Bankruptcy prediction

Project Work ASLII: Classification

Dario Falchetta

04-06-2024

Contents

Descrizione del dataset

Il dataset, reperibile da UCI Machine Learning Repository contiene dati ottenuti dal Taiwan Economic Journal e riguardanti imprese taiwanesi tra il 1999 ed il 2009. L'obiettivo è quello di prevedere la probabilità che un'azienda finisca in stato di bancarotta, a partire da una serie di indicatori di impresa, per un totale di 6819 osservazioni riguardanti principalmente imprese del settore manufatturiero (346), dei servizi (39) ed altre (93). Il dataset contiene 95 di questi indicatori, che verranno tutti usati come predittori, ed una colonna per la variabile di risposta, una variabile binaria che indica se l'impresa è poi andata in bancarotta o meno nel periodo successivo.

Gli aspetti più importanti per la previsione della bancarotta sono i metodi statistici e di machine learning utilizzati e gli indici finanziari considerati. Questi ultimi possono essere divisi in due categorie:

- Financial Ratios (FR): sono indici creati utilizzando alcuni aspetti quantitativi di un'azienda in modo da poter ottenere informazioni significative su di essa. Possono essere a loro volta classificabili in coefficienti di:
  - Solvibilità (Solvency Ratios)
  - Redditività (Profitability Ratios)
  - Flusso di cassa (Cash Flow Ratios)
  - Struttura di capitale (Capital Structure Ratios)
  - Turnover (Turnover Ratios)
  - Growth (Growth Ratios)
  - Altri
- Corporate Governance Indicators (CGI): includono meccanismi, processi e relazioni attraverso le quali le aziende sono controllate e dirette. Possono essere a loro volta classificabili in:
  - Consiglio di amministrazione (Board structure)
  - Struttura/assetto proprietario (Ownership structure)
  - Diritti di flusso di cassa (Cash flow rights)
  - Ritenzione dei dipendenti (Key person retained)
  - Altri

Molte delle definizioni sono state reperite dalle seguenti fonti:

- paper che ha utilizzato il dataset;
- Investopedia;
- Wikipedia.

FR: Coefficienti di solvibilità (Solvency Ratios)

Misurano il livello di patrimonializzazione di un'azienda. Sono calcolati come rapporto tra i fondi propri (Own Funds) e il requisito di capitale di solvibilità (Solvency Capital Requirement) a una certa data. I fondi propri sono determinati a partire dalla valutazione del bilancio del'azienda a valori di mercato (Fair Value) mentre lo SCR è determinato valutando i rischi sottostanti al business sviluppato secondo quanto previsto dalla Normativa Solvency II.

Sostanzialmente, indicano se un'azienda ha un flusso di moneta che le permetta di rispettare i debiti a lungo termine con i suoi creditori.

FR: Coefficienti di redditività (Profitability Ratios)

Il coefficiente di redditività è la percentuale che, nel Regime Forfettario, viene applicata agli incassi conseguiti da un'azienda per ottenere il reddito imponibile sul quale verrà calcolata l'imposta sostitutiva e i contributi da pagare.

Il reddito imponibile si calcola, generalmente, sottraendo i costi dell'attività al fatturato generato. Nel Regime Forfettario, invece, il reddito imponibile si ricava con l'applicazione del coefficiente di redditività. Il coefficiente di reddività, inoltre, non è il medesimo per tutte i tipi di attività, ma varia in base al Codice ATECO di riferimento (codice che identifica il tipo di azienda).

FR: Coefficienti di flusso di cassa (Cash Flow Ratios)

Simili ai coefficienti di redditività, questi rappresentano la capacità di un'azienda di saldare i suoi debiti attuali con il flusso di cassa ottenuto nello stesso periodo.

FR: Coefficienti di struttura del capitale (Capital Structure Ratios)

Rappresentano il mix di debiti ed equity (quote dell'azienda di proprietà degli azionisti) che l'azienda usa per finanziare le sue attività e crescere.

Il rapporto debt-to-equity è solitamente usato per misurare la struttura del capitale di un'azienda e può essere vista come rappresentativa del suo livello di rischio: un'azienda con un'alto valore di debito nella struttura del suo capitale può essere considerata come rischiosa per i suoi investitori ma, allo stesso tempo, potrebbe avere un maggiore potenziale di crescita.

FR: Coefficienti di turnover (Turnovers Ratios)

Si tratta della percentuale di fondi in un certo portafoglio che sono stati rimpiazzati nell'arco di un anno. Alcuni fondi durano meni di 12 mesi, il che significa che il loro coefficiente di turnover è maggiore del 100%. Tuttavia, questo non significa necessariamente che l'intero portafoglio è stato rimpiazzato.

Un turnover ratio non dovrebbe essere utilizzato esclusivamente per decidere se investire o meno nelle azioni di un'azienda. Tuttavia, valori stranamente alti o bassi per questi indici dovrebbero portare gli interessati ad investigare sulle performance di quel portafoglio nel tempo, in modo da poter decidere quanto sia stata efficiente la sua strategia di investimento.

FR: Coefficienti di crescita (Growth Ratios)

Rappresentano la percentuale di variazione di un certo aspetto dell'azienda in un certo arco di tempo. Ovviamente, se sono positivi, significa che l'azienda è cresciuta sotto quegli specifici punti di vista.

CGI: Consiglio di amministrazione (Board structure)

Il Consigli di amministrazione è l'insieme di individui che governano un'azienda, i cui membri sono eletti dagli shareholders per definire le strategie, occuparsi della gestione e proteggere gli interessi degli shareholders e degli stakeholders.

CGI: Struttura/assetto proprietario (Ownership structure)

È una variabile d'impresa che può essere definita come "la distribuzione dei diritti di proprietà, cioè che possono devono prendere le decisioni aziendali e godere dei loro risultati, tra i vari soggetti che partecipano alla vita dell'istituto".

CGI: Diritti di flusso di cassa (Cash flow rights)

Permettono ad un'azienda "acquirente" di monitorare da vicino una porzione dei profitti di un'azienda "target", in modo da ridurre le asimmetrie informative ed i costi di monitoraggio tra le due. Questi diritti garantiscono, all'azienda acquirente, una maggiore familiarità con gli obiettivi ed il profilo risk-reward dell'azienda target e le permettono di ridefinire, nel tempo, la quantità di capitale da investire in essa.

CGI: Ritenzione dei dipendenti (Key person retained)

Rappresenta la capacità di un'azienda di tenersi stretta i suoi dipendenti migliori, ovvero quelli i cui obiettivi a medio-lungo termine si allineano maggiormente con quelli dell'azienda stessa. Questo è vero soprattutto in periodi con poche assuzioni ed elevata competizione tra le aziende che cercano di accaparrarsi gli individui più qualificati.

La perdità di produttività ed i vantaggi competitivi sono alcune delle perdite più grandi che un'azienda subisce quando i suoi migliori dipendenti decidono di lasciarla, per non parlare di eventuali costi necessari per addestrare i nuovi sostituti o dell'impatto morale che quel licenziamento potrebbe avere sul resto degli impiegati.

La ritenzione dei dipendenti è perseguita principalmente attraverso tecniche di Human Resources. Alcune "strategie" per assicurarsi la fedeltà dei propri dipendenti includono apprezzare il loro lavoro, fornire stipendi e benefici competitivi ed incoraggiare un bilancio vita-lavoro più salutare.

Analisi dei dati

Data wrangling

```
dat <- read.csv(file = "./Dati/bankruptcy_data.csv", header = TRUE)
```

La variabile "Net.Income.Flag" presenta valore 1 per tutti i record. Non ha contenuto informativo quindi vado a rimuoverla.

```
dat <- dat %>%
  select(-Net.Income.Flag)
```

Siccome il senso intrinseco delle variabili va oltre l'obiettivo di questo progetto, ho deciso di non riportare l'intera lista e, per motivi di leggibilità, di andare a rinominare le colonne del dataset. I nomi originali del dataset sono reperibili dal sito sorgente. Inoltre, converto in categoriale la variabile di risposta Y, invertendo i suoi livelli perché le metriche di performance del pacchetto yardstick considerano il primo come classe dei positivi.

```
data <- dat %>%
  tibble() %>%
  rename_with( ~ c("Y", paste0("X", 1:(ncol(dat)-1)))) %>%
  mutate(Y = factor(Y, levels = c(1, 0)))
```

Il paper che accompagna il dataset afferma che le variabili sono state tutte normalizzate tuttavia, attraverso un superficiale summary, è possibile vedere che non è così e che invece ci sono variabili che non sembrano essere normalizzate.

```
data %>%
  select(X11) %>%
  summarise_all(range)
```

```
## # A tibble: 2 x 1
##          X11
##        <dbl>
## 1          0
## 2 9990000000
```

Le variabili in questione potrebbero essere espresse in termini percentuali, oppure far riferimento a dei conteggi. Ad ogni modo, vado a normalizzare anche quelle.

```
normFunc <- function(x){
  return({x - min(x)}/{max(x) - min(x)})
}

data <- data %>%
  mutate_at(.vars = c(2:ncol(data)), .funs = normFunc)
```

EDA

Variabile di risposta

Il problema della bankruptcy prediction è un problema intrinsecamente sbilanciato perché, ragionevolmente, le aziende che falliscono sono meno di quelle che riescono a sopravvivere. Questo si tradurrà in una classe dei positivi poco numerosa ovvero i record per i quali Y=1 saranno molti di meno rispetto a quelli per i quali Y=0.

```
data %>%
  group_by(Y) %>%
  summarise(n = n())
```

```
## # A tibble: 2 x 2
##   Y         n
##   <fct> <int>
## 1 1       220
## 2 0      6599
```

Quanto detto è infatti valido anche per il dataset in questione. Per affrontare questo problema, andrò a confrontare i risultati ottenuti sui dati inalterati e su una loro versione trasformata, usando il metodo SMOTE per l'oversamplig dei dati.

Correlazione tra i predittori

Il dataset contiene un gran numero di variabili quindi andare a considerare gli scatterplot tra tutte le coppie oppure l'intera matrice di correlazione, sarebbe poco informativo. Per questo motivo, scrivo una funzione che restituisce il numero di coppie di variabili la cui correlazione è maggiore di una certa soglia.

```r
check_cor <- function(data, cor_val){
  cor_mat <- cor(data[, -1])
  upper <- upper.tri(cor_mat, diag = FALSE)

  to_return <- rep(0, length(cor_val))

  for(i in 1:length(to_return)){
    to_return[i] <- ifelse(cor_val[i] > 0,
                           yes = sum(cor_mat[upper] > cor_val[i]),
                           no = sum(cor_mat[upper] < cor_val[i]))

  }

  to_return <- as.data.frame(matrix(to_return, ncol = length(to_return)))

  colnames(to_return) <- paste0("cor", ifelse(cor_val > 0, ">", "<"), cor_val)

  return(to_return)
}

cor_vals <- c(-0.99, -0.9, -0.8, -0.5, -0.3, 0.3, 0.5, 0.8, 0.9, 0.99)

check_cor(data[, -1], cor_val = cor_vals)
```

```
##   cor<-0.99 cor<-0.9 cor<-0.8 cor<-0.5 cor<-0.3 cor>0.3 cor>0.5 cor>0.8 cor>0.9
## 1         1        1        3       18       38     220     108      37      25
##   cor>0.99
## 1       11
```

I risultati ottenuti evidenziano la presenza di coppie di variabili più o meno fortemente collegate, sia da una relazione positiva che negativa. In particolare molte coppie di variabili hanno una correlazione superiore a 0.3 ed addirittura 11 coppie finiscono per avere correlazione praticamente uguale ad 1. Questo non dovrebbe essere troppo sorprendente dal momento che alcune delle quantità riportate nel dataset sono probabilmente calcolate a partire dalle stesse misure economiche e finanziarie, quindi dipendono dalle stesse quantità.

Essendo in ambito di reti neurali, correlazioni molto alte non ci portano a problemi di multicollinearità (che ci impedirebbero di stimare i parametri di un modello di regressione lineare) dal momento che non stiamo stimando modelli di regressione tradizionali. Avere una forte struttura di correlazione nei dati porterà la rete ad apprendere una loro rappresentazione efficiente molto velocemente. Questa potrebbe sembrare una cosa buona ma in realtà potrebbe essere causa di una serie di problemi:

- informazioni ridondanti: la presenza di features fortemente correlate può influenzare negativamente il processo di stima, introducendo inutili complessità e creando numerosi minimi locali, impedendogli di estrapolare dettagli importanti;
- instabilità nel gradiente: input correlati possono rendere instabile il calcolo del gradiente nell'algoritmo di backpropagation, mettendo in difficoltà il modello;
- interpretabilità e trasparenza: se diversi input hanno lo stesso contenuto informativo, diventa difficile stabilire quali sono quelli più rilevanti;

Per mitigare questi effetti ci sono diverse possibili tecniche che possono riguardare i dati, come la feature selection, extraction o eventuali operazioni di pre-processing, oppure il modello stesso, come l'utilizzo di

metodi ensemble oppure la regolarizzazione. In particolare, nel processo di stima della rete, confronterò i risultati includendo un termine di regolarizzaizone con quelli ottenuti senza di esso.

Fonti: CrossValidated e Quora.

Costruzione dei modelli

Per la costruzione dei modelli approfitterò della comodità del pacchetto tidymodels.

Pre-processing

Per un corretto addestramento dei modelli ed un'opportuna valutazione delle loro performance, vado a costruirmi il training set ed il test set.

```
set.seed(6719)
# Splitting in train e test set
splittedData <- rsample::initial_split(data, prop = 3/4)
dataTrain <- rsample::training(splittedData)
dataTest <- rsample::testing(splittedData)


# Splitting di features e target
yTrain <- dataTrain %>%
  select(Y)

xTrain <- dataTrain %>%
  select(-Y)

yTest <- dataTest %>%
  select(Y)

xTest <- dataTest %>%
  select(-Y)
```

Inoltre, vado a costruirmi anche i folds da usare nella cross-validation per la validazione del numero di hidden_units e del parametro di regolarizzazione.

```
set.seed(2851)
data_folds <- vfold_cv(dataTrain, v = 10)
```

Modello semplice

Per iniziare, stimo arbitrariamente un modello con 5 neuroni nell'hidden layer e nessuna penalizzazione sui parametri.

```
# Specificazione del modello
nnet_spec1 <- mlp(epochs = 50,
                  hidden_units = 5,
                  penalty = 0) %>%
  set_mode(mode = "classification") %>%
  set_engine("nnet")
```

Training

```
nnet_fit1 <- nnet_spec1 %>%
  fit(Y ~ ., dataTrain)
```

Testing

```
yPred <- predict(nnet_fit1, new_data = xTest)

confusionMatrix(reference = yTest$Y,
                data = yPred$.pred_class, mode = "sens_spec")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    1    0
##          1   15    8
##          0   31 1651
##
##                Accuracy : 0.9771
##                  95% CI : (0.9689, 0.9837)
##     No Information Rate : 0.973
##     P-Value [Acc > NIR] : 0.165780
##
##                   Kappa : 0.4244
##
##  Mcnemar's Test P-Value : 0.000427
##
##             Sensitivity : 0.326087
##             Specificity : 0.995178
##          Pos Pred Value : 0.652174
##          Neg Pred Value : 0.981570
##              Prevalence : 0.026979
##          Detection Rate : 0.008798
##    Detection Prevalence : 0.013490
##       Balanced Accuracy : 0.660632
##
##        'Positive' Class : 1
##
```

Il modello stimato produce risultati apparentemente ottimi in termini di accuratezza. Tuttavia, come detto precedentemente, questo è esclusivamente dovuto al fatto che le osservazioni appartenenti alla classe dei positivi sono molto meno numerose quindi, siccome l'indice di accuratezza dà lo stesso peso a classificazioni corrette per i negativi ed i positivi, questi ultimi vengono "nascosti" dai negativi.

Dato lo sbilanciamento tra classi, converrebbe utilizzare metriche che si concentrano sulla classe dei positivi:

- *Precision* (Positive Predictive Value): rappresenta la capacità intrinseca del modello di prevedere bene la classe dei positivi. Questo indice quindi è basato sulla realtà dal punto di vista del modello (al denominatore abbiamo il numero di previsti positivi dal modello).

$$Precision = P\{Y = 1|\hat{Y} = 1\} = \frac{TP}{TP + FP}$$

- *Recall* (Sensitivity, TPR): rappresenta il tasso di positivi che sono stati predetti correttamente rispetto al totale dei veri positivi. Questo indice è invece basato sulla realtà oggettiva (al denominatore abbiamo il numero di veri positivi del campione).

$$Recall = \frac{TP}{TP + FN}$$

- F_1-*score*: è una media armonica (simmetrica, dà lo stesso peso ad entrambe) delle due precedenti misure. Per costruzione, il valore massimo che può assumere è 1

$$F_1 - score = 2\frac{Prevision * Recall}{Precision + Recall}$$

- *balanced accuracy*: calcolata come media aritmetica tra *Sensitivity* e *Specificity*

$$bal\_accuracy = \frac{Sensitivity + Specificity}{2}$$

- *roc_auc*: area sotto la curva costruita a partire da *TPR* (Sensitivity, Recall) e *FPR* $\left(\frac{FP}{TN+FP}\right)$;
- *pr_auc*: area sotto la curva costruita a partire da *Precision* (PPV) e *Recall* (Sensitivity, TPR).

Confronto tra modelli

Tuning

Come primo tentativo di migliorare le performance ottenute dal modello precedente, provo a stimare un modello più complesso andando a fare cross-validation per validare il numero ottimale di neuroni nell'hidden layer e del valore del parametro di penalizzazione, in modo da prevenire l'overfitting. In particolare, per motivi di tempo e costo computazionale, vado a provare 9 combinazioni diverse per questi valori:

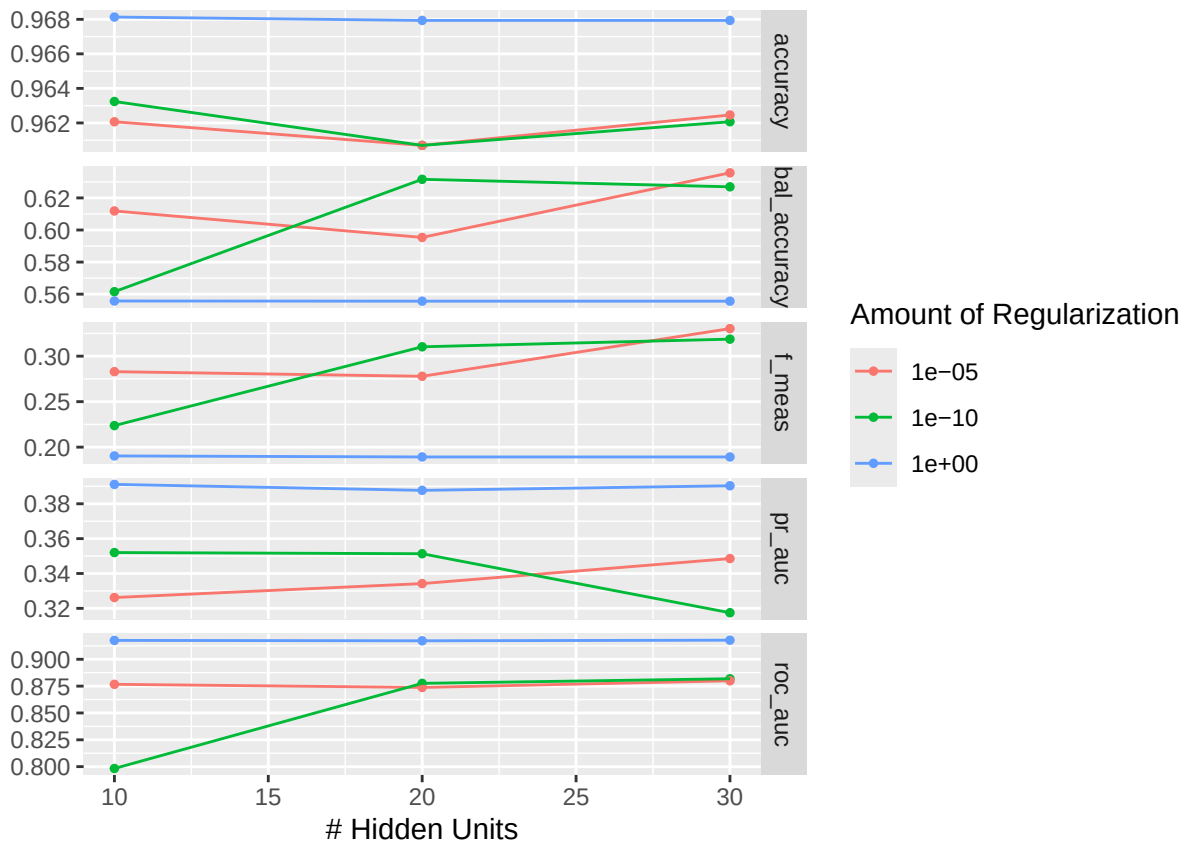
- i 3 valori per il numero di neuroni dell'hidden layer sono scelti sull'intervallo [10, 30] in modo da vedere cosa succede con una rete più complicata della precedente. Il software finisce per scegliere i valori 10, 20 e 30;
- la scelta dei 3 valori per il parametro di penalizzazione è lasciata completamente al software.

Quando la rete è composta da una gran numero di parametri, superiore ad un certo numero fissato (già dai 3000 circa), nnet ferma l'addestramento di default. Per evitare che questo accada, vado a cambiare il valore del parametro MaxNWts, fissandolo ad un valore arbitrariamente alto (ma non troppo, per evitare di stimare modelli inutilmente complicati).

```
# Salva le previsioni ed i risultati intermedi del workflow
keep_pred <- control_resamples(save_pred = TRUE, save_workflow = TRUE)

# Impostazioni per l'addestramento
nnet_spec_tune <- mlp(epochs = 50,
                      hidden_units = tune(),
                      penalty = tune()) %>%
  set_mode(mode = "classification") %>%
  set_engine("nnet", MaxNWts = 80000)


# Recipe
```

```
nnet_rec_tune <- recipe(Y ~ ., data = dataTrain)


# Workflow
nnet_wflow_tune <- workflow() %>%
  add_model(nnet_spec_tune) %>%
  add_recipe(nnet_rec_tune)


# Metriche di performance
class_and_probs_metrics <- metric_set(roc_auc, pr_auc, f_meas, bal_accuracy, accuracy)


# Tuning
nnet_fit_tune <- nnet_wflow_tune %>%
  tune_grid(resamples = data_folds,
            grid = grid_regular(hidden_units(range = c(10L, 30L)),
                                penalty(), levels = 3),
            metrics = class_and_probs_metrics,
            control = control_grid(verbose = TRUE)
            )
```

```
autoplot(nnet_fit_tune)
```



Come detto precedentemente, a prescindere dal numero di neuroni e dalla quantità di regolarizzazione, tutti

i modelli raggiungono performance superiori al 95% di accuratezza ma questo è principalmente dovuto allo sbilanciamento tra le classi.

La *balanced accuracy* va a ridurre questo sbilanciamento perché va a combinare il *TNR* (Specificity), cioè le corrette previsioni sulla classe dei negativi, con il *TPR* (Sensitivity), cioè le corrette previsioni sulla classe dei positivi: dati che questi ultimi sono molto meno numerosi, il valore di questo indice tenderà ad essere basso.

Simili risultati sono ottenuti dall'F_1-*score* che non riesce ad assumere valori maggiori di 0.35.

Ancora, l'area sotto la curva *Precision-Recall* è molto bassa, addirittura minore di 0.5. Questo significa che potremmo ottenere performance migliori (in termini di "prevedere bene la classe dei positivi") se andassimo ad assegnare il 50% delle osservazioni a ciascuna delle due classi.

Invece, per quanto riguarda l'area sotto la curva *ROC*, questa sembra indicare ad alcune configurazioni molto buone. Tuttavia, questo è dovuto al fatto che è costruita a partire dal *FPR*, misura che non considera affatto i positivi.

Le migliori combinazioni di iperparametri per ciascuna delle misure di performance sono riportate nella seguente tabella.

```
best_accuracy <- select_best(nnet_fit_tune, metric = "accuracy")
best_bal_accuracy <- select_best(nnet_fit_tune, metric = "bal_accuracy")
best_f_meas <- select_best(nnet_fit_tune, metric = "f_meas")
best_pr_auc <- select_best(nnet_fit_tune, metric = "pr_auc")
best_roc_auc <- select_best(nnet_fit_tune, metric = "roc_auc")

best_models <- rbind.data.frame(best_accuracy,
                                best_bal_accuracy,
                                best_f_meas,
                                best_pr_auc,
                                best_roc_auc)

best_models <- cbind.data.frame(perf_meas = c("best_accuracy", "best_bal_accuracy", "best_f_meas",
                                              "best_pr_auc", "best_roc_auc"),
                                best_models)
best_models
```

```
##           perf_meas hidden_units penalty              .config
## 1     best_accuracy           10   1e+00 Preprocessor1_Model7
## 2 best_bal_accuracy           30   1e-05 Preprocessor1_Model6
## 3       best_f_meas           30   1e-05 Preprocessor1_Model6
## 4       best_pr_auc           10   1e+00 Preprocessor1_Model7
## 5      best_roc_auc           30   1e+00 Preprocessor1_Model9
```

In termini di F_1-*score* e *PR_AUC*, misure che abbiamo visto essere le più rilevanti, i modelli migliori sono diversi.

```
collect_metrics(nnet_fit_tune, summarize = TRUE) %>%
  filter(.config %in% c("Preprocessor1_Model6", "Preprocessor1_Model7")) %>%
  filter(.metric %in% c("f_meas", "pr_auc"))
```

```
## # A tibble: 4 x 8
##   hidden_units penalty .metric .estimator  mean     n std_err .config
##          <int>   <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
```

```
## 1           30 0.00001 f_meas  binary     0.330    10  0.0296 Preprocessor1_Mod~
## 2           30 0.00001 pr_auc  binary     0.349    10  0.0422 Preprocessor1_Mod~
## 3           10 1       f_meas  binary     0.190    10  0.0312 Preprocessor1_Mod~
## 4           10 1       pr_auc  binary     0.391    10  0.0482 Preprocessor1_Mod~
```

A questo punto vado a confrontare entrambi i modelli sul test set per vedere se l'overfitting è stato evitato con successo.

```
# Finalizzo separatamente i modelli
best_net_f_meas <- finalize_model(nnet_spec_tune, parameters = best_f_meas)
best_net_pr_auc <- finalize_model(nnet_spec_tune, parameters = best_pr_auc)


# Finalizzo separatamente i workflow
best_workflow_f_meas <- finalize_workflow(x = nnet_wflow_tune, parameters = best_f_meas)
best_workflow_pr_auc <- finalize_workflow(x = nnet_wflow_tune, parameters = best_pr_auc)


# Fitting delle reti
best_fit_f_meas <- best_workflow_f_meas %>%
  fit(dataTrain)

best_fit_pr_auc <- best_workflow_pr_auc %>%
  fit(dataTrain)
```

Testing

```
## Modello:
#   - hidden_neurons = 30
#   - penalty = 1e-05
## Dati: originali

# Previsione sul test set
yPred_f_meas <- predict(best_fit_f_meas, new_data = dataTest)

# Matrice di confusione
confusionMatrix(reference = dataTest$Y,
                data = yPred_f_meas$.pred_class,
                mode = "sens_spec")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    1    0
##          1   15   15
##          0   31 1644
##
##                Accuracy : 0.973
##                  95% CI : (0.9642, 0.9802)
##     No Information Rate : 0.973
##     P-Value [Acc > NIR] : 0.53911
```

```
##
##                  Kappa : 0.3816
##
##  Mcnemar's Test P-Value : 0.02699
##
##             Sensitivity : 0.326087
##             Specificity : 0.990958
##          Pos Pred Value : 0.500000
##          Neg Pred Value : 0.981493
##              Prevalence : 0.026979
##          Detection Rate : 0.008798
##    Detection Prevalence : 0.017595
##       Balanced Accuracy : 0.658523
##
##        'Positive' Class : 1
##
```

```
## Modello:
#   - hidden_neurons = 10
#   - penalty = 1
## Dati: originali

yPred_pr_auc <- predict(best_fit_pr_auc, new_data = dataTest)

confusionMatrix(reference = dataTest$Y,
                data = yPred_pr_auc$.pred_class,
                mode = "sens_spec")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    1    0
##          1    3    2
##          0   43 1657
##
##                Accuracy : 0.9736
##                  95% CI : (0.9648, 0.9807)
##     No Information Rate : 0.973
##     P-Value [Acc > NIR] : 0.4796
##
##                  Kappa : 0.113
##
##  Mcnemar's Test P-Value : 2.479e-09
##
##             Sensitivity : 0.065217
##             Specificity : 0.998794
##          Pos Pred Value : 0.600000
##          Neg Pred Value : 0.974706
##              Prevalence : 0.026979
##          Detection Rate : 0.001760
##    Detection Prevalence : 0.002933
##       Balanced Accuracy : 0.532006
##
##        'Positive' Class : 1
```

##

A giudicare dalle matrici di confusione, il primo modello, ovvero quello con $hidden\_units = 30$ e $penalty = 1e-05$ risulta essere leggermente migliore dal punto di vista della capacità di prevedere bene i positivi. Tuttavia, anche il miglior modello tra i due produce pessimi risultati a causa del forte sbilanciamento tra le classi, riuscendo a classificare bene solo 15 dei 46 positivi.

Per compensare l'effetto dello sbilanciamento tra classi, introduco la tecnica SMOTE.

SMOTE: Synthetic Minority Oversampling TEchnique

Oversampling e *undersampling* sono alcune delle numerose possibili tecniche utilizzabili per regolare la distribuzione delle classi in un campione. Considerato un fenomeno in cui il 20% delle osservazioni appartengono alla classe dei positivi e l'80% a quella dei negativi:

- condurre oversampling significherebbe campionare con reinserimento dalla classe dei positivi quattro volte di più, fino ad ottenere un campione in cui entrambi le classi sono ugualmente rappresentate;
- condurre undersampling invece significherebbe campionare la classe dei negativi un quarto delle volte, ottenendo un campione in cui le classi hanno la stessa numerosità.

Tipicamente, tra questi due metodi, l'oversampling è quello preferibile dal momento che con l'undersampling si va a scartare informazione campionaria che potrebbe essere utile ad ottenere modelli di migliore qualità. Un punto a favore dell'undersampling potrebbero essere i minori costi pratici e relativi al campionamento ma questi si sono rivelati triviali con le nuove tencologie dell'era dei Big Data.

Oversampling e undersampling si pongono ai due estremi delle tecniche di campionamento utilizzate nei problemi con classi sbilanciate ma, tra di esse, esistono in letteratura un gran numero di alternative che vanno a compensare alcune delle loro rispettive mancanze. *SMOTE* (*Synthetic Minority Oversampling Technique*) è la più comune di queste in cui, invece di fare oversampling con reinserimento, questo è condotto attraverso la creazione di osservazioni "artificiali". La tecnica SMOTE è presentata con la possibilità di essere affiancata ad un undersampling della majority class, scartando osservazioni da essa finché la minority class non rappresenta una certa percentuale della precedente. Si dimostra che, così facendo, l'iniziale distorsione del metodo di learning verso la classe più numerosa cambia di direzione a favore di quella meno numerosa, ottenendo così migliori risultati rispetto al semplice undersampling o all'oversampling con reinserimento. Il motivo che spiega ciò risulta essere abbastanza intuitivo: ricampionando la stessa osservazione più volte, la regione attorno ad essa diventa molto "densa" e piccola all'aumentare delle ripetizioni perché concentrata in un solo punto, rimpicciolendo lo spazio attribuito alla minority class provocando così l'effetto opposto a quello desiderato; al contrario, generando osservazioni artificiali attorno ad essa, tale regione viene ingrandita, migliorando le capacità della regola decisionale di trattare la minority class.

Questa tecnica è implementata attraverso la funzione smote() del pacchetto "performanceEstimation". I suoi parametri permettono di controllare la quantità di oversampling e undersampling da applicare:

- *perc.over*: specifica quante nuove osservazioni generare per ognuna delle osservazioni nella minority class;
- k: specifica quante osservazioni usare per la generazione di ognuna delle nuove osservazioni. Ad esempio, se con $perc.over = 10$ abbiamo stabilito di creare 10 osservazioni artificiali a partire da ogni singola osservazione x_i della minority class. Con $k = 2$ stiamo dicendo che ognuna di queste 10 osservazioni artificiali deve essere creata a partire dai 2-nearest neighbours di x_i;
- *perc.under*: controlla la proporzione di osservazioni della majority class che verranno inserite nel dataset finale. Ad esempio, se sono state generate 200 osservazioni per la minority class, $perc.under = 1$ inserirà nel dataset finale esattamente 200 osservazioni casuali della majority class del dataset di partenza.

Attualmente la distribuzione delle class labels presenta solo un 3% come appartenenti alla classe dei positivi. Attraverso la tecnica SMOTE vado a fare un oversampling fino ad avere una prevalenza almeno del 20%.

```
set.seed(1763)
data_smote <- smote(Y ~ ., data = data, perc.over = 10, k = 2, perc.under = 3)
table(data_smote$Y)
```

```
##
##    1    0
## 2420 6600
```

Nel dataset originale avevamo 220 osservazioni nella minority class. Con *perc.over* = 10 andiamo a creare 10 osservazioni artificiali per ognuna di esse, per un totale di $2200 + 220 = 2420$ osservazioni. Per ognuna delle 2200 osservazioni generate, ne vengono selezionate (con reinserimento) *perc.under* = 3 del dataset originale, per un totale di 6600 osservazioni.

Pre-processing

Costruito il nuovo dataset, vado a separare i dati in train set e test set e a costruirmi i folds per la cross-validation.

```
set.seed(7165)
splittedData_smote <- rsample::initial_split(data_smote, prop = 3/4)
dataTrain_smote <- rsample::training(splittedData_smote)
dataTest_smote <- rsample::testing(splittedData_smote)


# Splitting in train e test set
yTrain_smote <- dataTrain_smote %>%
  select(Y)

xTrain_smote <- dataTrain_smote %>%
  select(-Y)

yTest_smote <- dataTest_smote %>%
  select(Y)

xTest_smote <- dataTest_smote %>%
  select(-Y)


# Creazione folds per cross-validation
set.seed(7327)
data_folds_smote <- vfold_cv(dataTrain_smote, v = 10)
```

Training modello semplice

Fatto ciò, provo a riaddestrare il modello semplice, usato precedentemente, suoi nuovi dati, in modo da vedere l'impatto della tecnica SMOTE sulle sue capacità previsive.

```
nnet_spec_smote <- mlp(epochs = 50,
                       hidden_units = 5,
                       penalty = 0) %>%
  set_mode(mode = "classification") %>%
  set_engine("nnet")
```

```
nnet_fit_smote <- nnet_spec_smote %>%
  fit(Y ~ ., dataTrain_smote)
```

Testing modello semplice su dataset SMOTE

```
yPred_smote <- predict(nnet_fit_smote, new_data = xTest_smote)

confusionMatrix(reference = yTest_smote$Y,
                data = yPred_smote$.pred_class, mode = "sens_spec")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    1    0
##          1  546  118
##          0   52 1539
##
##                Accuracy : 0.9246
##                  95% CI : (0.9129, 0.9352)
##     No Information Rate : 0.7348
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8132
##
##  Mcnemar's Test P-Value : 6.187e-07
##
##             Sensitivity : 0.9130
##             Specificity : 0.9288
##          Pos Pred Value : 0.8223
##          Neg Pred Value : 0.9673
##              Prevalence : 0.2652
##          Detection Rate : 0.2421
##    Detection Prevalence : 0.2945
##       Balanced Accuracy : 0.9209
##
##        'Positive' Class : 1
##
```

È facile vedere come l'introduzione della tecnica SMOTE abbia permesso di migliorare le capacità previsive del modello semplice di una notevole quantità. Infatti, come è possibile vedere dalla matrice di confusione e dalle relative metriche di performance, il modello addestrato sul nuovo dataset presenta valori migliori per tutte le metriche elencate, in particolare:

- raggiunge una Precision (PPV) pari a 0.82, rispetto alla precedente 0.65;
- una balanced accuracy di 0.92 rispetto al precedente 0.66;
- una Sensitivity notevolmente migliore, passando da 0.32 a 0.91.

Testing modello semplice su dataset originale

Andando ad utilizzare il modello appena addestrato per prevedere sui dati sbilanciati, è possibile vedere come queste performance vengano mantenute.

```
yPred_smote_original <- predict(nnet_fit_smote, new_data = xTest)

confusionMatrix(reference = yTest$Y,
                data = yPred_smote_original$.pred_class, mode = "sens_spec")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    1    0
##          1   39  129
##          0    7 1530
##
##                Accuracy : 0.9202
##                  95% CI : (0.9063, 0.9327)
##     No Information Rate : 0.973
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.3364
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.84783
##             Specificity : 0.92224
##          Pos Pred Value : 0.23214
##          Neg Pred Value : 0.99545
##              Prevalence : 0.02698
##          Detection Rate : 0.02287
##    Detection Prevalence : 0.09853
##       Balanced Accuracy : 0.88503
##
##        'Positive' Class : 1
##
```

La Precision (PPV) appare molto bassa semplicemente per il divario che c'è tra le due classi. Tuttavia, come è possibile vedere dalle altre misure (Sensitivity, balanced accuracy), la capacità previsiva è migliorata.

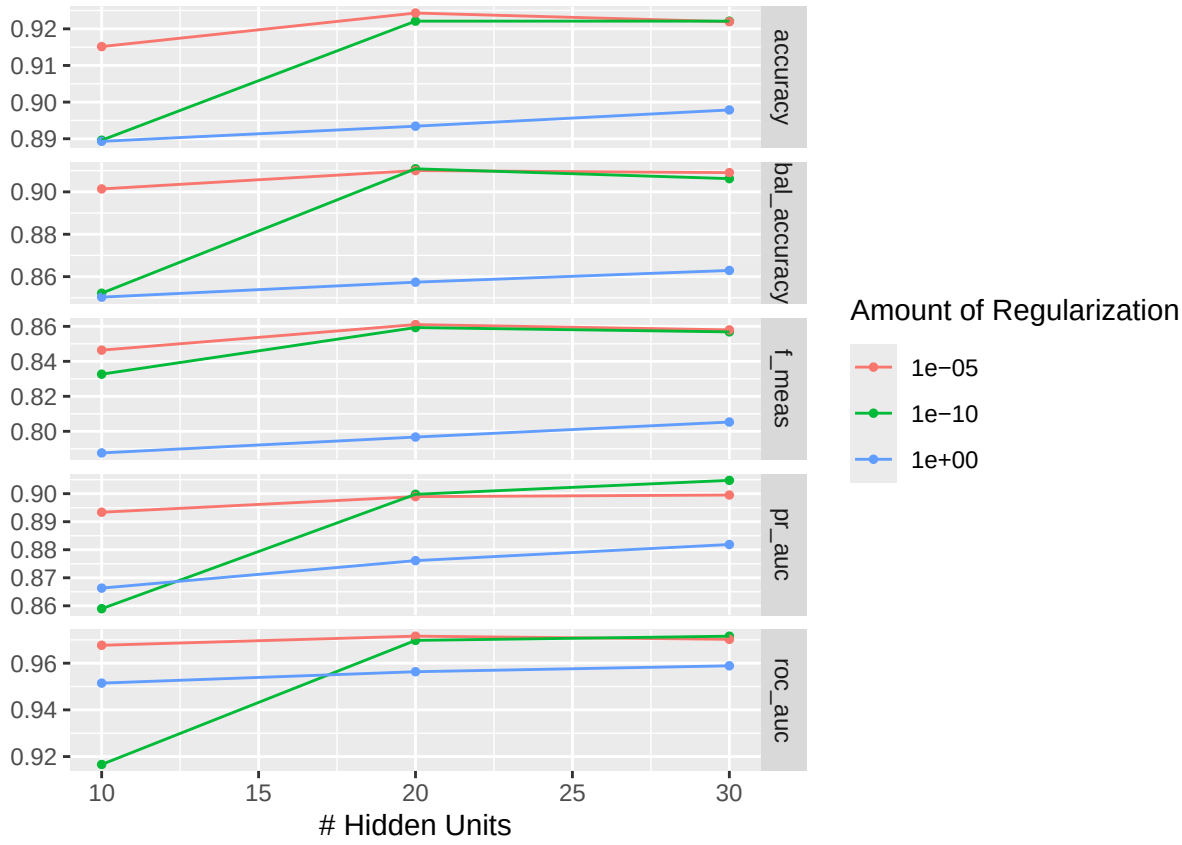
Confronto tra modelli

Per verificare se è possibile ottenere risultati ancora migliori sul nuovo dataset costruito con la tecnica SMOTE, vado a ripetere il tuning condotto precedentemente per la ricerca dei valori ottimali per il numero di neuroni nell'hidden layer e per il parametro di penalizzazione.

```
# Salva le previsioni ed i risultati intermedi del workflow
keep_pred <- control_resamples(save_pred = TRUE, save_workflow = TRUE)

# Impostazioni per l'addestramento
nnet_spec_tune_smote <- mlp(epochs = 50,
```

```
                            hidden_units = tune(),
                            penalty = tune()) %>%
  set_mode(mode = "classification") %>%
  set_engine("nnet", MaxNWts = 80000)


# Recipe
nnet_rec_tune_smote <- recipe(Y ~ ., data = dataTrain_smote)


# Workflow
nnet_wflow_tune_smote <- workflow() %>%
  add_model(nnet_spec_tune_smote) %>%
  add_recipe(nnet_rec_tune_smote)


# Metriche di performance
class_and_probs_metrics <- metric_set(roc_auc, pr_auc, f_meas, bal_accuracy, accuracy)


# Tuning
nnet_fit_tune_smote <- nnet_wflow_tune_smote %>%
  tune_grid(resamples = data_folds_smote,
            grid = grid_regular(hidden_units(range = c(10L, 30L)),
                                penalty(), levels = 3),
            metrics = class_and_probs_metrics,
            control = control_grid(verbose = TRUE)
            )
```

```
autoplot(nnet_fit_tune_smote)
```



Come è possibile vedere dal plot delle metriche di performance, i modelli con $penalty = 1e-05$ sono quelli che si comportano meglio, ma presentano un leggero disaccordo sul numero di neuroni in termini di F_1-*score* e di *pr_auc*. Come prima, vado a salvare entrambi i modelli e confrontare le loro previsioni sul test set.

```
best_f_meas_smote <- select_best(nnet_fit_tune_smote, metric = "f_meas")
best_pr_auc_smote <- select_best(nnet_fit_tune_smote, metric = "pr_auc")


# Finalizzo separatamente i modelli
best_net_f_meas_smote <- finalize_model(nnet_spec_tune_smote, parameters = best_f_meas_smote)
best_net_pr_auc_smote <- finalize_model(nnet_spec_tune_smote, parameters = best_pr_auc_smote)


# Finalizzo separatamente i workflow
best_workflow_f_meas_smote <- finalize_workflow(x = nnet_wflow_tune_smote, parameters = best_f_meas_smo
best_workflow_pr_auc_smote <- finalize_workflow(x = nnet_wflow_tune_smote, parameters = best_pr_auc_smo


# Fitting delle reti
best_fit_f_meas_smote <- best_workflow_f_meas_smote %>%
  fit(dataTrain)

best_fit_pr_auc_smote <- best_workflow_pr_auc_smote %>%
  fit(dataTrain)
```

Testing modelli su dataset SMOTE

```
## Modello:
#   - hidden_neurons = 20
#   - penalty = 1e-05
## Dati: smote

yPred_f_meas_smote <- predict(best_fit_f_meas_smote, new_data = xTest_smote)

confusionMatrix(reference = yTest_smote$Y,
                data = yPred_f_meas_smote$.pred_class, mode = "sens_spec")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    1    0
##          1  214   12
##          0  384 1645
##
##                Accuracy : 0.8244
##                  95% CI : (0.808, 0.8399)
##     No Information Rate : 0.7348
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4376
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.3579
##             Specificity : 0.9928
##          Pos Pred Value : 0.9469
##          Neg Pred Value : 0.8107
##              Prevalence : 0.2652
##          Detection Rate : 0.0949
##    Detection Prevalence : 0.1002
##       Balanced Accuracy : 0.6753
##
##        'Positive' Class : 1
##
```

```
## Modello:
#   - hidden_neurons = 30
#   - penalty = 1e-10
## Dati: smote

yPred_pr_auc_smote <- predict(best_fit_pr_auc_smote, new_data = xTest_smote)

confusionMatrix(reference = yTest_smote$Y,
                data = yPred_pr_auc_smote$.pred_class, mode = "sens_spec")
```

```
## Confusion Matrix and Statistics
##
```

```
##           Reference
## Prediction    1    0
##          1  247   12
##          0  351 1645
##
##                Accuracy : 0.839
##                  95% CI : (0.8232, 0.854)
##     No Information Rate : 0.7348
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4956
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.4130
##             Specificity : 0.9928
##          Pos Pred Value : 0.9537
##          Neg Pred Value : 0.8241
##              Prevalence : 0.2652
##          Detection Rate : 0.1095
##    Detection Prevalence : 0.1149
##       Balanced Accuracy : 0.7029
##
##        'Positive' Class : 1
##
```

Testing modelli su dataset originale

```
## Modello:
#   - hidden_neurons = 20
#   - penalty = 1e-05
## Dati: originali

yPred_f_meas_smote_original <- predict(best_fit_f_meas_smote, new_data = xTest)

confusionMatrix(reference = yTest$Y,
                data = yPred_f_meas_smote_original$.pred_class, mode = "sens_spec")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    1    0
##          1   10   15
##          0   36 1644
##
##                Accuracy : 0.9701
##                  95% CI : (0.9609, 0.9776)
##     No Information Rate : 0.973
##     P-Value [Acc > NIR] : 0.796749
##
##                   Kappa : 0.2678
##
```

```
##  Mcnemar's Test P-Value : 0.005101
##
##             Sensitivity : 0.217391
##             Specificity : 0.990958
##          Pos Pred Value : 0.400000
##          Neg Pred Value : 0.978571
##              Prevalence : 0.026979
##          Detection Rate : 0.005865
##    Detection Prevalence : 0.014663
##       Balanced Accuracy : 0.604175
##
##        'Positive' Class : 1
##
```

```
## Modello:
#   - hidden_neurons = 20
#   - penalty = 1e-05
## Dati: originali

yPred_pr_auc_smote_original <- predict(best_fit_pr_auc_smote, new_data = xTest)

confusionMatrix(reference = yTest$Y,
                data = yPred_pr_auc_smote_original$.pred_class, mode = "sens_spec")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    1    0
##          1   10   17
##          0   36 1642
##
##                Accuracy : 0.9689
##                  95% CI : (0.9595, 0.9766)
##     No Information Rate : 0.973
##     P-Value [Acc > NIR] : 0.86785
##
##                   Kappa : 0.2592
##
##  Mcnemar's Test P-Value : 0.01342
##
##             Sensitivity : 0.217391
##             Specificity : 0.989753
##          Pos Pred Value : 0.370370
##          Neg Pred Value : 0.978546
##              Prevalence : 0.026979
##          Detection Rate : 0.005865
##    Detection Prevalence : 0.015836
##       Balanced Accuracy : 0.603572
##
##        'Positive' Class : 1
##
```

I due modelli appena valutati sembrano presentare performance peggiori rispetto al modello semplice addestrato sui dataset smote. Questo potrebbe essere un caso evidente di come la maggiore complessità introdotta

dagli iperparametri *hidden_units* e *penalty*, in aggiunta alla maggiore informazione prodotta con la tecnica SMOTE, abbiano portato il modello ad adattarsi eccessivamente ai dati di training, perdendo così la sua capacità di generalizzare su dati unseen.

Conclusioni

Il progetto ha avuto come obiettivo l'esplorazione delle reti neurali applicate ad un caso di classificazione con classi fortemente sbilanciate. L'addestramento di reti caratterizzate da gradi variabili di coplessità usando il dataset originale ha prodotto risultati scadenti, fortemente influenzati dalla disparità tra le classi. L'implementazione della tecnica SMOTE ha permesso di mitigare questo effetto. Operativamente, in seguito alla creazione del dataset artificiale, le stesse reti sono state addestrate e le loro performance sono state valutate sia sul test set costruito a partire dal dataset artificiale, sia su quello costruito a partire dal dataset originale.

Per quanto riguarda il test set artificiale, i modelli più complessi riuscivano ad ottenere buone performance su di esso. Tuttavia, questa elevata complessità li ha portati ad adattarsi troppo ai dati e, di conseguenza, a produrre scarse performance sul test set originale in cui non disponevano di tutta quell'informazione artificiale generata per classe dei positivi. Infatti, il modello che si è comportato meglio sul test set originale è proprio quello più semplice, con $hidden\_units = 5$ e $penalty = 0$, che è stato quindi in grado di apprendere abbastanza bene sul dataset artificiale e a mantenere questo livello di performance anche sul test set originale dal contenuto informativo più ridotto.