e sotto il modello Random Forest. In tutti e tre i casi la combinazione di parametri di tuning migliore è stata già eseguita tramite resampling.

Caret offre la funzione 'confusionMatrix', che date le predizioni fatte da un modello su un set di dati, e le relative osservazioni, calcola la matrice di confusione, e varie statistiche come illustrato nella figura 7. Notiamo che, rispetto alla figura 6, l'applicazione della funzione 'confusionMatrix' nella figura 7 contiene più informazioni, questo perchè nel primo caso parte dei risultati sono stati tagliati per facilità di rappresentazione.

```
> confusionMatrix(predictions, observations)
Confusion Matrix and Statistics

          Reference
Prediction c1 c2
        c1 76 25
        c2  6 26

               Accuracy : 0.7669
                 95% CI : (0.6858, 0.8358)
    No Information Rate : 0.6165
    P-Value [Acc > NIR] : 0.0001658

                  Kappa : 0.4697

 Mcnemar's Test P-Value : 0.0012254

            Sensitivity : 0.9268
            Specificity : 0.5098
         Pos Pred Value : 0.7525
         Neg Pred Value : 0.8125
             Prevalence : 0.6165
         Detection Rate : 0.5714
   Detection Prevalence : 0.7594
      Balanced Accuracy : 0.7183

       'Positive' Class : c1
```

Figure 7: La matrice di confusione calcolata sulle predizioni del modello Random Forest, sul dataset 'Titanic'. Vediamo che il valore di specificità è molto basso, mentre quello di sensitività è molto alto, di conseguenza questi modello sovrastima le persone non sopravviveranno, mentre è accurato nel predire le persone che sopravviveranno, non il meglio per il marketing di una nave da crociera.

6.4 Curve ROC

La receiver operating characteristic, o ROC, è un grafico che illustra le performance di un classificatore binario M (che può anche essere estesa ai classificatori multiclasse), in base alla variazione dei valori di una threshold t, un parametro della funzione 'roc' la quale, valutando il valore di probabilità deciso da $M(D)$, in un classificatore binario $M(x) \in [0,1]$, e variando il parametro t, valuta le performance del modello M sul dataset, producendo una matrice di confusione per ogni

valore di t [17].

$$\text{classe di x} = \begin{cases} classe1 & M(x) \geq t \\ classe2 & M(x) < t \end{cases} \tag{1}$$

Caret implementava in versioni precedenti una funzione ”roc”, da applicare ad un modello, ma è stata deprecata. Riportiamo ugualmente un pacchetto alternativo per l’implementazione di ROC, ”pROC”, che utilizza le funzioni ”roc”, e ”coords”, come illustrato nell’algoritmo 7, e la funzione ”plot.roc”, nella figura 8.

Algorithm 7 Utilizzo del pacchetto pROC, per trovare la threshold migliore del modello SVM radiale, addestrato con il dataset ”Titanic”

Require: model M, testset T

1: library(pROC)
2: predictions = predict(M, T)
3: rc = roc(T[,”Survived”], predictions$class1)
4: best_threshold = coords(rc, ”best”, ret = ”threshold”) ▷ Retrieve the threshold which minimizes the tradeoff between Sensitivity and Specificity
5: plot.roc(T[,”Survived”], predictions$class1)

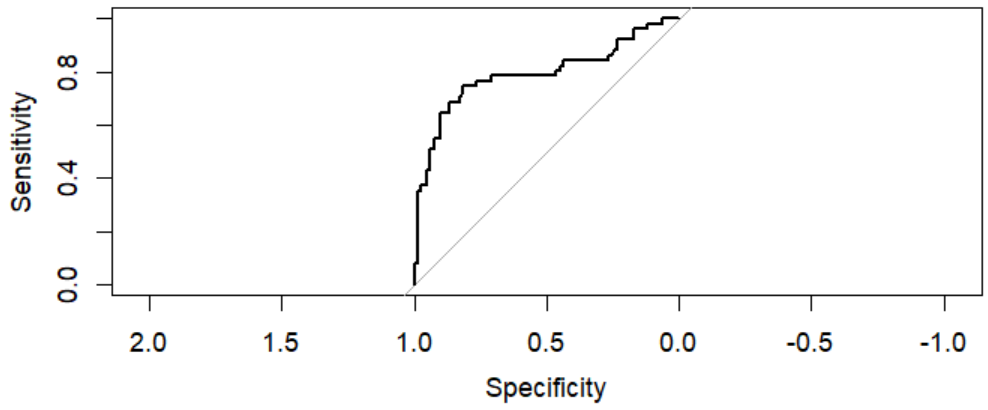

Figure 8: curva ROC del dataset ”Titanic”, con il modello SVM radiale

7 Conclusione

In questo lavoro abbiamo illustrato le funzionalità del pacchetto Caret, attraverso la sua applicazione ai dataset 'Titanic' e 'House Prices', per i quali abbiamo ottenuto un'accuratezza di 0.83, con il modello SVM radiale sul dataset 'Titanic', e un valore R^2 di 0.84, con il modello Random Forest sul dataset 'House Prices'

L'utilizzo di Caret è stato essenziale nelle parti di resampling dei modelli parallela, e di analisi delle prestazioni, senza il quale avremmo dovuto, invece, utilizzare un considerevole numero di librerie separate, mentre per quanto riguarda le fasi di visualizzazione dei dati e preprocessing, riteniamo Caret uno strumento utile ma non essenziale.

Concludendo riteniamo che Caret contenga funzionalità essenziali per costruire modelli basati sul machine learning in R, tuttavia riteniamo anche che, per alcune delle funzionalità che Caret offre, come la visualizzazione dei dati, librerie esterne come ggplot2 o GGally siano più efficaci.

8 Riferimenti

References

[1] "Cran." [Online]. Available: https://cran.r-project.org/web/packages/

[2] M. Kuhn, *Repository of Caret*, 2019. [Online]. Available: https://topepo.github.io/caret

[3] M. Kuhn and H. Wickham, "tidymodels: Easily install and load the 'tidymodels' packages." [Online]. Available: https://cran.r-project.org/web/packages/tidymodels/index.html

[4] *Titanic - Machine Learning from Disaster*. Kaggle competitions, 2017. [Online]. Available: https://www.kaggle.com/competitions/titanic

[5] *House Prices - Advanced Regression Techniques*. Kaggle competitions, 2017. [Online]. Available: https://www.kaggle.com/competitions/house-prices-advanced-regression-techniques

[6] "Analisi delle componenti principali (pca)." [Online]. Available: https://it.wikipedia.org/wiki/Analisi_delle_componenti_principali

[7] B. Schloerke, D. Cook, J. Larmarange, F. Briatte, M. Marbach, E. Thoen, A. Elberg, O. Toomet, J. Crowley, H. Hofmann, and H. Wickham, "Ggally: Extension to 'ggplot2'." [Online]. Available: https://cran.r-project.org/web/packages/GGally/index.html

[8] H. Wickham, W. Chang, L. Henry, T. L. Pedersen, K. Takahashi, C. Wilke, K. Woo, H. Yutani, D. Dunnington, and T. van den Brand, "ggplot2: Create elegant data visualisations using the grammar of graphics." [Online]. Available: https://cran.r-project.org/web/packages/GGally/index.html

[9] M. Kuhn, "Building predictive models in r using the caret package," 2008. [Online]. Available: https://www.jstatsoft.org/v28/i05/paper

[10] thiagogm, "Near-zero variance predictors. should we remove them?" 2014. [Online]. Available: https://tgmstat.wordpress.com/2014/03/06/near-zero-variance-predictors

[11] M. N. Wright, S. Wager, and P. Probst, "Package 'ranger'." [Online]. Available: https://cran.r-project.org/web/packages/ranger/ranger.pdf

[12] L. Breiman, A. Cutler, A. Liaw, and M. Wiener, "Package 'randomforest'." [Online]. Available: https://cran.r-project.org/web/packages/randomForest/randomForest.pdf

[13] A. Karatzoglou, A. Smola, K. Hornik, M. A. Maniscalco, and C. H. Teo, "kernlab: Kernel-based machine learning lab." [Online]. Available: https://cran.r-project.org/web/packages/kernlab/index.html

[14] M. Bojan, B. Concha, L. Pedro, and W. Hadley, "bnclassify: Learning discrete bayesian network classifiers from data." [Online]. Available: https://cran.r-project.org/web/packages/bnclassify/index.html

[15] M. Kuhn and K. Johnson, "While you wait for that to finish, can i interest you in parallel processing?" [Online]. Available: http://appliedpredictivemodeling.com/blog/2018/1/17/parallel-processing

[16] "Feature selection." [Online]. Available: https://h2o.ai/wiki/feature-selection

[17] "Receiver operating characteristic." [Online]. Available: https://it.wikipedia.org/wiki/Receiver_operating_characteristic