Tesina di Apprendimento Statistico

Giuseppe Intilla 297641

Anno accademico 2021/2022

Indice

Questo progetto ha lo scopo di analizzare un set contenente dati medici mediante l'uso di tecniche di machine learning.

1 Dataset

Il dataset in questione è presente su Kaggle ed è stato fornito dalla UCI Machine Learning Repository. Contiene 569 osservazioni con 32 features e non ci sono missing values.

1.1 Features

La prima feature è l'ID, utilizzato per riconoscere il paziente, che verrà eliminata dal dataset in quanto non influente per l'analisi che farò. Qui è riportata una frazione del dataset:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave.points_mean	symmetry_mean
1	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419
2	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812
3	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069
4	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597
5	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809
6	M	12.45	15.70	82.57	477.1	0.12780	0.17000	0.1578	0.08089	0.2087

	fractal_dimension_mean	radius_se	texture_se	perimeter_se	area_se	smoothness_se	compactness_se	concavity_se	concave.points_se	symmetry_se
1	0.07871	1.0950	0.9053	8.589	153.40	0.006399	0.04904	0.05373	0.01587	0.03003
2	0.05667	0.5435	0.7339	3.398	74.08	0.005225	0.01308	0.01860	0.01340	0.01389
3	0.05999	0.7456	0.7869	4.585	94.03	0.006150	0.04006	0.03832	0.02058	0.02250
4	0.09744	0.4956	1.1560	3.445	27.23	0.009110	0.07458	0.05661	0.01867	0.05963
5	0.05883	0.7572	0.7813	5.438	94.44	0.011490	0.02461	0.05688	0.01885	0.01756
6	0.07613	0.3345	0.8902	2.217	27.19	0.007510	0.03345	0.03672	0.01137	0.02165

	fractal_dimension_se	radius_worst	texture_worst	perimeter_worst	area_worst	smoothness_worst	compactness_worst	concavity_worst
1	0.006193	25.38	17.33	184.60	2019.0	0.1622	0.6656	0.7119
2	0.003532	24.99	23.41	158.80	1956.0	0.1238	0.1866	0.2416
3	0.004571	23.57	25.53	152.50	1709.0	0.1444	0.4245	0.4504
4	0.009208	14.91	26.50	98.87	567.7	0.2098	0.8663	0.6869
5	0.005115	22.54	16.67	152.20	1575.0	0.1374	0.2050	0.4000
6	0.005082	15.47	23.75	103.40	741.6	0.1791	0.5249	0.5355

	concave.points_worst	symmetry_worst	fractal_dimension_worst
1	0.2654	0.4601	0.11890
2	0.1860	0.2750	0.08902
3	0.2430	0.3613	0.08758
4	0.2575	0.6638	0.17300
5	0.1625	0.2364	0.07678
6	0.1741	0.3985	0.12440

Figura 1: Primi 6 record del dataset

La classe target è **diagnosis** che indica il tipo di tumore del paziente e può assumere valore **B**, cioè tumore benigno, e **M**, cioè tumore maligno. Si può notare, inoltre, come tutte le altre features, che saranno usate come variabili, sono di tipo numerico e a valori reali. In particolare ci sono 10 parametri rilevanti per la diagnosi e riguardanti misure fatte su ogni nucleo delle cellule. Per ognuno di questi parametri vengono riportati, in ordine, media, errore standard e caso peggiore (cioè la medie dei tre valori più grandi) tra tutte le misure fatte.

1.2 Exploratory analysis

Per esplorare il dataset ho utilizzato il linguaggio di programmazione R con la piattaforma RStudio. Inizialmente faccio un'analisi della distribuzione della classe target che risulta leggermente sbilanciata:

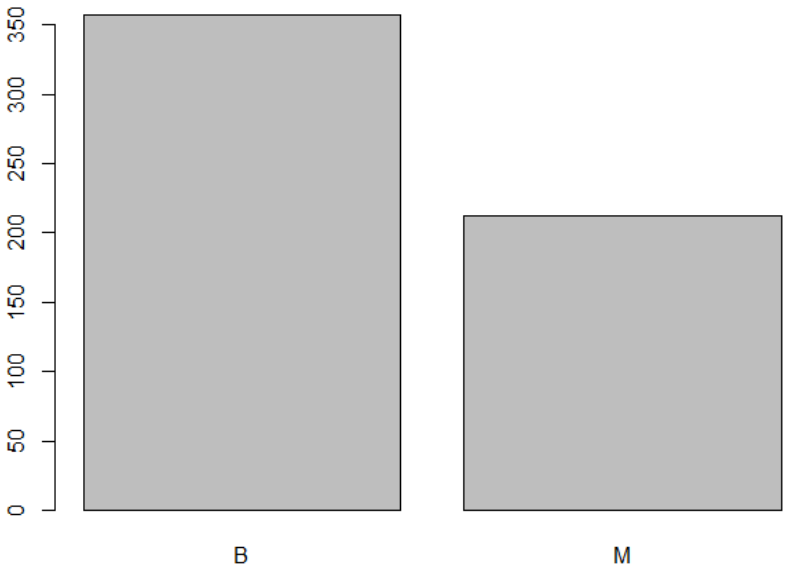


Figura 2: Distribuzione della classe target

Adesso vediamo un summary delle variabili per vederne le caratteristiche descrittive:

```
 radius_mean      texture_mean   perimeter_mean     area_mean      smoothness_mean   compactness_mean  concavity_mean    concave.points_mean
Min.   : 6.981   Min.   : 9.71   Min.   : 43.79   Min.   : 143.5   Min.   :0.05263   Min.   :0.01938   Min.   :0.00000   Min.   :0.00000
1st Qu.:11.700   1st Qu.:16.17   1st Qu.: 75.17   1st Qu.: 420.3   1st Qu.:0.08637   1st Qu.:0.06492   1st Qu.:0.02956   1st Qu.:0.02031
Median :13.370   Median :18.84   Median : 86.24   Median : 551.1   Median :0.09587   Median :0.09263   Median :0.06154   Median :0.03350
Mean   :14.127   Mean   :19.29   Mean   : 91.97   Mean   : 654.9   Mean   :0.09636   Mean   :0.10434   Mean   :0.08880   Mean   :0.04892
3rd Qu.:15.780   3rd Qu.:21.80   3rd Qu.:104.10   3rd Qu.: 782.7   3rd Qu.:0.10530   3rd Qu.:0.13040   3rd Qu.:0.13070   3rd Qu.:0.07400
Max.   :28.110   Max.   :39.28   Max.   :188.50   Max.   :2501.0   Max.   :0.16340   Max.   :0.34540   Max.   :0.42680   Max.   :0.20120
symmetry_mean    fractal_dimension_mean   radius_se        texture_se      perimeter_se       area_se         smoothness_se
Min.   :0.1060   Min.   :0.04996        Min.   :0.1115   Min.   :0.3602   Min.   : 0.757   Min.   :  6.802   Min.   :0.001713
1st Qu.:0.1619   1st Qu.:0.05770        1st Qu.:0.2324   1st Qu.:0.8339   1st Qu.: 1.606   1st Qu.: 17.850   1st Qu.:0.005169
Median :0.1792   Median :0.06154        Median :0.3242   Median :1.1080   Median : 2.287   Median : 24.530   Median :0.006380
Mean   :0.1812   Mean   :0.06280        Mean   :0.4052   Mean   :1.2169   Mean   : 2.866   Mean   : 40.337   Mean   :0.007041
3rd Qu.:0.1957   3rd Qu.:0.06612        3rd Qu.:0.4789   3rd Qu.:1.4740   3rd Qu.: 3.357   3rd Qu.: 45.190   3rd Qu.:0.008146
Max.   :0.3040   Max.   :0.09744        Max.   :2.8730   Max.   :4.8850   Max.   :21.980   Max.   :542.200   Max.   :0.031130
compactness_se      concavity_se     concave.points_se   symmetry_se       fractal_dimension_se  radius_worst   texture_worst
Min.   :0.002252   Min.   :0.00000   Min.   :0.000000   Min.   :0.007882   Min.   :0.0008948     Min.   : 7.93   Min.   :12.02
1st Qu.:0.013080   1st Qu.:0.01509   1st Qu.:0.007638   1st Qu.:0.015160   1st Qu.:0.0022480     1st Qu.:13.01   1st Qu.:21.08
Median :0.020450   Median :0.02589   Median :0.010930   Median :0.018730   Median :0.0031870     Median :14.97   Median :25.41
Mean   :0.025478   Mean   :0.03189   Mean   :0.011796   Mean   :0.020542   Mean   :0.0037949     Mean   :16.27   Mean   :25.68
3rd Qu.:0.032450   3rd Qu.:0.04205   3rd Qu.:0.014710   3rd Qu.:0.023480   3rd Qu.:0.0045580     3rd Qu.:18.79   3rd Qu.:29.72
Max.   :0.135400   Max.   :0.39600   Max.   :0.052790   Max.   :0.078950   Max.   :0.0298400     Max.   :36.04   Max.   :49.54
perimeter_worst     area_worst     smoothness_worst  compactness_worst concavity_worst  concave.points_worst symmetry_worst
Min.   : 50.41   Min.   : 185.2   Min.   :0.07117   Min.   :0.02729   Min.   :0.0000   Min.   :0.00000      Min.   :0.1565
1st Qu.: 84.11   1st Qu.: 515.3   1st Qu.:0.11660   1st Qu.:0.14720   1st Qu.:0.1145   1st Qu.:0.06493      1st Qu.:0.2504
Median : 97.66   Median : 686.5   Median :0.13130   Median :0.21190   Median :0.2267   Median :0.09993      Median :0.2822
Mean   :107.26   Mean   : 880.6   Mean   :0.13237   Mean   :0.25427   Mean   :0.2722   Mean   :0.11461      Mean   :0.2901
3rd Qu.:125.40   3rd Qu.:1084.0   3rd Qu.:0.14600   3rd Qu.:0.33910   3rd Qu.:0.3829   3rd Qu.:0.16140      3rd Qu.:0.3179
Max.   :251.20   Max.   :4254.0   Max.   :0.22260   Max.   :1.05800   Max.   :1.2520   Max.   :0.29100      Max.   :0.6638
fractal_dimension_worst
Min.   :0.05504
1st Qu.:0.07146
Median :0.08004
Mean   :0.08395
3rd Qu.:0.09208
Max.   :0.20750
```

Figura 3: Sommario delle varibili

Per valutare la relazione tra le diverse variabili faccio un plot del correlogramma:

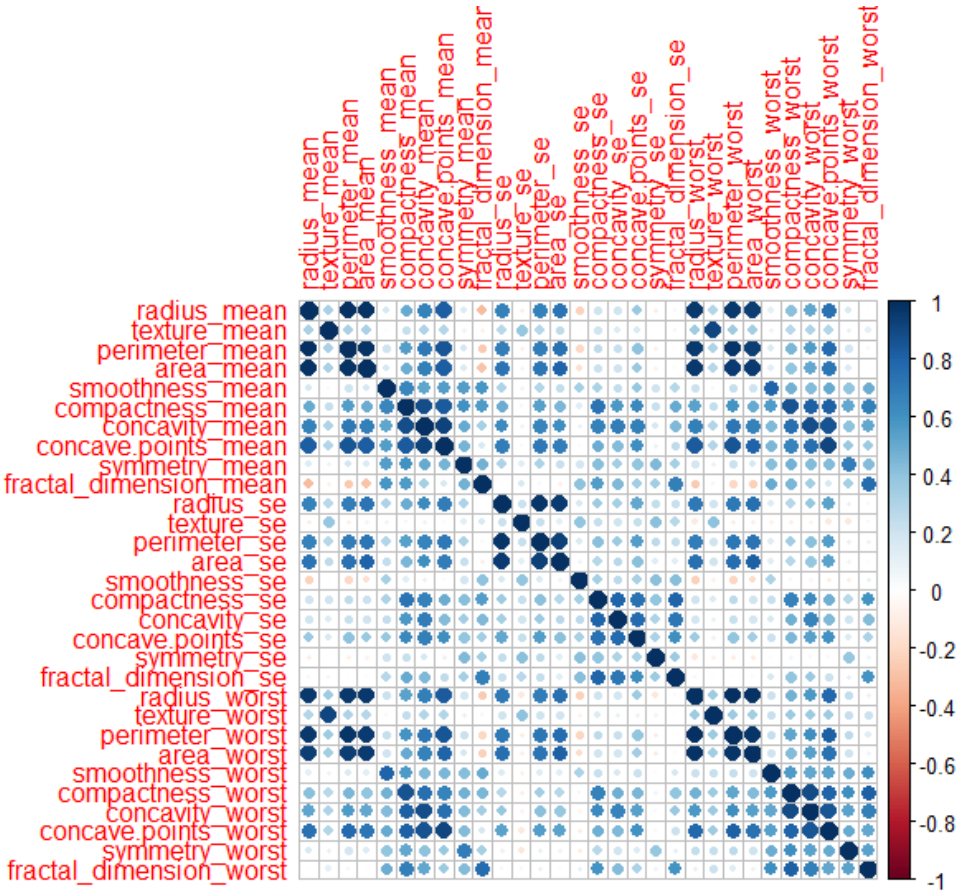


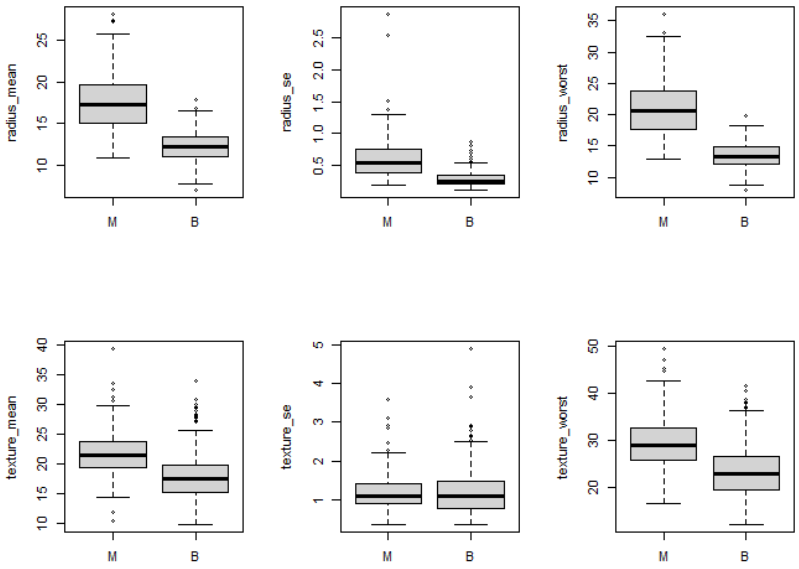
Figura 4: Correlogramma delle variabili

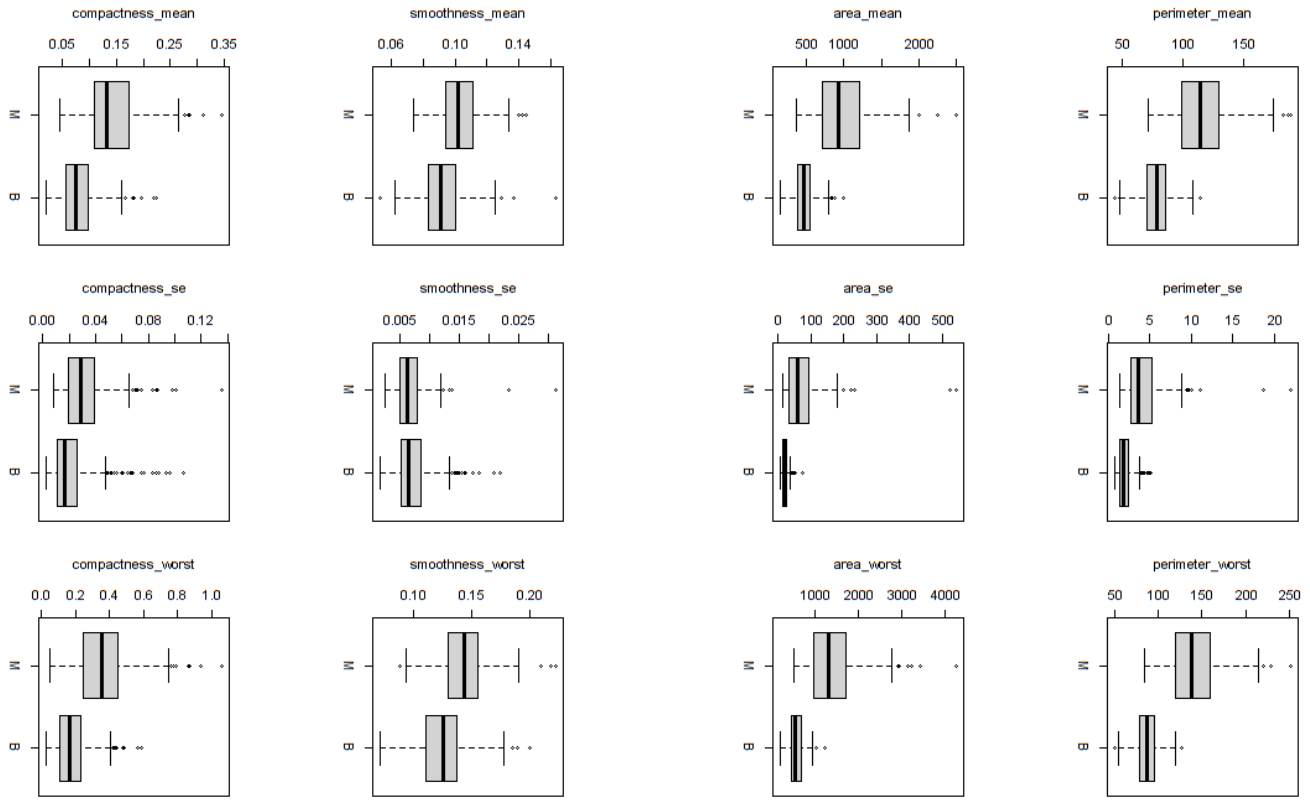
Qui possiamo notare che vi è importante correlazione fra alcune delle variabili. In particolare è facile intuire come ci sia correlazione tra le variabili che riportano una misura della media e una misura del caso peggiore come, ad esempio, **radius mean** e **radius worst**. Inoltre sono alte anche le correlazioni fra variabili che sono strettamente collegate alla grandezza fisica del nucleo della cellula, come, ad esempio, **radius, perimeter** e **area**.

Adesso faccio un'analisi delle singole variabili, valutandone la deviazione standard e facendone i boxplot:

radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean
3.524049e+00	4.301036e+00	2.429898e+01	3.519141e+02	1.406413e-02	5.281276e-02
concavity_mean	concave.points_mean	symmetry_mean	fractal_dimension_mean	radius_se	texture_se
7.971981e-02	3.880284e-02	2.741428e-02	7.060363e-03	2.773127e-01	5.516484e-01
perimeter_se	area_se	smoothness_se	compactness_se	concavity_se	concave.points_se
2.021855e+00	4.549101e+01	3.002518e-03	1.790818e-02	3.018606e-02	6.170285e-03
symmetry_se	fractal_dimension_se	radius_worst	texture_worst	perimeter_worst	area_worst
8.266372e-03	2.646071e-03	4.833242e+00	6.146258e+00	3.360254e+01	5.693570e+02
smoothness_worst	compactness_worst	concavity_worst	concave.points_worst	symmetry_worst	fractal_dimension_worst
2.283243e-02	1.573365e-01	2.086243e-01	6.573234e-02	6.186747e-02	1.806127e-02

Figura 5: Deviazione standard delle variabili




compactness_mean
0.05 0.15 0.25 0.35
smoothness_mean
0.06 0.10 0.14
area_mean
500 1000 2000
perimeter_mean
50 100 150
compactness_se
0.00 0.04 0.08 0.12
smoothness_se
0.005 0.015 0.025
area_se
0 100 200 300 400 500
perimeter_se
0 5 10 15 20
compactness_worst
0.0 0.2 0.4 0.6 0.8 1.0
smoothness_worst
0.10 0.15 0.20
area_worst
1000 2000 3000 4000
perimeter_worst
50 100 150 200 250
M
B

concavity_mean
concavity_se
concavity_worst
concave.points_mean
concave.points_se
concave.points_worst
symmetry_mean
symmetry_se
symmetry_worst
fractal_dimension_mean
fractal_dimension_se
fractal_dimension_worst
M
B

Possiamo vedere come alcune variabili, soprattutto quelle relative agli errori sulle misure, non hanno un range di valori tali da poter essere utili alla diagnosi. Infatti, in seguito, applicherò delle tecniche di riduzione dimensionale in modo da alleggerire il dataset, eliminando le variabili che non portano informazioni, ma senza perdere varianza spiegata.

2 Riduzione dimensionale

In questa sezione voglio applicare al dataset una tecnica di unsupervised learning per tentare di diminuirne la dimensionalità. In particolare l'obbiettivo è quello di ottenere un dataset di dimensioni notevolmente ridotte mantenendo, però, le informazioni necessarie per poter allenare dei modelli predittivi efficaci. Questa analisi e le successive sono state fatte con il linguaggio di programmazione Python ed in particolare la libreria Scikit-learn, usando la piattaforma Google Colab.

2.1 PCA

La principal component analysis è un algoritmo non-supervisionato che tenta di traformare le feature del dataset in modo da diminuirne la numerosità mantenendo alta la varianza spiegata. Operativamente si tenta di capire quali sono le direzioni nello spazio delle features con la maggiore varianza.

Partiamo dal presupposto che ogni vettore dei dati x sia distribuito come una normale $d - variata$, con media nulla e matrice di varianza e covarianza Σ. Dato che Σ è una matrice simmetrica e definita positiva può essere decomposta tramite SVD in $\Sigma = UD^2U'$. Si vede che U contiene nelle colonne le direzioni delle componenti principali di Σ, e D ne presenta i rispettivi valori. A questo punto scegliendo $k << d$ possiamo proiettare la nostra matrice di covariate nello spazio generato dalle prime k componenti principali, cioè colonne di U, nel modo seguente: $z_i = U_k U_k' x_i$. Inoltre possiamo vedere come la frazione di varianza spiegata dalla componente principale i-esima sia data da: $\frac{D_{ii}^2}{\sum_{j=1}^{d} D_{jj}^2}$.

Nel nostro caso non avendo Σ possiamo stimarla ed utilizzare lo stesso procedimento con $\hat{\Sigma} = \frac{1}{n} X'X$.

Qui vediamo i risultati dell'algoritmo applicato al training set:

PC1	PC2	PC3	PC4	PC5
9.79849847e-01	1.80025321e-02	1.94216857e-03	1.04561885e-04	8.97275429e-05

Tabella 1: Prime 5 componenti principali

Possiamo notare come l'algoritmo abbia avuto ottime performance. Infatti, già le prime componenti principali riescano a spiegare bene la varianza del dataset.

3 Classificazione

In questa parte analizzerò il dataset in questione con diverse tecniche di classificazione per creare dei modelli in grado di prevedere l'appartenenza o meno di un paziente alla classe target. Il primo step è quello di dividere in modo casuale il dataset in due campioni, uno che verrà usato per il training e l'altro per il test, utilizzando, in particolare, una proporzione di 6 : 4. Inoltre ho deciso di utilizzare un metodo di grid search combinato alla cross-validation per selezionare il modello migliore per ogni famiglia di stimatori. Il procedimento che ho seguito prevede di:

- dividere randomicamente il training set in k parti;
- ad ogni passo utilizzare una delle k parti per testare il modello e le restanti per addestrarlo;
- dopo k passi valutare il modello che ha avuto performance migliori;
- eseguire su quest'ultimo un addestramento completo con tutto il training set;
- testare il modello finale sul test set per valutarne il generalization risk.

Questa tecnica garantisce un miglioramento dell'accuratezza e evita problemi di overfitting o underfitting. Nella mia analisi ho scelto k=4, in modo che, ad ogni passo, l'addestramento venga fatto sul 75% del training set. Inoltre, nella tecnica di grid search, la selezione del modello migliore è stata fatta utilizzando la metrica *F1* che permette di ottenere modelli più bilanciati della semplice *accuracy*.

3.1 Regressione logistica

La regressione logistica è un *generalized linear model* in cui la varibile risposta, che è binaria, viene modellata nel modo seguente:

$$y = logit(p) = \beta_0 + X\beta_1 \,,$$

dove *logit* indica la funzione logistica, p rappresenta la probabilità di appartenere alla classe target, X è la matrice delle covariate e β_0, β_1 sono i regressori. In particolare, dopo un pò di manipolazione, si ottiene che il parametro di interesse vale:

$$p = \frac{e^{\beta_0 + X\beta_1}}{1 + e^{\beta_0 + X\beta_1}} \,.$$

Riporto, adesso, i risultati ottenuti valutando il modello con i parametri migliori sul test set.

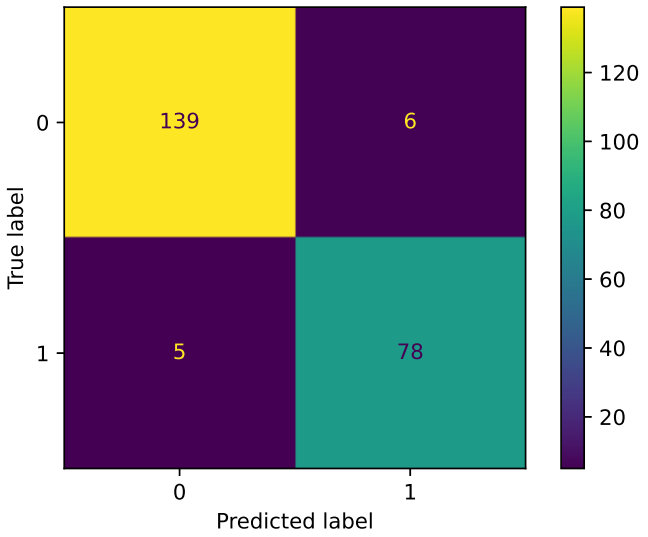


Figura 6: Confusion matrix del modello Logit

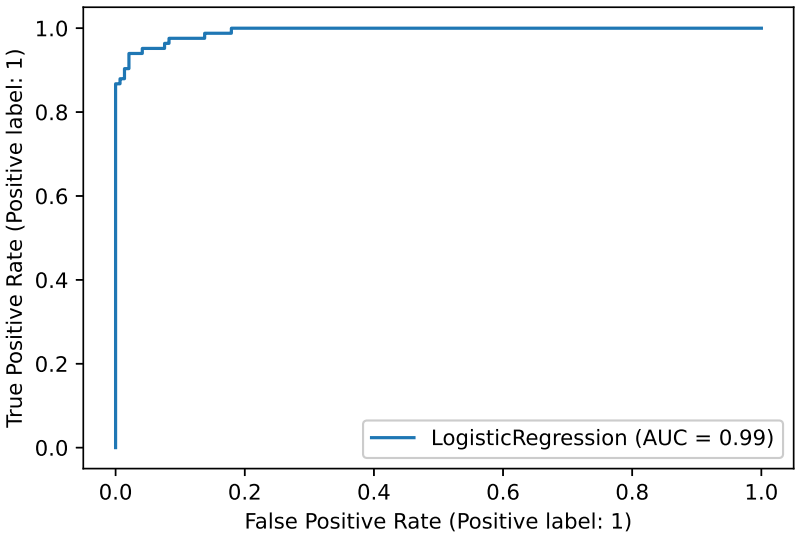


Figura 7: ROC del modello Logit

Accuracy	Precision	Recall	F1
0.952	0.929	0.94	0.934

Tabella 2: Risultati del modello Logit

Il modello riportato ha utilizzato i seguenti iper-parametri:

- C=0.1, cioè un parametro inversamente proporzionale al livello di regolarizzazione;
- Penalty=l2, cioè la norma usata per definire la penalty;

e possiamo notare come abbia raggiunto ottime performance.

3.2 Decision tree

Il decision tree è un metodo che si basa sull'idea di dividere il dataset in sottoparti sempre più piccole seguendo delle semplici regole di separazione. La previsione viene poi fatta seguendo le stesse regole, inserendo il nuovo valore ad uno dei sottogruppi creati e assegnandolo alla classe maggioritaria di quella regione. Le regole di separazione vengono scelte al fine di minimizzare la training loss. In particolare i criteri più usati per valutare una splitting rule sono:

- Gini: $\frac{1}{2}(1 - \sum_c p_c^2)$;
- Entropy: $-\sum_c p_c \, log_2(c)$;

dove c rappresenta la classe e p_c la corrispondente frequenza relativa nella corrispondente regione di dati da suddividere. Inoltre è importante scegliere un corretto criterio di stop in modo da evitare che l'albero si adatti troppo ai dati e perda capacità di predizione.

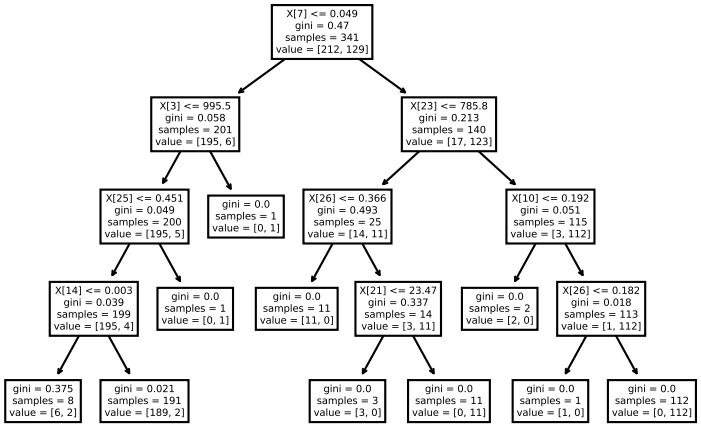


Figura 8: Albero creato dal modello

Riporto, adesso, i risultati ottenuti valutando il modello con i parametri migliori sul test set.

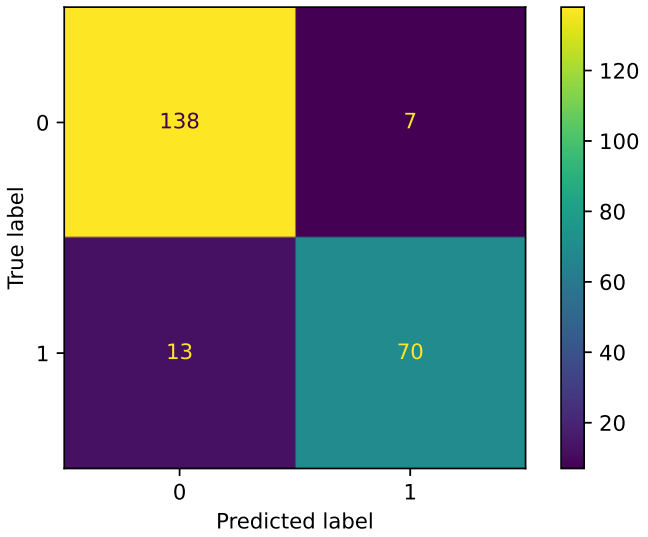


Figura 9: Confusion matrix del modello Tree

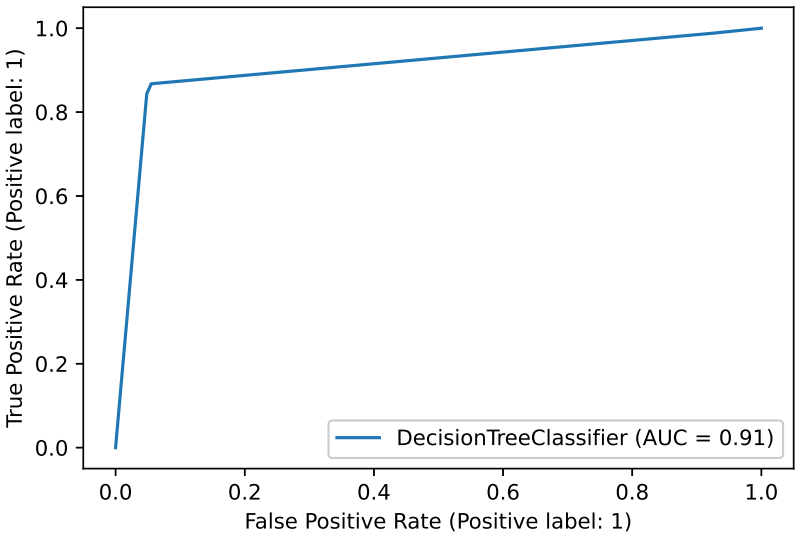


Figura 10: ROC del modello Tree

Accuracy	Precision	Recall	F1
0.912	0.909	0.843	0.875

Tabella 3: Risultati del modello Tree

Il modello riportato ha utilizzato i seguenti iper-parametri:

- Splitting rule: Gini;
- Max depth=4, cioè la massima profondità dell'albero, usata come stopping criterion;

e possiamo notare come abbia raggiunto buone performance, seppur peggiori della regressione logistica e con una forma della curva ROC diversa.

3.3 Random forest

La tecnica del random forest è un'estensione del decision tree e prevede di aggregare insieme un certo numero di alberi in un unico strong learner. L'idea alla base, avendo B dataset indipendenti su cui fare training, è di allenare B alberi e poi creare un modello medio. Secondo la teoria, se ognuna delle predizioni avesse varianza σ^2, la varianza delle predizioni ottenute dal modello finale si ridurrebbe a $\frac{\sigma^2}{B}$.

Nel nostro caso abbiamo un solo dataset a disposizione e utilizzando la tecnica del bootstrapping ne possiamo generare altri da quest'ultimo; i campioni generati, però, non saranno indipendenti. Di conseguenza la varianza finale dipenderà dalla correlazione tra i vari learner. Per questo con la random forest si vuole applicare il metodo del bagging cercando di ridurre la correlazione tra gli alberi. Praticamente questo viene implementato scegliendo randomicamente, per ogni albero, solo una parte delle features su cui il learner sarà allenato. Un'alternativa più efficace, ma anche più onerosa, sarebbe scegliere randomicamente le features ad ogni split di ogni singolo albero.

Riporto, adesso, i risultati ottenuti valutando il modello con i parametri migliori sul test set.

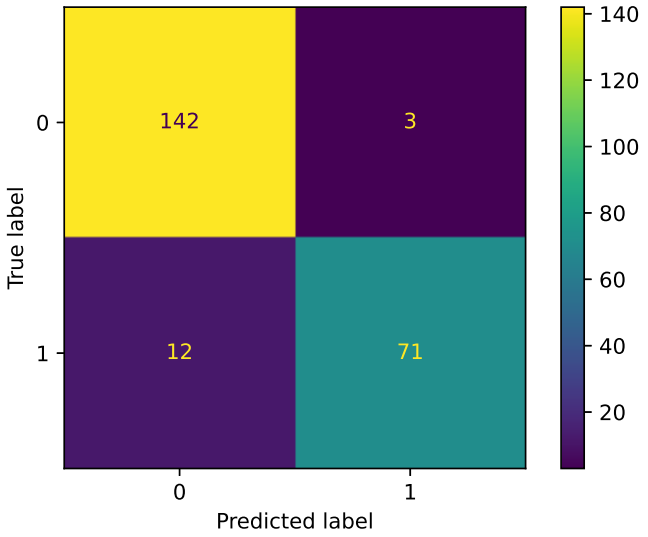


Figura 11: Confusion matrix del modello Forest

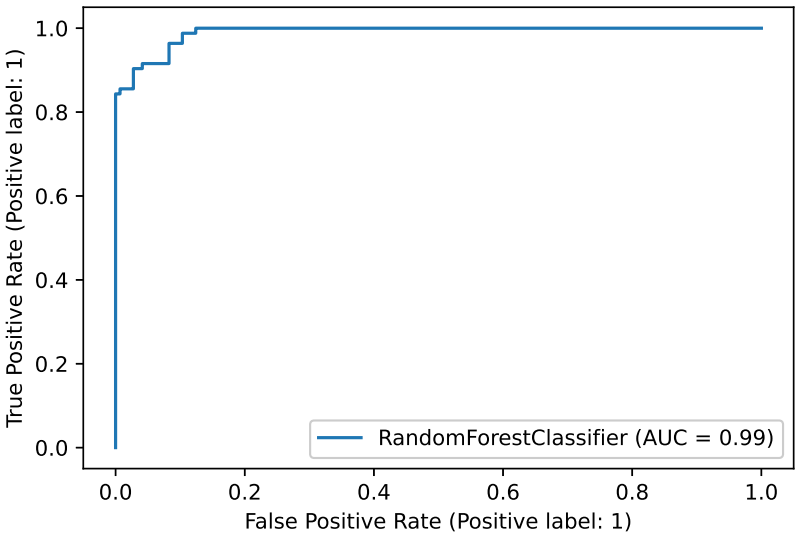


Figura 12: ROC del modello Forest

Accuracy	Precision	Recall	F1
0.934	0.959	0.855	0.904

Tabella 4: Risultati del modello Forest

Il modello riportato ha utilizzato i seguenti iper-parametri:

- Splitting rule: Entropy;
- Max depth=3, cioè la massima profondità del singolo albero;
- Numero di alberi: 20;

e possiamo notare come abbia raggiunto ottime performance, sicuramente migliori del semplice decision tree.

3.4 SVM

La tecnica support vector machine prevede la creazione di un iperpiano nello spazio delle features in modo da separare i dati in due regioni. In particolare l'idea è di trovare il piano che massimizzi la distanza minima con i dati, chiamato margine, in modo che le classi siano ben separate. Per farlo è necessario risolvere un problema di ottimizzazione nella forma seguente:

$$\begin{aligned} &maximize \;\; M(\theta) \\ &subject\, to \;\; ||\theta||^2 = 1 \\ &y_i(\theta' x_i) \geq M, \;\; i = 1, ..., n; \end{aligned}$$

dove θ è il vettore dei parametri del piano, y_i la singola osservazione, x_i il singolo vettore delle features e M il margine. Tale problema viene risolto seguendo la formulazione duale tramite i moltiplicatori di Lagrange. È interessante notare, dalla soluzione del problema, che solo i support vector, cioè quelli più vicini all'iperpiano, influenzano la scelta dei parametri.

Fondamentale perchè l'algoritmo funzioni è che i dati siano linearmente separabili, cioè che esista un iperpiano in grado di dividerli:

$$\exists \theta \in \mathbf{R}^{d+1} \,|\, y_i(\theta' x_i) \geq 0, \;\; i = 1, ..., n.$$

Notiamo che la dimensione dei parametri è $d + 1$, dove d è il numero delle features, poichè si è soliti aggiungere una feature "fittizia", cioè $x_0 = \mathbf{1}$, in modo da scrivere l'iperpiano nella forma $< x_i, \theta >= 0$. Nel caso in cui le classi non siano linearmente separabili è possibile introdurre un termine di rilassamento, che funge da regolarizzatore, permettendo ad alcuni punti di oltrepassare il piano.

Riporto, adesso, i risultati ottenuti valutando il modello con i parametri migliori sul test set.

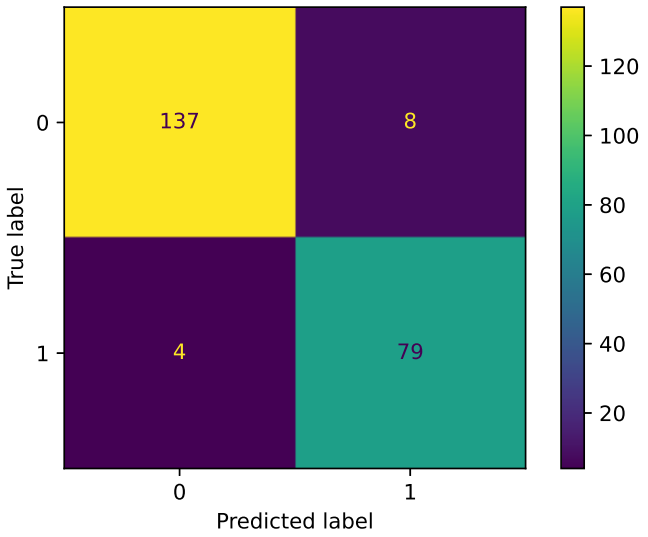


Figura 13: Confusion matrix del modello SVM

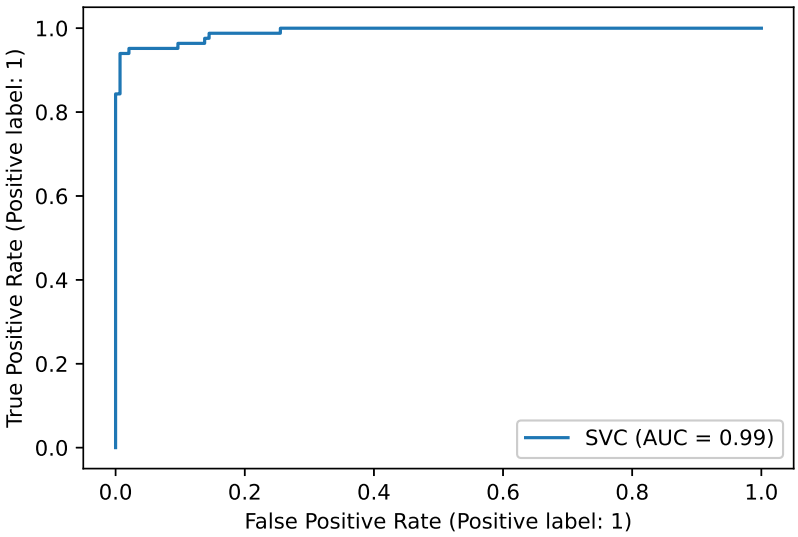


Figura 14: ROC del modello SVM

Accuracy	Precision	Recall	F1
0.947	0.908	0.952	0.929

Tabella 5: Risultati del modello SVM

Il modello riportato ha utilizzato i seguenti iper-parametri:

- C=0.1;
- Kernel: lineare;

e possiamo notare come abbia raggiunto ottime performance, soprattutto per la metrica recall che, data la natura del dataset, è quella che ci interessa di più.

3.5 KNN

L'algoritmo k-nearest neighbours si basa sul concetto di vicinanza tra i punti del dataset. Immaginando, infatti, i dati come punti nello spazio è possibile calcolare la distanza fra di essi. Scelto un metodo per misurare la distanza, ad esempio la norma euclidea, l'algoritmo assegna ad un nuovo dato l'etichetta della maggioranza dei k punti a lui più vicini.

Il training del modello è sostanzialmente nullo; i calcoli, infatti, vengono tutti svolti in fase di testing.

Riporto, adesso, i risultati ottenuti valutando il modello con i parametri migliori sul test set.

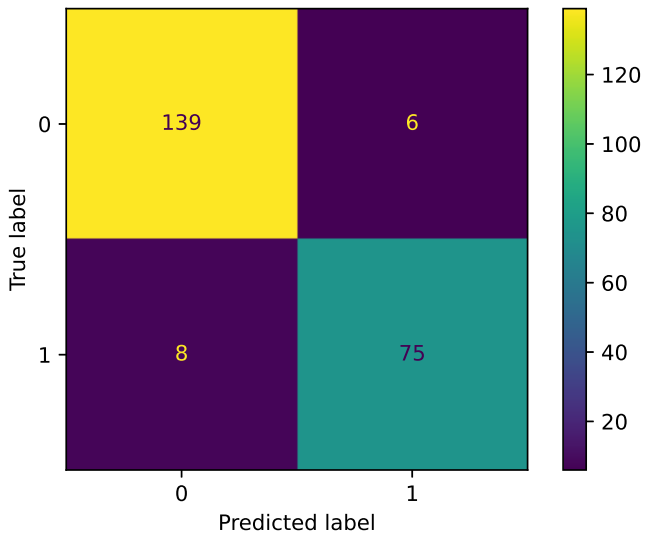


Figura 15: Confusion matrix del modello KNN

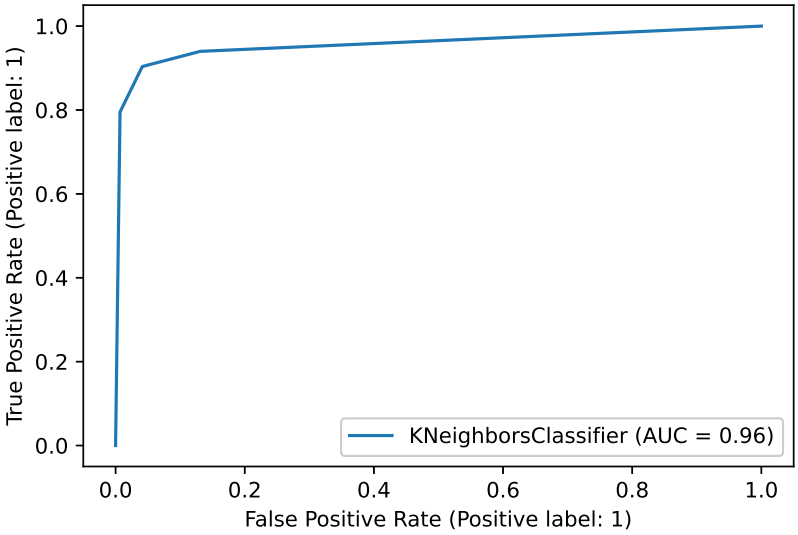


Figura 16: ROC del modello KNN

Accuracy	Precision	Recall	F1
0.939	0.926	0.904	0.915

Tabella 6: Risultati del modello KNN

Il modello riportato ha utilizzato i seguenti iper-parametri:

- Numero vicini=3, cioè k;

e possiamo notare come abbia raggiunto ottime performance, nonostante la grande semplicità.

4 Classificazione con riduzione dimensionale

Adesso voglio applicare le tecniche di classificazione al dataset ridotto tramite PCA per vedere come cambiano le metriche dei modelli. In particolare ho scelto di ridurre il dataset ad una sola componente principale che, da sola, riesce a spiegare il 97.98% della varianza.

Voglio sottolineare come l'algoritmo della PCA abbia usato solo la matrice del training set per generare il sottospazio di trasformazione. Il test set è stato poi proiettato in questo sottospazio già creato in precedenza tramite il training set. Inoltre ho utilizzato per i vari modelli gli iper-parametri che hanno performato meglio nella precedente grid search, in modo da valutare la differenza di prestazione delle singole tecniche. Qui riporto i risultati principali della fase di testing.

Modelli senza PCA				
Modello	Accuracy	Precision	Recall	F1
Logit	0.952	0.929	0.940	0.934
Tree	0.912	0.909	0.843	0.875
Forest	0.934	0.959	0.855	0.904
SVM	0.947	0.908	0.952	0.929
KNN	0.939	0.926	0.904	0.915

Modelli con PCA				
Modello	Accuracy	Precision	Recall	F1
Logit	0.917	0.932	0.831	0.879
Tree	0.912	0.970	0.783	0.867
Forest	0.930	0.972	0.831	0.896
SVM	0.921	0.945	0.831	0.885
KNN	0.908	0.908	0.831	0.868

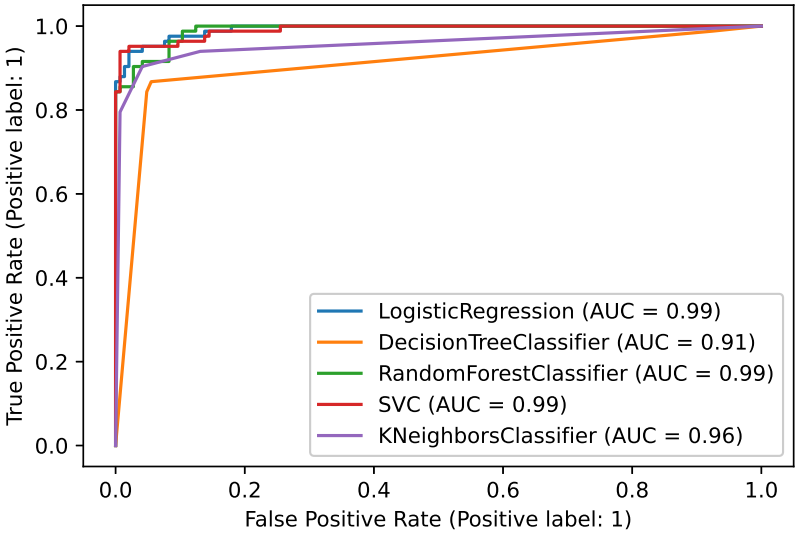


Figura 17: Confronto delle curve ROC senza PCA

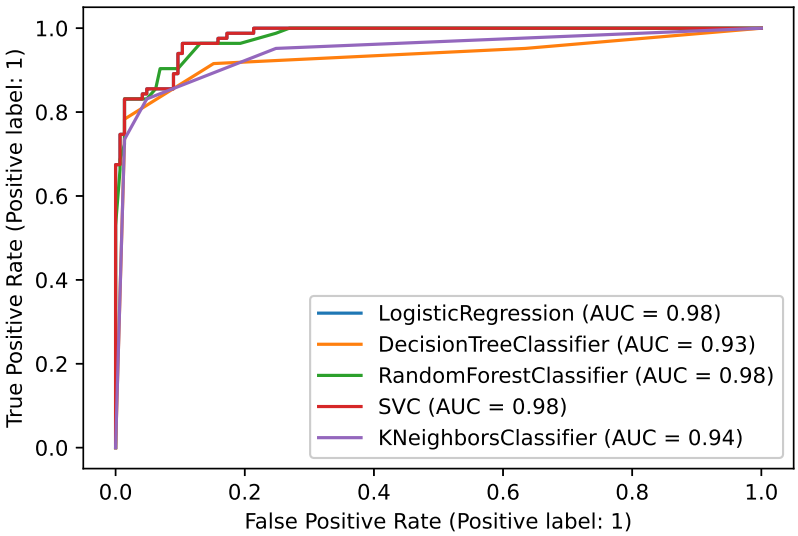


Figura 18: Confronto delle curve ROC con PCA

Come era prevedibile tutti i modelli hanno visto un peggioramento di quasi tutte le metriche di valutazione. Ciò è dovuto a una minore varianza spiegata dal nuovo dataset. Notiamo, inoltre, che il peggioramento delle prestazioni si nota soprattutto sulla metrica *recall*, che fa riferimento ai falsi negativi. Questo dipende dal fatto che il dataset contiene la maggioranza degli elementi appartenenti alla classe **B**. Quindi i modelli sono, generalmente, più propensi a etichettare un elemento come appartenente a tale classe; questa tendenza si accentua nel momento in cui diminuiamo la varianza spiegata dal dataset.

5 Conclusioni

Si può affermare che l'obbiettivo di partenza è stato raggiunto con ottimi risultati. Tutti i modelli utilizzati, infatti, hanno raggiunto un'accuratezza maggiore del 90%. In particolare, il modello che risulta più efficace per la natura del dataset è sicuramente l'SVM che raggiunge un livello di *recall* del 95%, ottimo per la diagnosi. Anche la riduzione dimensionale con la PCA ha avuto ottime performance nel dataset, permettendo di eliminare ben 29 feature a fronte di una riduzione massima del 4% sull'accuratezza. In particolare l'algoritmo random forest ha avuto i risultati migliori sul dataset ridotto, con un valore di $F1 = 89.6\%$.

6 Codice

6.1 Exploratory analysis su R

```
rm(list = ls())
library(tidyverse)
library(corrplot)
library(ggplot2)

df<-data[,2:32] #elimino colonna id

head(df) #prime righe
summary(df) #sommario

ggplot(df, aes(x=reorder(df$diagnosis, df$diagnosis, function(x)-
    length(x)))) +
  geom_bar() +  labs(x='Class') #plot della classe diagnosi

X<-df[,2:31]
y<-df[,1]

m=data.matrix(X)
corr<-cor(m) #correlazione
sd <- apply(m, 2, sd)  #standard deviation
corrplot(corr, method="circle") #correlogramma
pairs(m) #pairsplot

boxplot(m) #boxplot generale, inutilizzabile

par(mfrow=c(2,3)) #divido lo schermo in 6 parti

for(i in 2:3){ #disegno i boxplot separatmente
  for(j in 1:3){
    col=i+10*(j-1)
  a4<-df[df$diagnosis=='M',col]
  a4=data.matrix(a4)
  a2<-df[df$diagnosis=='B',col]
  a2=data.matrix(a2)
  a=list(a4,a2)
  names(a) <- c(paste("M"), paste("B"))
  boxplot(a,ylab=colnames(X)[col-1])
  }
}

for(i in 4:5){
  for(j in 1:3){
    col=i+10*(j-1)
    a4<-df[df$diagnosis=='M',col]
    a4=data.matrix(a4)
    a2<-df[df$diagnosis=='B',col]
    a2=data.matrix(a2)
    a=list(a4,a2)
    names(a) <- c(paste("M"), paste("B"))
    boxplot(a,ylab=colnames(X)[col-1])
  }
}
for(i in 6:7){
```

```
  for(j in 1:3){
    col=i+10*(j-1)
    a4<-df[df$diagnosis=='M',col]
    a4=data.matrix(a4)
    a2<-df[df$diagnosis=='B',col]
    a2=data.matrix(a2)
    a=list(a4,a2)
    names(a) <- c(paste("M"), paste("B"))
    boxplot(a,ylab=colnames(X)[col-1])
  }
}
for(i in 8:9){
  for(j in 1:3){
    col=i+10*(j-1)
    a4<-df[df$diagnosis=='M',col]
    a4=data.matrix(a4)
    a2<-df[df$diagnosis=='B',col]
    a2=data.matrix(a2)
    a=list(a4,a2)
    names(a) <- c(paste("M"), paste("B"))
    boxplot(a,ylab=colnames(X)[col-1])
  }
}
for(i in 10:11){
  for(j in 1:3){
    col=i+10*(j-1)
    a4<-df[df$diagnosis=='M',col]
    a4=data.matrix(a4)
    a2<-df[df$diagnosis=='B',col]
    a2=data.matrix(a2)
    a=list(a4,a2)
    names(a) <- c(paste("M"), paste("B"))
    boxplot(a,ylab=colnames(X)[col-1])
  }
}
```

6.2 Classificazione su Python

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import sklearn.model_selection as ms
import sklearn.linear_model as lm
import sklearn.metrics as met
import sklearn.tree as tree
import sklearn.ensemble as ensemble
import sklearn.svm as svm
import sklearn.neighbors as neig
import sklearn.decomposition as dec
from google.colab import files

df=pd.read_csv("data.csv")
df=df.iloc[:,1:32] #elimino la colonna id
y = df.iloc[:,0]
X= df.iloc[:,1:32]

```

```
X=X.to_numpy()
y=y.to_numpy() #trasformo il dataset in matrici

for i in range(y.size):
    if y[i]=='B':
        y[i]=0

    if y[i]=='M':
        y[i]=1

y=y.astype('int') #trasformo y in vettore binario

X_train, X_test, y_train, y_test = ms.train_test_split(X, y,
    test_size=0.4,
random_state=0) #divido training e testing set

reduction=0 #flag per applicare pca al dataset
if reduction:
  pca=dec.PCA(n_components=1)
  pca2=dec.PCA(n_components=5)
  pca2.fit(X_train)
  X_train=pca.fit_transform(X_train)
  X_test=pca.transform(X_test)
  print(pca2.explained_variance_ratio_)

imm=0 #flag per salvare le immagini

def classifier(par,est): #funzione che crea il modello migliore
    gs = ms.GridSearchCV(estimator=est,cv=4,param_grid=par,
scoring='f1').fit(X_train, y_train) #grid search sugli iper-
    parametri
    model = gs.best_estimator_  #salvo modello migliore trainato su
     tutto set
    y_pred=gs.predict(X_test) #faccio previsione col modello
    migliore
    a=met.accuracy_score(y_test,y_pred) #calcolo metriche
    r=met.recall_score(y_test,y_pred)
    p=met.precision_score(y_test,y_pred)
    f=met.f1_score(y_test,y_pred)
    met.ConfusionMatrixDisplay.from_estimator(model,X_test,y_test)
    plt.savefig("matrix.eps",format="eps")
    if imm: files.download("matrix.eps")
    met.RocCurveDisplay.from_estimator(model, X_test, y_test)
    plt.savefig("roc.eps",format="eps")
    if imm: files.download("roc.eps")
    print("best parameters", gs.best_params_) #stampo parametri
    migliori
    plt.show()
    a=round(a,3)
    p=round(p,3)
    r=round(r,3)
    f=round(f,3)
    print(a,p,r,f)
    return model

est=lm.LogisticRegression(random_state=0) #creazione del modello
par = {"penalty": ["l2","l1"], 'C': [0.01, 0.1, 1]} #iper-parametri
```

```
    per gs
class_log=classifier(par,est) #applico funzione classifier

#ripeto per tutti i modelli

par= {"criterion": ["gini","entropy"], "max_depth": [3,4,5]}
est=tree.DecisionTreeClassifier(random_state=0)
class_tree=classifier(par,est)
tree.plot_tree(class_tree) #stampo albero creato
plt.savefig("tree.eps",format="eps")
if imm: files.download("tree.eps")

est=ensemble.RandomForestClassifier(random_state=0)
par= {"criterion": ["gini","entropy"],"n_estimators":[10, 20, 30],
      "max_depth": [2,3,4]}
class_forest=classifier(par,est)

est=svm.SVC(random_state=0)
par={'C':[0.01, 0.1, 1], "kernel":["linear","poly","sigmoid","rbf"
    ]}
class_svm=classifier(par,est)

est=neig.KNeighborsClassifier()
par={"n_neighbors":[3, 5, 7]}
class_knn=classifier(par,est)

disp=met.plot_roc_curve(class_log,X_test,y_test) #disegno tutte le
    curve roc
met.plot_roc_curve(class_tree,X_test,y_test,ax=disp.ax_)
met.plot_roc_curve(class_forest,X_test,y_test,ax=disp.ax_)
met.plot_roc_curve(class_svm,X_test,y_test,ax=disp.ax_)
met.plot_roc_curve(class_knn,X_test,y_test,ax=disp.ax_)
plt.savefig("confronto.eps",format="eps")
files.download("confronto.eps")









#parte relativa all'analisi con PCA


#creo tutti i modelli con i migliori iper-parametri
class_log=lm.LogisticRegression(C=0.1,random_state=0)
class_tree=tree.DecisionTreeClassifier(criterion="gini",max_depth
    =4,
                                       random_state=0)
class_forest=ensemble.RandomForestClassifier(criterion="entropy",
max_depth=3, n_estimators=20, random_state=0)
class_svm=svm.SVC(kernel='linear', C =0.1,random_state=0)
class_knn=neig.KNeighborsClassifier(n_neighbors=3)

#creo funzione per allenare e testare sul dataset ridotto
def evaluation(model):
  clf=model.fit(X_train,y_train) #alleno modelli su training set
    ridotto
  y_pred = clf.predict(X_test) #valuto modelli su test set ridotto
```

```
  print('{:3.3f} {:3.3f} {:3.3f} {:3.3f}'.format(
      met.accuracy_score(y_test,y_pred),met.precision_score(y_test,
    y_pred),
      met.recall_score(y_test,y_pred), met.f1_score(y_test,y_pred))
    )

#testo tutti i modelli
evaluation(class_log)
evaluation(class_tree)
evaluation(class_forest)
evaluation(class_svm)
evaluation(class_knn)

disp=met.plot_roc_curve(class_log,X_test,y_test) #disegno curve roc
     per tutti i modelli
met.plot_roc_curve(class_tree,X_test,y_test,ax=disp.ax_)
met.plot_roc_curve(class_forest,X_test,y_test,ax=disp.ax_)
met.plot_roc_curve(class_svm,X_test,y_test,ax=disp.ax_)
met.plot_roc_curve(class_knn,X_test,y_test,ax=disp.ax_)
plt.savefig("confronto_pca.eps",format="eps")
files.download("confronto_pca.eps")
```