Progetto Machine Learning

Damiano Caputo | Matricola: 27469/410

1 Marzo, 2022

Sommario

Piano di Lavoro

Il progetto consiste, dopo un'attenta etichettatura del dataset, nell'affermare quale tra gli algoritmi di classificazione Random Forest, Support Vector Machine, Naive Bayes e Decision Tree sia il migliore in entrambi i linguaggi. La struttura del progetto che seguiremo sarà andare a vedere per ogni algoritmo i risultati della classificazione, ottenuti da entrambi i linguaggi, sia per il dataset sbilanciato che per il bilanciato.

1 Introduzione

Nella cartella .zip sono presenti:

1. Il pdf, realizzato con il linguaggio latex, della relazione del progetto
2. Il file Etichettatura.R, contenente l'etichettatura manuale
3. Il file Pulizia_e_Algoritmi.R, contenente l'esecuzione degli algoritmi e la pulizia del dataset
4. Il file prog_AIML.ipynb, contenente la pulizia e gli algoritmi.

1.1 Ambienti di Lavoro

- RStudio, versione 4.1.2
- Visual Studio Code versione, versione 1.65.2
- Overleaf per la stesura della relazione nel linguaggio latex (`https://it.overleaf.com`)

1.2 Descrizione dei Dati

- I file che sono stati consegnati sono 4 ed essi sono stati creati estraendo delle features da twitter, infatti contengono tutti le stesse variabili. Che sono:
  - **...1**: indentatura delle righe
  - **text**: contiene il testo presente nel tweet
  - **favorited**: se al tweet è stato messo un like
  - **favoritedCount**: quanti like ha preso il commento
  - **replyToSN**: da chi è stato replicato/risposto
  - **created**: data e ora della pubblicazione
  - **truncated**: se il tweet quando è stato estratto è stato troncato ovvero che ha perso una parte di testo e al posto di quella è stato inserito un link per andare sulla pagina di twitter e controllare il testo
  - **id**: id del profilo che ha pubblicato il tweet
  - **statusSource**: percorso della sorgente
  - **retweetCount**: quante volte è stato replicato
  - **isRetweet**: se è stato replicato
  - **retweeted**: replica della colonna isRetweet
  - **longitude**: longitudine
  - **latitude**: latitudine
- Sono stati uniti tutti e 4 insieme sia su python che su R:
  - Python: sono stati importati attraverso la funzione di pandas pd.read_csv() e pd.merge() per unirli in uno solo attraverso la creazione di 2 subset contenti rispettivamente 2 file, di conseguenza essi sono stati uniti insieme e poi il dataset è stato salvato.
  - R: attraverso la libreria readr sono stati importati su R e poi uniti attraverso la funzione rbind() e conseguentemente salvati.

1.3 Pulizia dei Dati

La pulizia dei dati nella colonna text nel dataset è stata eseguita secondo le tecniche di Text Mining viste durante lo scorso anno. E' stata eseguita sia in R che in Python.

1.3.1 R

- Prima di cominciare eseguiamo un controllo, tramite un grafico, che ci consente di capire se il nostro dataset sia sbilanciato o meno. Notiamo che è fortemente sbilanciato.

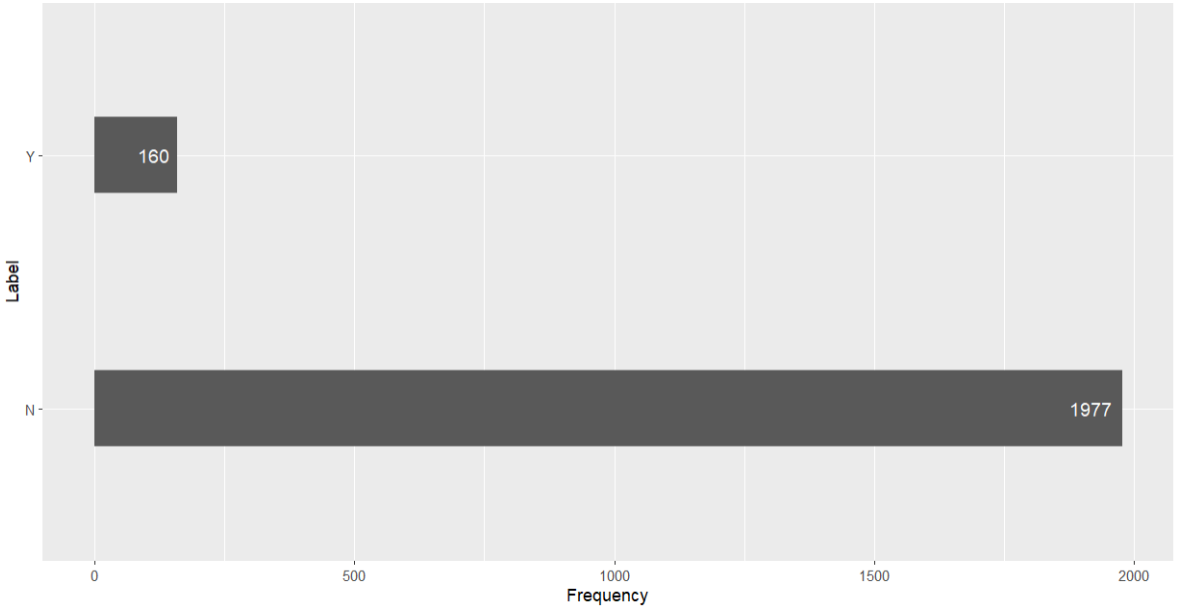


Figure 1: Control Label

- Iniziamo con la pulizia la colonna dai evenutuali hashtag "#" e tag "@" per integrarli come parole per la ricerca delle intenzioni, in quanto possono comunicare la voglia di viaggiare verso l'Italia. Inoltre usando la funzione gsub() in unione ai metdi regex eliminiamo i link presenti nei tweet ed eventuali emoji.

- In seguito si sono create due funzioni che permettono la pulizia generale della colonna. Con:

  1. tryTolower(): rendiamo il testo totalmente minuscolo, cercando di controllare gli errori che possono accadere;
  2. corpus(): una volta reso tutto minuscolo procediamo, attraverso la libreria tm, con: la rimozione delle stopwords presenti nel testo, rimozione della punteggiatura, rimozione degli spazi, rimozione di eventuali numeri e infine lo stemming delle parole presenti.

- Ottenendo il corpus, contente il testo dei tweet, pulito realizziamo il dataset per l'apprendimento e per il testing, il risultato che otteniamo è il seguente:

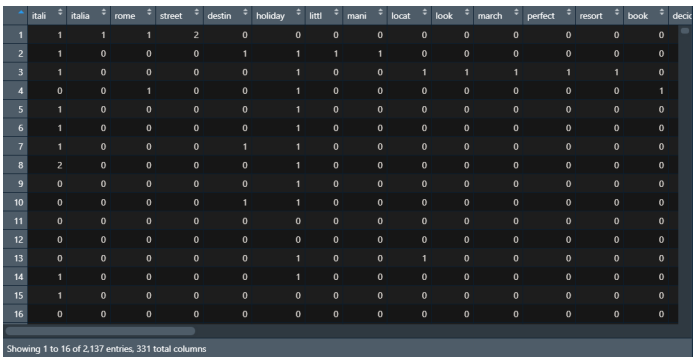


Figure 2: Dataset

L'identatura delle righe rappresenta il tweet mentre l'intestazione delle colonne sono le parole presenti nei tweet. All'interno troviamo quindi, sotto forma numerica, le volte in cui appare una determinata parola in quel tweet. In questo modo permettiamo una corretta esecuzione dei modelli degli algoritmi che vedremo in seguito.

- Per concludere la parte della pulizia e creazione del dataframe ho prodotto un grafico contenente la frequenza delle parole:

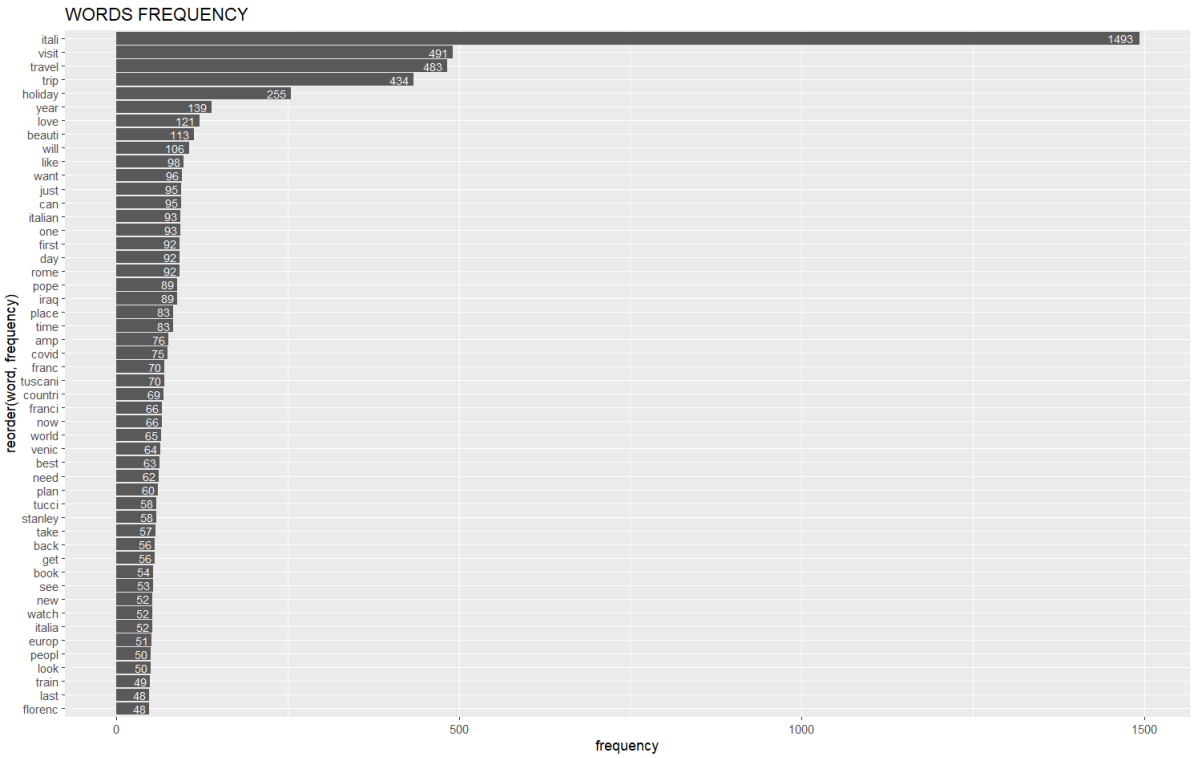


Figure 3: Word Frequency in R

Possiamo quindi affermare che la maggior parte dei tweet presentano argomenti relativi all'Italia in quanto è la parola più frequente. Infine la cosa che ha attirato la mia attenzione sono le parole "tucci" e "stanley" (frequenza 58), che si riferiscono ad un programma americano incentrato sul nostro paese che è stato condotto appunto da Stanley Tucci, infatti sono presenti molti tweet relativi alla sua serie.

1.3.2 Python

- Come in R partiamo con il controllo dello sbilanciamento del dataset. Quello che otteniamo è coerente con il controllo fatto su R:

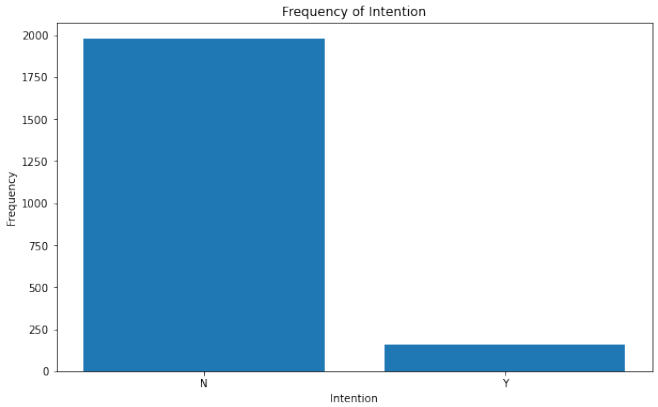


Figure 4: Dataset

- Iniziamo dunque la pulizia rendendo minuscolo il carattere dei tweet usando .str.lower(), successivamente rimuovendo i link e la rimozione delle emoji attraverso il metodo regex in unione alla funzione di pandas .replace(). Viene eseguito in questa fase in quanto quando

andremo a tokenizzare i link e i codici delle emoji si divideranno e sarà più complesso andare a rimuoverli.

- In seguito si procede con la tokenizzazione della colonna del testo, salvando il risultato in una colonna diversa dove su di essa eseguiremo i modelli. Di conseguenza attuiamo la rimozione delle stop words. Quello che otteniamo è mostrato di seguito:

```
'.@Roma__Go @Turismoromaweb @Italia @discoveroma Street Performers in #Rome, #Italy.\r\r\r\n\r\r\r\n#Streets #streetphoto<U+0085>
https://t.co/upVdb3H91s'
```

Figure 5: Senza pulizia

```
.@roma__go @turismoromaweb @italia @discoveroma street performers in #rome, #italy.

#streets #streetphoto
roma__go turismoromaweb italia discoveroma street performers rome italy streets streetphoto
```

Figure 6: Risultato della pulizia e tokenizzazione

- Infine eseguiamo lo stemming tramite la libreria nltk e attraverso essa possiamo utilizzare il metodo PorterStemmer() e di conseguenza realizziamo il grafico della frequenza delle parole.

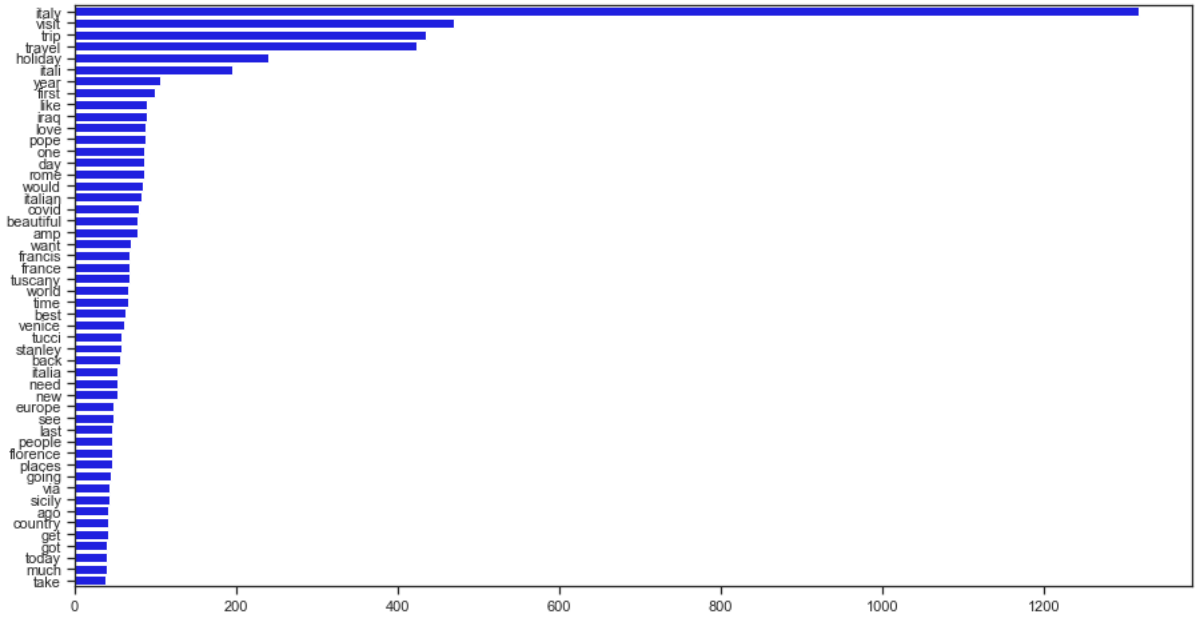


Figure 7: Word Frequency in python

Quello che possiamo notare è che il grafico coincide pressochè con quello creato su R, ma la cosa che salta all'occhio è che lo stemming in python è meno preciso del metodo utilizzato in R.

1.4 Bilanciamento del Dataset

- Data la poca quantità di istanze presenti nel dataset, come visto nella figura 1, a fine etichettatura, si è arrivati ad avere uno sbilanciamento in favore della classe N, corrispondente al concetto che il post non sia un'intenzione di venire in Italia, con un totale di 1977 istanze e le restanti 160 sono Y, ovvero intenzioni di venire in Italia.

- Per risolvere questo problema attuiamo le tecniche di bilanciamento viste a lezione. Queste tecniche si basano su due funzioni:

  1. Tecniche di Sottocampionamento: dove si procede a ridurre la classe maggioritaria, nel nostro caso N, andando ad eliminare alcune delle istanze.

2. Tecniche di Sovracampionamento: si procede inversamente al sottocampionamento in quanto non andiamo ad eliminare le istanze ma aumenteremo la classe minoritaria, nel nostro caso Y, fino ad ottenere una cospicua quantità di istanze tale da compararla con la maggioritaria.

- La tecnica scelta è il Sovracampionamento, in quanto non disponendo di una grande mole di istanze è la scelta migliore. Si applica, dopo vari tentativi, l'algoritmo SMOTE sia in python che in R. L'algoritmo funziona selezionando esempi vicini nello spazio delle caratteristiche, tracciando una linea tra gli esempi nello spazio delle caratteristiche e disegnando un nuovo campione in un punto lungo quella linea.
  - In python grazie alla libreria sklearn e imbalanced-learn, possiamo generare nuovi campioni per la classe Y senza troppa difficoltà. Applicando la funzione SMOTE() inizializziamo il comando di sovracampionamento e con fit_resample() avviamo il processo ottenendo un dataframe bilanciato per entrambe le classi.
  - In R grazie alla libreria imbalance e alla funzione mwmote() (Majority Weighted Minority Oversampling TEchnique) possiamo usare il metodo SMOTE con un miglioramento. Dato che la libreria che permette la funzione smote generava, invece che delle istanze concrete, dell'istanze basate sul rumore del dataset e mwmote evita appunto questo problema.

1.5 Etichettatura in R

- Per questioni di tempo e per una comodità personale l'etichettatura è stata eseguita su RStudio.
- Prima di eseguire 'etichettatura si sono esguiti dei piccoli controlli, tra cui quanti NA sono presenti nelle colonne e di conseguenza eliminare quelle con più missing values. Infine si è deciso di fare una piccola scrematura andando ad eliminare tutti i tweet che si ripetono nelle righe del dataset
- Tale rimozione dei duplicati non funziona al 100% perchè in alcuni tweet sono presenti link alla pagina twitter che rende unico il tweet anche se uguale ad un altro, stessa cosa vale per la presenza di "RT" posta davanti al testo rendendo anch'esso unico il post.
- Succesivamente si è creata una colonna per la variabile target, contenente le classi per la classificazione dei tweet:
  - Con N indicheremo i tweet che non contengono un'intenzione di venire in Italia
  - Con Y indicheremo i tweet che presentano l'intenzione di venire in Italia
- L'etichettatura delle righe è stata eseguita stampando a schermo ogni tweet presente e poi leggendo il testo del tweet ho deciso come etichettarlo.
- Il problema più grande riscontrato è stato capire nella lingua inglese se effettivamente si trattasse di un'intenzione o meno. Infatti per questo motivo l'espressione del desiderio di venire in Italia e il ricordo di una vancanza fatta non sono stati inseriti come Y ma come N.
- Una volta completata l'etichettatura ho salvato il dataset completo per continuare le analisi successive.

2 Random Forest

Applicando il bagging agli alberi di decisione quello che otteniamo è il random forest. Il random forest riduce la varianza nelle previsioni attraverso dataset il piu scorrelati possibile per l'ensemble averaging. Inoltre utilizza il bootstrap per ottenere dataset diversi e parzialmente correlati. Quindi i pro ed i contro sono:

- Pro: Un'elevata accuratezza
- Contro: Lentezza e bassa interpretabilità

2.1 Python

2.1.1 Sbilanciato

- Prima della creazione del modello si crea il dataset per il training e per il testing, splittando le istanze del dataframe in: ≈70-80% per il training e il 30-20% per il testing.

```
Istanze per il Training: (1709, 6322)
Istanze per il Testing: (428, 6322)
```

(a) Divisione delle istanze

```
Istanze per il testing:

0    397
1     31
```

(b) Istanze per il testing

- Successivamente ho stampato a schermo il valore dell'accuracy nel training e nel testing, ottenendo:

```
Random Forest                0.999415
```

(a) Accuracy nel Training

```
Accuracy of Testing: 0.927
```

(b) Accuracy nel Testing

Tenendo conto del fatto che il dataset non è bilanciato, possiamo notare che la percentuale dell'Accuracy diminuisca circa del 7%. Inoltre ci aspettiamo che la classe che il modello riesca a riconoscere meglio è quella relativa alla non intenzione (N).

- Creiamo quindi la matrice di confusione del modello del random forest, quello che otteniamo è:

```
True Positives(TP) =  395

True Negatives(TN) =  2

False Positives(FP) =  2

False Negatives(FN) =  29
```

	Actual Positive	Actual Negative
Predict Positive	395	2
Predict Negative	29	2

Figure 10: Matrice di Confusione

Possiamo confermare che il modello classifica meglio la classe indicata come N, infatti viene inserita nei TP per il fatto che il dataset in questo caso è sbilanciato. Inoltre la classe Y viene totalmente misclassificata in quanto la sua presenza è molto bassa.

- Creando le formule delle metriche su visual studio code possiamo confermare ancora meglio le affermazioni sopra accennate, in quanto otteniamo:

```
Other Metrics:
- TP: 395 , TN: 2 , FP: 2 , FN: 29
- Classification accuracy: 0.9276
- Classification error: 0.0724
- Precision: 0.9950
- Recall or Sensitivity: 0.9316
- True Positive Rate: 0.9316
- False Positive Rate: 0.5000
- Specificity: 0.5000
```

Figure 11: Metriche

Capiamo quindi che l'errore nel trovare le intenzioni è del 93%, mentre come già detto in precedenza l'errore relativo alla classe N è del 7%.

- Infine ho creato le curve ROC (Receiver Operating Characteristic) dei modelli utilizzati, che rappresentano le prestazioni del classificatore tanto più si avvicina alla bisettrice tanto più sarà pessimo. Con il modello del random forest con il dataset sbilanciato otteniamo:

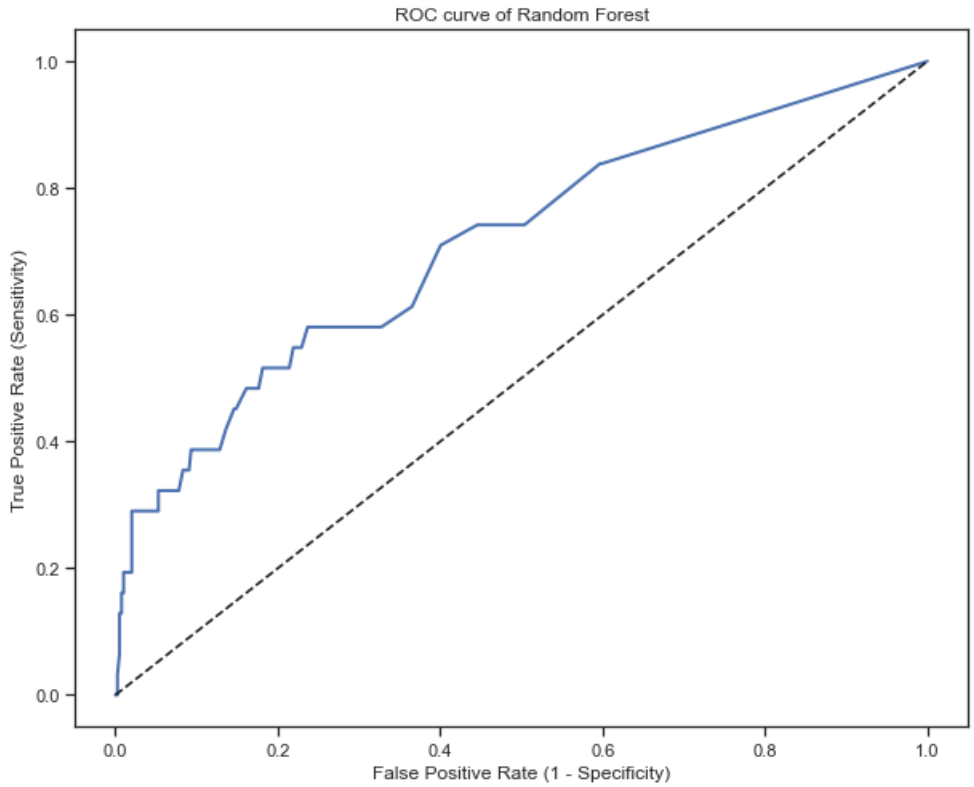


Figure 12

Ne deduciamo che come modello in questa fase non è efficientissimo.

2.1.2 Bilanciato

- Applicando la tecnica del sovracampionamento tramite l'algoritmo SMOTE, spiegato in precedenza, prepariamo il dataset per l'apprendimento e quello che otteniamo è mostrato di seguito:

```
Counter({0: 1977, 1: 160})
```

(a) Prima del Sovracampionamento

```
Counter({0: 1977, 1: 1977})
```

(b) Dopo il Sovracampionamento

In questo modo possiamo rendere più veritieri i risultati dei modelli e analizzare le effettive prestazioni. Successivamente splittiamo il dataframe per l'apprendimento e per il testing, ottenendo le seguenti istanze:

```
Istanze per il Training: (3163, 6322)
Istanze per il Testing: (791, 6322)
```

(a) Divisione delle Istanze

```
Istanze per il Testing:

0    412
1    379
```

(b) Istanze per il Testing

- Succesivamente confrontiamo l'accuracy del training con quella del testing, ottenendo:

```
Random Forest                0.999684
```

(a) Accuracy nel Training

```
Accuracy of Testing: 0.9898862199747156
```

(b) Accuracy nel Testing

Possiamo notare il netto miglioramento del modello nella fase di testing, in quanto è migliorato del ≈7% rispetto alla fase con il dataset sbilanciato. Quindi possiamo supporre che la classificazione e la ricerca della classe delle intenzioni (Y) è ottimizzata.

- Con la matrice di confusione confermiamo quanto detto, in quanto il modello creato è riuscito a classificare quasi perfettamente tutte e due le classi:

```
True Positives(TP) =  408

True Negatives(TN) =  375

False Positives(FP) =  4

False Negatives(FN) =  4
```

	Actual Positive	Actual Negative
Predict Positive	408	4
Predict Negative	4	375

Figure 16: Matrice di Confusione

Da ciò confermiamo il grande miglioramento del modello con il dataset bilanciato

- Grazie alle altre metriche possiamo renderci conto del'andamento delle prestazioni:

```
Other Metrics:
- TP: 408 , TN: 375 , FP: 4 , FN: 4
- Classification accuracy: 0.9899
- Classification error: 0.0101
- Precision: 0.9903
- Recall or Sensitivity: 0.9903
- True Positive Rate: 0.9903
- False Positive Rate: 0.0106
- Specificity: 0.9894
```

Figure 17: Metriche con DB bilanciato

Tutte le metriche sono migliorate, potendo quindi confermare quanto detto in precedenza.

- Infine si crea la curva ROC del modello:

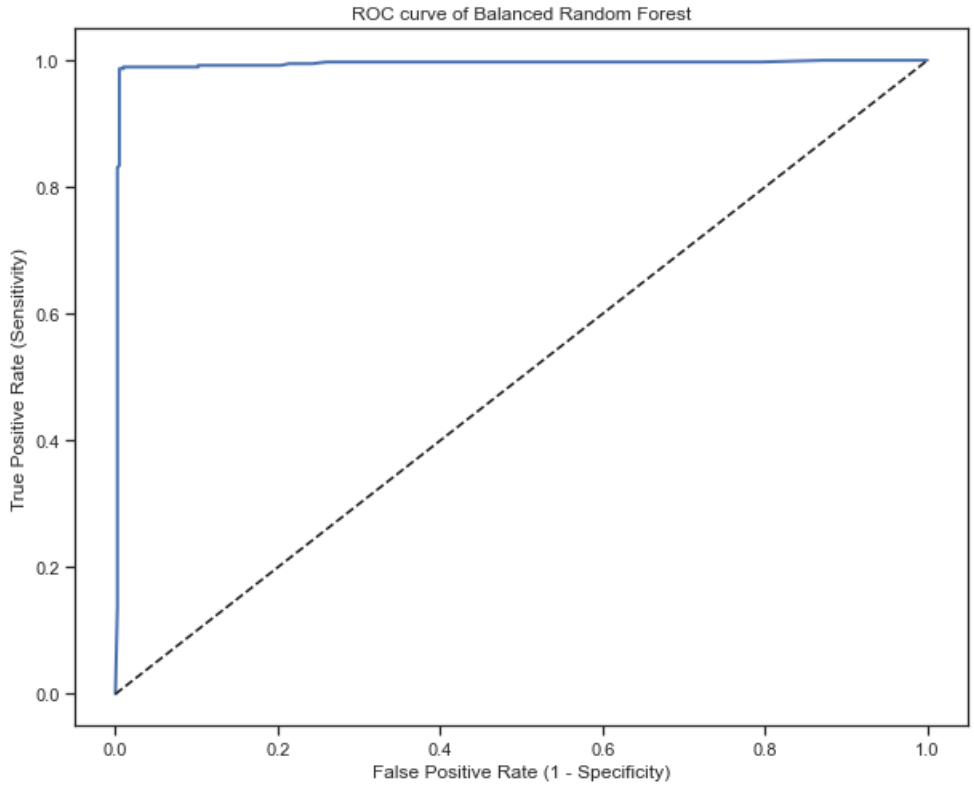


Figure 18

Il risultato è più che conforme alle assunzioni fatte in precedenza ovvero che il random forest a discapito dell'interpretabilità e del tempo di esecuzione ha ottimo risultato in termini di accuracy.

2.2 R

2.2.1 Sbilanciato

- Come prima cosa partiamo creando i rispettivi dataset, utilizzando il dataset originario sbilanciato, per l'apprendimento e per il testing. Attuiamo una divisione di ≈70%:

```
test                    641 obs. of 331 variables
```

(a) Istanze per il testing

```
train                   1496 obs. of 331 variables
```

(b) Istanze per il training

Inoltre verifichiamo le istanze etichettate nel dataframe del testing, ovviamente la presenza delle etichette N sarà superiore:

```
> length(which(test$intention == "Y"))
[1] 48
> length(which(test$intention == "N"))
[1] 593
```

Figure 20: Divisione delle Istanze del Testing

- Successivamente, attraverso la libreria randomForest, ho creato i modelli per l'esecuzione del Random Forest. Di default la funzione randomForest() realizza una foresta 500 alberi su cui eseguire il modello.

  Di conseguenza per visualizzare l'andamento delle prestazioni visualizziamo, tramite il comando confusionMatrix() della libreria caret, la matrice di confusione del modello del training e del testing, ottenendo:

```
Confusion Matrix and Statistics

          Reference
Prediction    N    Y
         N 1383   83
         Y    1   29

               Accuracy : 0.9439
                 95% CI : (0.931, 0.955)
    No Information Rate : 0.9251
    P-Value [Acc > NIR] : 0.002548

                  Kappa : 0.3891

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.25893
            Specificity : 0.99928
         Pos Pred Value : 0.96667
         Neg Pred Value : 0.94338
             Prevalence : 0.07487
         Detection Rate : 0.01939
   Detection Prevalence : 0.02005
      Balanced Accuracy : 0.62910

       'Positive' Class : Y
```

(a) Matrice di confusione del Training

```
Confusion Matrix and Statistics

          Reference
Prediction   N   Y
         N 592  46
         Y   1   2

               Accuracy : 0.9267
                 95% CI : (0.9037, 0.9456)
    No Information Rate : 0.9251
    P-Value [Acc > NIR] : 0.4786

                  Kappa : 0.0702

 Mcnemar's Test P-Value : 1.38e-10

            Sensitivity : 0.04167
            Specificity : 0.99831
         Pos Pred Value : 0.66667
         Neg Pred Value : 0.92790
             Prevalence : 0.07488
         Detection Rate : 0.00312
   Detection Prevalence : 0.00468
      Balanced Accuracy : 0.51999

       'Positive' Class : Y
```

(b) Matrice di confusione del Testing

Quello che salta all'occhio è che con l'uso di questo pacchetto la matrice ottenuta è traslata rispetto a quella normale, ovvero nelle colonne avremo i valori attuali e nelle righe le predizioni. Nel caso del dataset sbilanciato le misurazioni per la classe Y non sono ottimali. Infatti:

- per la classe N la misclassificazione è praticamente nulla.
- per la classe Y il modello misclassifica quasi totalmente le istanze.

Infine grazie alle metriche presesnti in basso nelle immagini, deduciamo che il modello non è adatto al riconoscimento della classe Y poichè la sensibilità è praticamente nulla e la specificità invece è quasi al 100%.

2.2.2 Bilanciato

- Come in python, come abbiamo detto in precedenza, per bilanciare il dataframe abbiamo eseguito il sovracampionamento tramite l'algoritmo SMOTE. In R il procedimento è leggermente diverso, in quanto dobbiamo creare un subset, che usiamo come modello di esempio delle classi da sovracampionare, per il comando mwmote() che ne creerà nel nostro caso 1850. Una volta fatto ciò, uniamo le istanze create dall'algoritmo al nuovo dataset rendendolo bilanciato. Useremo quest'ultimo per le successive analisi bilanciate in R. Il risultato che otteniamo è il seguente:

```
balanced_dta              3987 obs. of 331 variables
```

Figure 22: Dataset Bilanciato

- Divido il dataframe bilanciato per creare le istanze per il training e per il testing e successivamente controllo quante istanze Y ed N sono presenti nel testing:

```
train_3                 2791 obs. of 331 variables
```

(a) Istanze del Training

```
test_3                  1196 obs. of 331 variables
```

(b) Istanze del Testing

```
> length(which(test_3$intention == "1"))
[1] 603
> length(which(test_3$intention == "0"))
[1] 593
```

(c) Istanze classificate nel Testing

- Successivamente inizializzo il modello del Random Forest con il dataset bilanciato e realizzo le matrici di confusione per vedere l'eventuale miglioramento del modello:

```
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 1377   16
         1    7 1391

               Accuracy : 0.9918
                 95% CI : (0.9877, 0.9948)
    No Information Rate : 0.5041
    P-Value [Acc > NIR] : < 2e-16

                  Kappa : 0.9835

 Mcnemar's Test P-Value : 0.09529

            Sensitivity : 0.9886
            Specificity : 0.9949
         Pos Pred Value : 0.9950
         Neg Pred Value : 0.9885
             Prevalence : 0.5041
         Detection Rate : 0.4984
   Detection Prevalence : 0.5009
      Balanced Accuracy : 0.9918

       'Positive' Class : 1
```

(a) Matrice di Confusione nel Training

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 576  16
         1  17 587

               Accuracy : 0.9724
                 95% CI : (0.9615, 0.9809)
    No Information Rate : 0.5042
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.9448

 Mcnemar's Test P-Value : 1

            Sensitivity : 0.9735
            Specificity : 0.9713
         Pos Pred Value : 0.9719
         Neg Pred Value : 0.9730
             Prevalence : 0.5042
         Detection Rate : 0.4908
   Detection Prevalence : 0.5050
      Balanced Accuracy : 0.9724

       'Positive' Class : 1
```

(b) Matrice di Confusione nel Testing

Come possiamo notare il modello è migliorato in quanto nel testing vengono riconosciute quasi tutte le istanze, aumentando l'accuracy del 5%. Inoltre notiamo che la sensibilità, una volta bilanciato il dataframe, aumenta del 93% rendendo nettamente superiore il riconoscimento della classe Y in questo modello rispetto al modello sbilanciato.

2.3 Confronto tra Python ed R

Dall'analisi di questo algoritmo possiamo affermare che le prestazioni del modello in python è lievemente migliore del modello realizzato in R, in quanto la misclassificazione è minima e il valore dell'accuracy, quindi la ricerca dei valori positivi, anch'essa è minima. A livello di tempistiche il modello in R impiega meno tempo per l'esecuzione a discapito delle prestazioni.

Quindi per l'algoritmo Random Forest il linguaggio migliore, su cui eseguire il modello, è Python poichè è più versatile e più preciso.

3 Support Vector Machine

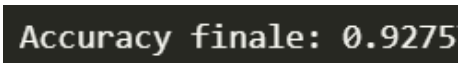
Le Support-Vector Machine o SVM sono modelli di classificazione il cui obiettivo è quello di trovare la retta di separazione delle classi che massimizza il margine tra le classi stesse, dove con margine si intende la distanza minima dalla retta ai punti delle due classi. Questo obiettivo viene raggiunto utilizzando una parte minimale del dataset di allenamento, i cosiddetti vettori di supporto che sono i valori di una classe più vicini alla retta di separazione, quelli che più si avvicinano all'altra classe. In sostanza sono i valori classificabili con maggiore difficoltà.

3.1 Python

3.1.1 Sbilanciato

- Attraverso la divisione del dataset creata in precedenza inizializziamo i modelli per l'algoritmo SVM.
- Come prima cosa stampiamo a schermo i valori dell'accuracy del modello del training e del testing:

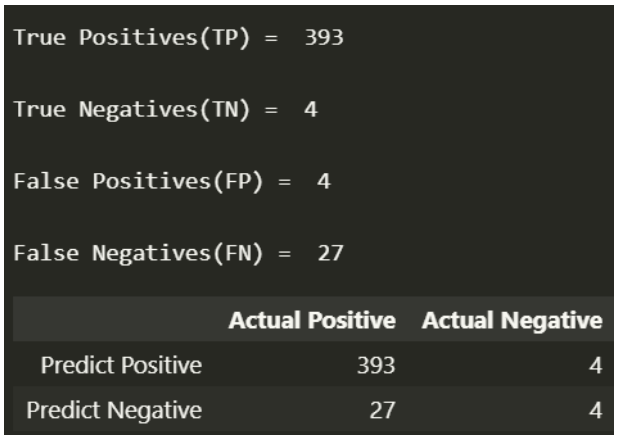
(a) Accuracy nel Training



(b) Accuracy nel Testing

Essendo il dataset sbilanciato mi aspetto che riesca a classificare meglio la classe delle N, inoltre l'accuracy si è abbassata del 5%.

- Successivamente creo la matrice di confusione del modello del SVM, il risultato è mostrato di seguito:



	Actual Positive	Actual Negative
Predict Positive	393	4
Predict Negative	27	4

Figure 26: Matrice di confusione

Possiamo confermare quanto appena detto, infatti il classificatore misclassifica quasi totalmente le classi Y.

- Per comprendere meglio le prestazioni del modello ne stampiamo le metriche:

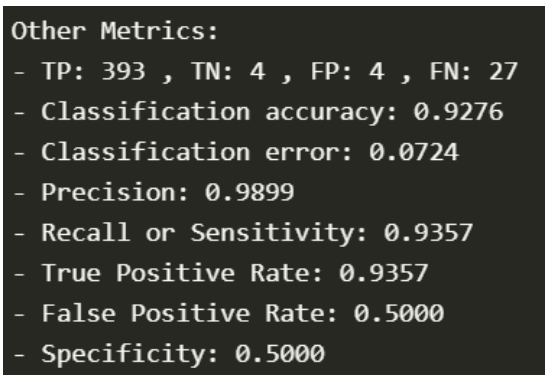


Figure 27: Metriche

Notiamo come l'errore nella classificazione delle etichette Y è del 93%.

- Infine creando la curva ROC possiamo comprendere al meglio le prestazioni dell'algoritmo

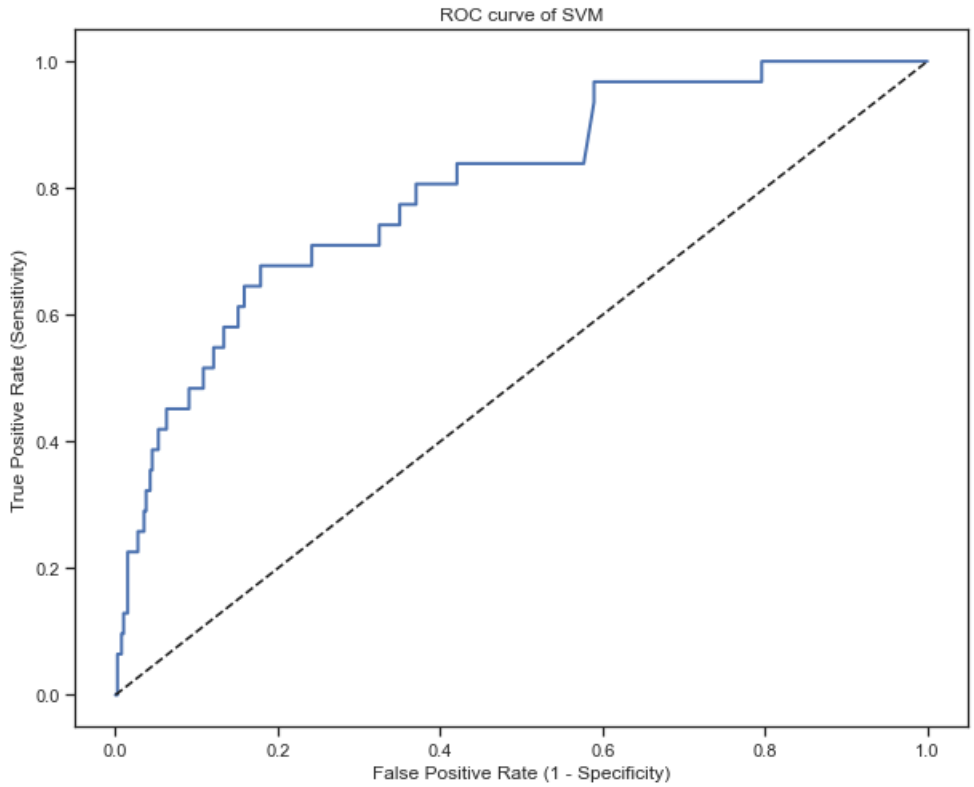


Figure 28

Il risultato ci mostra la conferma di quanto abbiamo detto in precedenza, ovvero che il modello non è molto efficiente.

3.1.2 Bilanciato

- Utilizzo lo stesso metodo e gli stessi dataframe bilanciati usati per la divisione tra apprendimento e testing nel random forest e li applico per i modelli del SVM bilanciati.
- Iniziamo controllando i valori delle accuracy:

(a) Accuracy nel Training

(b) Accuracy nel Testing

Possiamo notare il netto miglioramento del modello nella fase di testing, in quanto è migliorato del ≈6% rispetto alla fase con il dataset sbilanciato. Quindi possiamo supporre che la classificazione e la ricerca della classe delle intenzioni (Y) è migliorata. Inoltre il modello di questo algoritmo ha un'accuracy lievemente maggiore del modello del Random Forest.

- In seguito creo la matrice di confusione e le metriche di prestazione, così da poter confermare quanto detto:

```
True Positives(TP) =  401

True Negatives(TN) =  377

False Positives(FP) =  11

False Negatives(FN) =  2
```

	Actual Positive	Actual Negative
Predict Positive	401	11
Predict Negative	2	377

(a) Matrice di Confusione

```
Other Metrics:
- TP: 401 , TN: 377 , FP: 11 , FN: 2
- Classification accuracy: 0.9836
- Classification error: 0.0164
- Precision: 0.9733
- Recall or Sensitivity: 0.9950
- True Positive Rate: 0.9950
- False Positive Rate: 0.0284
- Specificity: 0.9716
```

(b) Metriche

Dal risultato che otteniamo capiamo che il modello riconosce perfettamente la classe Y, inoltre l'errore è praticamente nullo e grazie alla specificità e alla sensibilità del modello confermiamo un'ottima prestazione.

Confermata inoltre dalla curva ROC, che ci mostra le sue prestazioni:

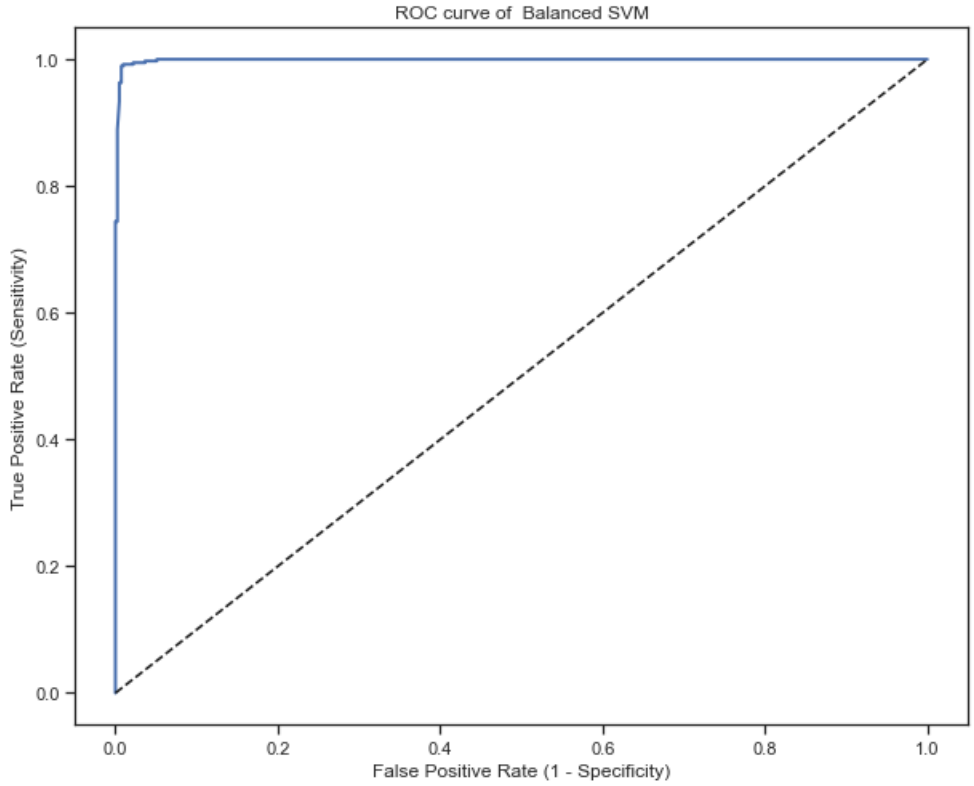


Figure 31

Possiamo notare dalla curva che il modello ottenuto, come già detto, è molto efficiente e che si discosta di poco dal Random Forest.

3.2 R

3.2.1 Sbilanciato

- Utilizzo gli stessi dataframe, figure 19a e 19b, ottenuti dalla divisione utilizzata nel Random Forest.
- Iniziamo creando il modello, attraverso la funzione svm() del pachetto e1071, e in esso inseriamo:

- La colonna in cui è contenuta l'etichettatura in confronto con tutte le altre
- Il dataframe per l'apprendimento
- Il tipo di classificazione della macchina, nel nostro caso sarà la C-classification. Essa serve per l'utilizzo della classificazione binaria.

- Procediamo con la creazione delle martrici di confusione del training e del testing, quello che otteniamo è mostrato di seguito:

```
Confusion Matrix and Statistics

          Reference
Prediction    N    Y
         N 1384   94
         Y    0   18

               Accuracy : 0.9372
                 95% CI : (0.9237, 0.9489)
    No Information Rate : 0.9251
    P-Value [Acc > NIR] : 0.04015

                  Kappa : 0.2616

 Mcnemar's Test P-Value : < 2e-16

            Sensitivity : 0.16071
            Specificity : 1.00000
         Pos Pred Value : 1.00000
         Neg Pred Value : 0.93640
             Prevalence : 0.07487
         Detection Rate : 0.01203
   Detection Prevalence : 0.01203
      Balanced Accuracy : 0.58036

       'Positive' Class : Y
```

(a) Matrice di Confusione del Training

```
Confusion Matrix and Statistics

          Reference
Prediction   N   Y
         N 592  46
         Y   1   2

               Accuracy : 0.9267
                 95% CI : (0.9037, 0.9456)
    No Information Rate : 0.9251
    P-Value [Acc > NIR] : 0.4786

                  Kappa : 0.0702

 Mcnemar's Test P-Value : 1.38e-10

            Sensitivity : 0.04167
            Specificity : 0.99831
         Pos Pred Value : 0.66667
         Neg Pred Value : 0.92790
             Prevalence : 0.07488
         Detection Rate : 0.00312
   Detection Prevalence : 0.00468
      Balanced Accuracy : 0.51999

       'Positive' Class : Y
```

(b) Matrice di Confusione del Testing

Come per il Random Forest la matrice è traslata. Notiamo come il valore dell'accuracy dal training al testing si è abbassato di poco, di conseguenza il modello riconosce totalmente la classe N mentre le Y vengono misclassificate. Viene confermato anche dalla sensibilità che ha un valore di ≈4% e la specificità a ≈100%.

3.2.2 Bilanciato

- Per eseguire l'analisi bilanciata gli stessi dataframe bilanciati, figure 23a e 23b, ottenuti dalla divisione utilizzata nel Random Forest Bilanciato.
- Inizializziamo il modello per l'algoritmo SVM, ma questa volta con il dataset bilanciato e utilizzando sempre il metodo della C-classification. Per verificare le prestazioni del nostro modello grafichiamo le matrici di confusione:

```
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 1310   35
         1   74 1372

               Accuracy : 0.9609
                 95% CI : (0.9531, 0.9678)
    No Information Rate : 0.5041
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9219

 Mcnemar's Test P-Value : 0.0002729

            Sensitivity : 0.9751
            Specificity : 0.9465
         Pos Pred Value : 0.9488
         Neg Pred Value : 0.9740
             Prevalence : 0.5041
         Detection Rate : 0.4916
   Detection Prevalence : 0.5181
      Balanced Accuracy : 0.9608

       'Positive' Class : 1
```

(a) Matrice di Confusione del Training

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 557  12
         1  36 591

               Accuracy : 0.9599
                 95% CI : (0.9471, 0.9703)
    No Information Rate : 0.5042
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9197

 Mcnemar's Test P-Value : 0.0009009

            Sensitivity : 0.9801
            Specificity : 0.9393
         Pos Pred Value : 0.9426
         Neg Pred Value : 0.9789
             Prevalence : 0.5042
         Detection Rate : 0.4941
   Detection Prevalence : 0.5242
      Balanced Accuracy : 0.9597

       'Positive' Class : 1
```

(b) Matrice di Confusione del Testing

Dalle matrici possiamo notare che l'accuracy nel testing è aumentata del ≈3%, infatti il riconoscimento della classe Y è quasi totale. La sensibilita è aumentata dell'86%, quindi possiamo dire che si comporta decentemente e lo rende un buon classificatore della classe Y.

3.3 Confronto tra Python ed R

Da questa analisi sull'algoritmo SVM possiamo affermare che le presetazioni, ottenute dai modelli creati, sono nettamente migliori nell'esecuzione di python. In quanto il modello di python restituisce:

- un'accuracy più alta del 3%,
- la sensibilità più alta di ≈10%
- una specificità più alta del 4%

Possiamo quindi dire che il linguaggio migliore su cui eseguire quest'algoritmo è Python.

4 Naive Bayes

La regola di Bayes richiede il calcolo della probabilità condizionale dell'istanza, ma essa è costituita dal verificarsi congiunto dei valori delle features; quindi, dovremmo calcolare la probabilità congiunta dei valori di quest'ultimi, ma questo calcolo può essere computazionalmente oneroso quando abbiamo molte features. L'algoritmo naive Bayes: applica la regola di Bayes ma assume indipendenza tra le features, la probabilità congiunta è quindi il prodotto delle probabilità condizionali di ogni features. Inoltre si basa sull'ipotesi di indipendenza tra le features, il training è facile e veloce e considera ogni attributo separatamente per calcolare le probabilità condizionali, il testing è immediato avendo a disposizione le tabelle delle probabilità condizionate, le prestazioni sono buone anche quando è violata l'ipotesi di indipendenza.

4.1 Python

4.1.1 Sbilanciato

- Utilizziamo gli stessi dataframe, figura 8a, ottenuti dalla divisione utilizzata nel Random Forest.
- Iniziamo stampando i valori dell'accuracy nella fase dell'apprendimento e del testing:

Naive Bayes	0.991808

(a) Accuracy nel Training

```
Accuracy finale: 0.8971
```

(b) Accuracy nel Testing

Notiamo come l'accuracy nel testing diminuisca del ≈10%, di conseguenza mi aspetto che il modello non riconosca le istanze Y e che abbia i valori delle metriche molto bassi.

- Successivamente, per confermare quanto appena detto, creiamo la matrice di confusione e le metriche delle prestazioni:

```
True Positives(TP) =  377

True Negatives(TN) =  7

False Positives(FP) =  20

False Negatives(FN) =  24
```

	Actual Positive	Actual Negative
Predict Positive	377	20
Predict Negative	24	7

(a) Matrice di Confusione

```
Other Metrics:
- TP: 377 , TN: 7 , FP: 20 , FN: 24
- Classification accuracy: 0.8972
- Classification error: 0.1028
- Precision: 0.9496
- Recall or Sensitivity: 0.9401
- True Positive Rate: 0.9401
- False Positive Rate: 0.7407
- Specificity: 0.2593
```

(b) Metriche

Vediamo come nella matrice di confusione la classe delle Y è misclassificata, inoltre a confermare quanto detto sono i valori della specificità e della sensibilità che sono molto bassi.

- Infine come ultima analisi per confermare che in questa fase il modello non è ottimale, ho realizzato la curva ROC:

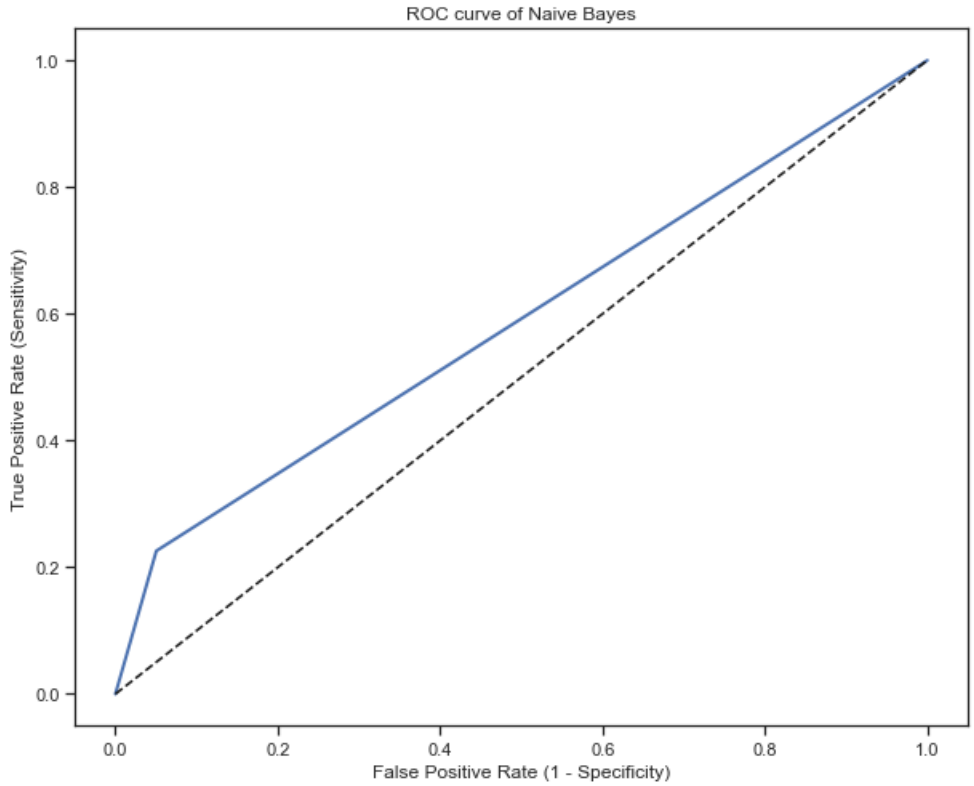


Figure 36: Matrice di Confusione

4.1.2 Bilanciato

- Utilizziamo gli stessi dataframe bilanciati, figure 14a 14b, ottenuti dalla divisione utilizzata nel Random Forest.
- Cominciamo con il verificare le accuracy del training e del testing:

(a) Accuracy del Training

(b) Accuracy del Testing

Notiamo che il modello, bilanciando il dataframe, sia migliorato dato che l'accuracy nella fase del testing è aumentata del 6%. Di conseguenza mi aspetto che la classe delle Y venga riconosciuta.

- In seguito per verificare le assunzioni sopracitate, realizziamo la matrice di confusione e calcolato le metriche:

```
True Positives(TP) =  386

True Negatives(TN) =  379

False Positives(FP) =  26

False Negatives(FN) =  0
```

	Actual Positive	Actual Negative
Predict Positive	386	26
Predict Negative	0	379

(a) Matrice di Confusione

```
Other Metrics:
- TP: 386 , TN: 379 , FP: 26 , FN: 0
- Classification accuracy: 0.9671
- Classification error: 0.0329
- Precision: 0.9369
- Recall or Sensitivity: 1.0000
- True Positive Rate: 1.0000
- False Positive Rate: 0.0642
- Specificity: 0.9358
```

(b) Metriche

Dai risultati deduciamo che il modello creato, con il dataframe bilanciato, si presta perfettamente nella ricerca delle istanze Y in quanto vengeno riconosciute totalmente. Inoltre notiamo un netto miglioramento delle metriche, infatti:

- la sensibilità è aumentata del $\approx 6\%$
- la specificità è aumentata del $\approx 20\%$

Possiamo dire che come classificatore è molto discreto, l'unica cosa in cui pecca, ed è il motivo per cui l'accuracy è bassa, è il riconoscimento delle N perchè viene misclassificata leggermente e questo non lo rende eccelso. Quest'ultima cosa la verifichiamo con la curva ROC:

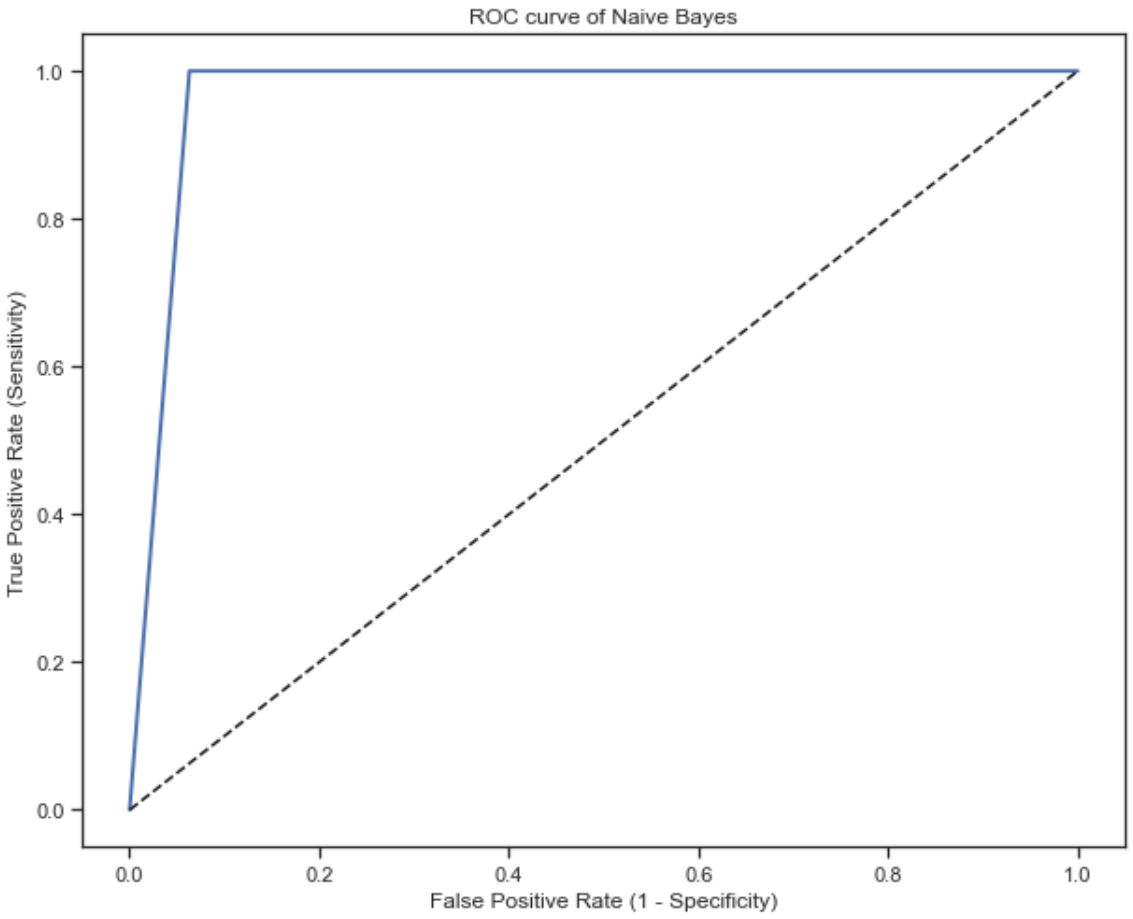


Figure 39

4.2 R

4.2.1 Sbilanciato

- Utilizziamo gli stessi dataframe, figure 19a e 19b, ottenuti dalla divisione utilizzata nel Random Forest.
- Cominciamo con il creare il modello per il training e per il testing, in essi dobbiamo inserire:
  - La colonna in cui è contenuta la variabile target in confronto con tutte le altre
  - Il dataframe per l'apprendimento
- Successivamente creiamo le matrici di confusione delle due fasi, training e testing, e il risultato è:

```
Confusion Matrix and Statistics

          Reference
Prediction    N    Y
         N    0    0
         Y 1384  112

               Accuracy : 0.0749
                 95% CI : (0.062, 0.0894)
    No Information Rate : 0.9251
    P-Value [Acc > NIR] : 1

                  Kappa : 0

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 1.00000
            Specificity : 0.00000
         Pos Pred Value : 0.07487
         Neg Pred Value :     NaN
             Prevalence : 0.07487
         Detection Rate : 0.07487
   Detection Prevalence : 1.00000
      Balanced Accuracy : 0.50000

       'Positive' Class : Y
```

(a) Matrice di Confusione del training

```
Confusion Matrix and Statistics

          Reference
Prediction   N   Y
         N   0   0
         Y 593  48

               Accuracy : 0.0749
                 95% CI : (0.0557, 0.0981)
    No Information Rate : 0.9251
    P-Value [Acc > NIR] : 1

                  Kappa : 0

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 1.00000
            Specificity : 0.00000
         Pos Pred Value : 0.07488
         Neg Pred Value :     NaN
             Prevalence : 0.07488
         Detection Rate : 0.07488
   Detection Prevalence : 1.00000
      Balanced Accuracy : 0.50000

       'Positive' Class : Y
```

(b) Matrice di Confusione del testing

Inaspettatamente, anche se il dataframe è sbilaniciato quello che otteniamo è un modello che classifica perfettamente la variabile target. Però la classe opposta viene totalmente misclassificata andando a ridurre drasticamente i valori delle metriche fatta eccezione della sensibilità.

4.2.2 Bilanciato

- Per eseguire l'analisi bilanciata utilizziamo gli stessi dataframe bilanciati, figure 23a e 23b, ottenuti dalla divisione utilizzata nel Random Forest Bilanciato.
- Cominciamo quindi con il creare le matrici di confusione del training e del testing in maniera tale da verificare l'andamento delle prestazioni: Notiamo come, dopo il bilanciamento, nel

```
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0   12    3
         1 1372 1404

               Accuracy : 0.5073
                 95% CI : (0.4886, 0.5261)
    No Information Rate : 0.5041
    P-Value [Acc > NIR] : 0.3738

                  Kappa : 0.0066

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.997868
            Specificity : 0.008671
         Pos Pred Value : 0.505764
         Neg Pred Value : 0.800000
             Prevalence : 0.504120
         Detection Rate : 0.503046
   Detection Prevalence : 0.994626
      Balanced Accuracy : 0.503269

       'Positive' Class : 1
```

(a) Matrice di Confusionedel training

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0   6   1
         1 587 602

               Accuracy : 0.5084
                 95% CI : (0.4796, 0.5371)
    No Information Rate : 0.5042
    P-Value [Acc > NIR] : 0.3974

                  Kappa : 0.0085

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.99834
            Specificity : 0.01012
         Pos Pred Value : 0.50631
         Neg Pred Value : 0.85714
             Prevalence : 0.50418
         Detection Rate : 0.50334
   Detection Prevalence : 0.99415
      Balanced Accuracy : 0.50423

       'Positive' Class : 1
```

(b) Matrice di Confusione del testing

testing l'accuracy sia aumentata del $\approx$ 43% e la senisibilità rimanga invariata; mentre la specificità rimane molto bassa, 1%. Possiamo affermare quindi che il modello creato per l'algoritmo Naive Bayes non sia ottimale, in quanto non riconosce la classe delle etichettata come N abbassando tutte le prestazioni.

4.3 Confronto tra Python

Dopo l'esecuzione di questa analisi sull'algoritmo Naive Bayes possiamo affermare che le prestazioni, ottenute dai modelli bilanciati, sono senza ombra di dubbio migliori in Python. In quanto le prestazioni del modello restituiscono:

- un'accuracy più alta del$\approx$ 46%
- una sensibilità più alta dell'1%
- una specificità più alta del 92%

Affermiamo quindi che l'esecuzione su R non sia per nulla ottimale a livello complessivo, di conseguenza Python è il linguaggio migliore su cui eseguire una classificazione tramite il Naive Bayes.

5 Decision Tree

L'albero di decisione è un classificatore che ha una struttura ad albero:

- Ogni nodo di decisione effettua un test su un singolo attributo
- I rami da ogni nodo di decisione indicano i possibili valori dell'attributo
- Le foglie dell'albero portano alla decisione finale

L'albero di decisione viene utilizzato partendo dalla radice e muovendosi verso il basso attraversando i vari nodi di decisione fino a raggiungere una foglia che indica la classe di appartenenza, a volte possono essere anche usati per scopi di regressione.

Gli alberi di decisione sono molto semplici, usano regole facilmente comprensibili ed implementabili con costrutti if e then, i requisiti fondamentali sono: l'oggetto deve essere rappresentato mediante una serie di features, che possono essere sia categoriche che numeriche, e i dati di training devono essere sufficienti e possibilmente ricoprire tutti i possibili intervalli di valori degli attributi

5.1 Python

5.1.1 Sbilanciato

- Utilizziamo gli stessi dataframe, figura 8a, ottenuti dalla divisione utilizzata nel Random Forest.
- Successivamente visualizziamo l'accuracy del training e del testing:

DecisioneTree	0.937390

(a) Accuracy del training

```
Accuracy finale: 0.9252
```

(b) Accuracy del testing

Capiamo dal risultato ottenuto che nonostante lo sbilanciamento l'accuracy rimane stabile, quindi non mi aspetto prestazioni elevate ma nemmeno del tutto basse.

- Per confermare quanto appena detto, visualizziamo la matrice di confusione e le metriche per stabilire le prestazioni del modello:

```
True Positives(TP) =  394

True Negatives(TN) =  2

False Positives(FP) =  3

False Negatives(FN) =  29
```

	Actual Positive	Actual Negative
Predict Positive	394	3
Predict Negative	29	2

(a) Matrice di Confusionedel training

```
Other Metrics:
- TP: 394 , TN: 2 , FP: 3 , FN: 29
- Classification accuracy: 0.9252
- Classification error: 0.0748
- Precision: 0.9924
- Recall or Sensitivity: 0.9314
- True Positive Rate: 0.9314
- False Positive Rate: 0.6000
- Specificity: 0.4000
```

(b) Matrice di Confusione del testing

Dai risultati che otteniamo possiamo affermare che il modello non si presta bene nella ricerca delle classi Y. In quanto la misclassificazione è molto elevata, quindi dobbiamo ricrederci su quanto assunto all'inizio.

A confermare ancora quanto appena detto è la curva ROC delle prestazioni del modello:

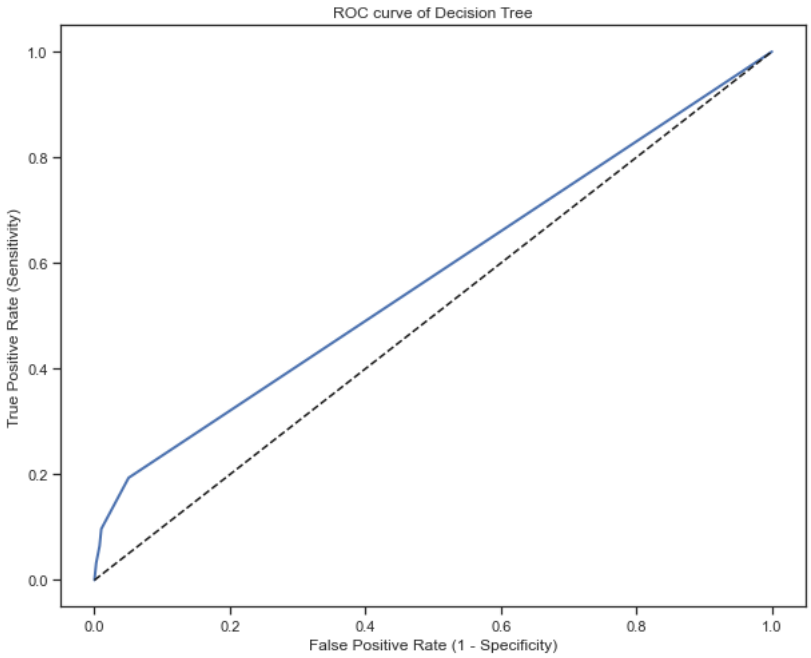


Figure 44

5.1.2 Bilanciato

- Utilizziamo gli stessi dataframe bilanciati, figure 14a 14b, ottenuti dalla divisione utilizzata nel Random Forest.
- Procediamo dunque visualizzando le accuracy del testing e del training e vedere come sono le prestazioni con un dataset bilanciato rispetto allo sbilanicato:

DecisioneTree	0.790389

(a) Accuracy nel training

```
Accuracy finale: 0.81036
```

(b) Accuracy nel testing

Notiamo come l'accuracy sia discretamente bassa anche con un dataframe bilanciato e inspiegabilmente sia inferiore allo sbilanciato.

- Controlliamo di conseguenza la matrice di confusione e le metriche per valutare le prestazioni:

```
True Positives(TP) =  304

True Negatives(TN) =  337

False Positives(FP) =  108

False Negatives(FN) =  42
```

	Actual Positive	Actual Negative
Predict Positive	304	108
Predict Negative	42	337

(a) Matrice di Confusione

```
Other Metrics:
- TP: 304 , TN: 337 , FP: 108 , FN: 42
- Classification accuracy: 0.8104
- Classification error: 0.1896
- Precision: 0.7379
- Recall or Sensitivity: 0.8786
- True Positive Rate: 0.8786
- False Positive Rate: 0.2427
- Specificity: 0.7573
```

(b) Metriche

Il risultato ci mostra una buona classificazione delle classi Y anche se parti di esse vengono misclassificate. Nel modello notiamo comunque un buon miglioramento delle prestazioni:

- la specificità in questo caso va confrontata con la sensibilità in quanto la classe Y essendo in netta minoranza il modello riconosce come variabile target la classe N infatti è diminunita del $\approx 17\%$
- invece la sensibilità va confrontata con la specificità e di conseguenza è presente un miglioramento del 47%

Infine possiamo dire che come classificatore non è eccelso in quanto non riconosce totalmente le due classi, inoltre va a misclassificare eccessivamente la classe delle N. Possiamo anche confermarlo osservando la curva ROC del modello, che ci conferma quanto appena detto:

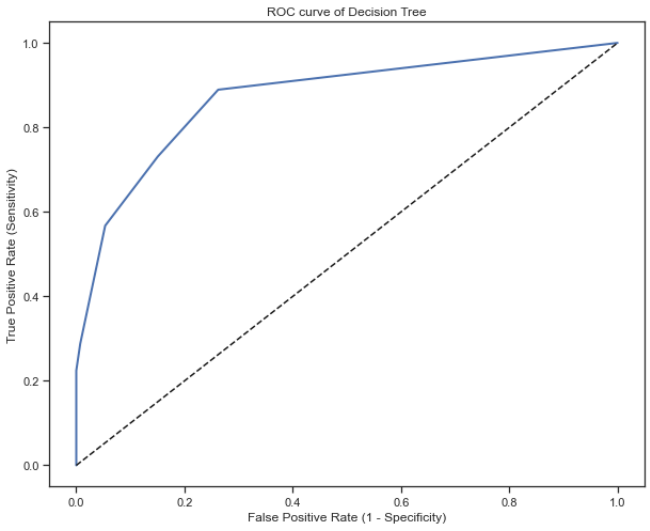


Figure 47

5.2 R

5.2.1 Sbilanciato

- Utilizziamo gli stessi dataframe, figure 19a e 19b, ottenuti dalla divisione utilizzata nel Random Forest.
- Iniziamo creando il modello per il training e per il testing, ed in essi inseriamo:
  - La colonna in cui è contenuta la variabile target in confronto con tutte le altre
  - Il dataframe per l'apprendimento
- Realizziamo quindi le matrici di confusione delle due fasi, quello che otteniamo è mostrato di seguito.

```
Confusion Matrix and Statistics

          Reference
Prediction    N    Y
         N 1377  101
         Y    7   11

               Accuracy : 0.9278
                 95% CI : (0.9135, 0.9404)
    No Information Rate : 0.9251
    P-Value [Acc > NIR] : 0.3701

                  Kappa : 0.1516

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.098214
            Specificity : 0.994942
         Pos Pred Value : 0.611111
         Neg Pred Value : 0.931664
             Prevalence : 0.074866
         Detection Rate : 0.007353
   Detection Prevalence : 0.012032
      Balanced Accuracy : 0.546578

       'Positive' Class : Y
```

(a) Matrice di Confusionedel training

```
Confusion Matrix and Statistics

          Reference
Prediction   N   Y
         N 592  47
         Y   1   1

               Accuracy : 0.9251
                 95% CI : (0.9019, 0.9443)
    No Information Rate : 0.9251
    P-Value [Acc > NIR] : 0.5383

                  Kappa : 0.0342

 Mcnemar's Test P-Value : 8.293e-11

            Sensitivity : 0.02083
            Specificity : 0.99831
         Pos Pred Value : 0.50000
         Neg Pred Value : 0.92645
             Prevalence : 0.07488
         Detection Rate : 0.00156
   Detection Prevalence : 0.00312
      Balanced Accuracy : 0.50957

       'Positive' Class : Y
```

(b) Matrice di Confusione del testing

Dai risultati che otteniamo possiamo dedurre che in questa fase il modello non è ottimale, in quanto la variabile target etichettata come Y non viene riconosciuta.

5.2.2 Bilanciato

- Per eseguire l'analisi bilanciata utilizziamo gli stessi dataframe bilanciati, figure 23a e 23b, ottenuti dalla divisione utilizzata nel Random Forest Bilanciato.
- Iniziamo creando le matrici di confusione del training e del testing in maniera tale di verificarne le prestazioni:

```
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 1309  168
         1   75 1239

               Accuracy : 0.9129
                 95% CI : (0.9019, 0.9231)
    No Information Rate : 0.5041
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.826

 Mcnemar's Test P-Value : 3.596e-09

            Sensitivity : 0.8806
            Specificity : 0.9458
         Pos Pred Value : 0.9429
         Neg Pred Value : 0.8863
             Prevalence : 0.5041
         Detection Rate : 0.4439
   Detection Prevalence : 0.4708
      Balanced Accuracy : 0.9132

       'Positive' Class : 1
```

(a) Matrice di Confusionedel training

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 559  86
         1  34 517

               Accuracy : 0.8997
                 95% CI : (0.8812, 0.9161)
    No Information Rate : 0.5042
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.7995

 Mcnemar's Test P-Value : 3.23e-06

            Sensitivity : 0.8574
            Specificity : 0.9427
         Pos Pred Value : 0.9383
         Neg Pred Value : 0.8667
             Prevalence : 0.5042
         Detection Rate : 0.4323
   Detection Prevalence : 0.4607
      Balanced Accuracy : 0.9000

       'Positive' Class : 1
```

(b) Matrice di Confusione del testing

Il risultato fornisce delle buone prestazioni in quanto l'accuracy è diminuita leggermente ed ha un valore di $\approx 89\%$. Inoltre otteniamo:

- la classe delle Y complessivamente viene riconosciuta
- la sensibilità è aumentata dell'83%
- la specificità è rimasta invariata

Possiamo dunque confermare che il modello creato secondo l'algoritmo dei Decision Tree è un buon classificatore.

5.3 Confronto tra Python ed R

Dopo aver concluso quest'analisi dell'algoritmo dei Decision Tree e confrontato i risultati delle prestazioni, ottenuti dai due modelli eseguiti su dataframe bilanciati, il miglior linguaggio su cui eseguire il modello in questo caso è R. In quanto le prestazioni sono:

- per l'accuracy ci sta un aumento dell'8%
- la sensiblità è pressochè uguale
- la specificità è aumentata del 19%

Avendo dunque visto i risultati confermiamo che il linguaggio migliore, anche se di poco, su cui eseguire l'algoritmo è R.

6 Conclusioni

6.1 Python

Dopo aver eseguito tutte le analisi possiamo stimare il modello con le migliori prestazioni, verificheremo ciò attraverso un confronto delle curve ROC create nelle analisi precedenti:

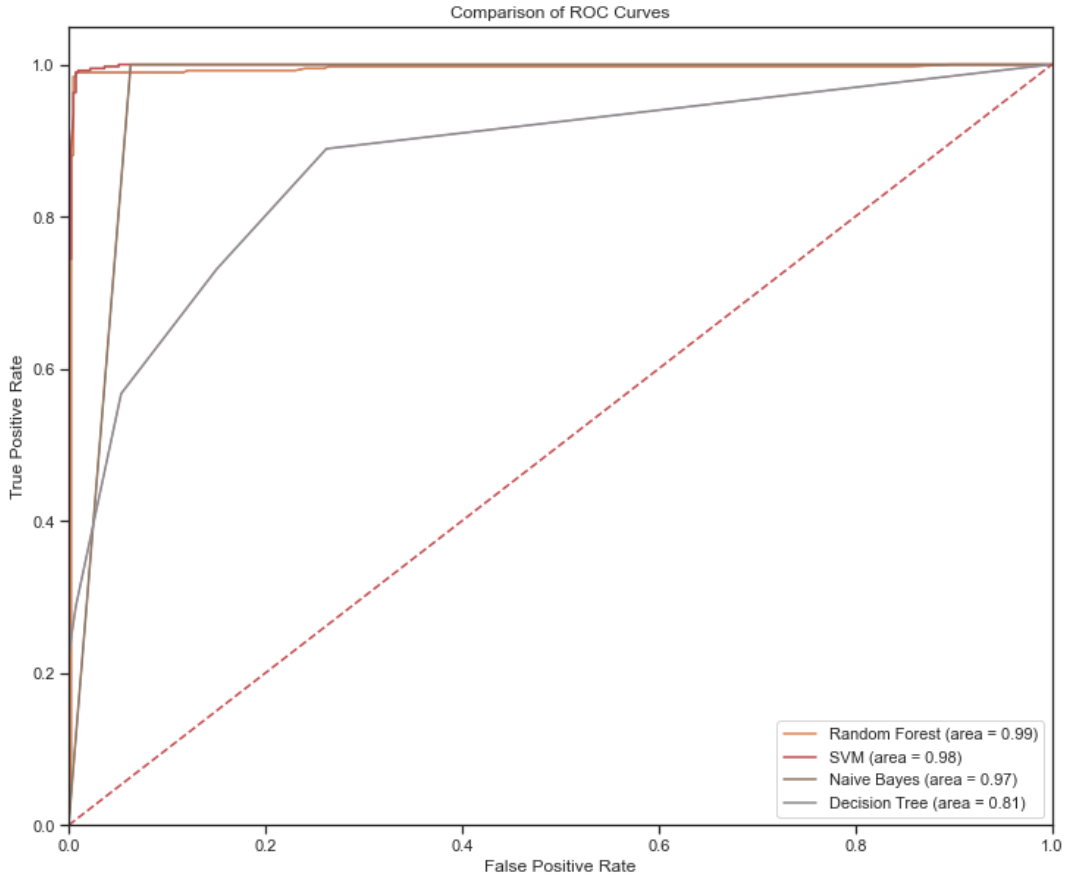


Figure 50

Dal risultato ottenuto possiamo affermare che in quest'analisi il peggior algoritmo che possiamo utilizzare è il Decision Tree avendo comunque l'accuracy più bassa di tutti. Invece il miglior algoritmo che abbiamo realizzato in Python risulta essere il modello dell'SVM, in quanto supera di poco il modello del Random Forest nella sensibilità rendendolo più preciso.

6.2 R

Dopo un attento controllo delle prestazioni tramite le matrici di confusione possiamo assumere che il peggior algoritmo da utilizzare è il Naive Bayes, figura 41b, in quanto non riconosce la classe delle N e questo porta ad un peggioramento delle prestazioni. Invece il miglior algoritmo che abbiamo realizzato è il Random Forest, figura 24b, in quanto le sue prestazioni sono le più alte e anche i valori dell'accuracy sono molto elevati.

6.3 Complessive

Complessivamente nei due linguaggi gli algoritmi lavorano discretamente sia in termini di tempo e sia in termini di prestazioni. Ma il migliore su cui eseguirli è Python, in quanto:

- restituisce le prestazioni migliori,
- un'accuracy più alta,
- le variabili target vengono classificate perfettamente.

Infine, come già detto, il modello creato più conveniente è l'algoritmo SVM poichè le sue prestazioni sono le migliori. Inoltre nella fase di testing le classi Y ed N vengono riconosciute totalmente.