**Academic Year:** 2023/2024

**Course:** Artificial Intelligent Systems - Intelligent Model Project

**Prof. Alfredo Milani**

**Ongoing assignmen**t: Machine Learning Basics

**Student:** Lorenzo Mariotti

**ID:** 369094

Sommario

[1. Dataset 3](#_Toc155453836)

[2. Classificatori 5](#_Toc155453837)

[2.1. Decision Tree 6](#_Toc155453838)

[2.2. Nearest Neighbor 7](#_Toc155453839)

[2.3. Random Forest 8](#_Toc155453840)

[2.4. AdaBoost 9](#_Toc155453841)

[3. Comparazione 10](#_Toc155453842)

# Dataset

**Link:**

<https://www.kaggle.com/datasets/fedesoriano/company-bankruptcy-prediction/data>

**Descrizione:**

Il dataset riporta i dati sulla bancarotta di aziende tratti dal *Taiwan Economic Journal* per gli anni 1999-2009. Il fallimento dell'azienda è stato definito in base al regolamento aziendale della Borsa di Taiwan.

**Attributi:**

Il dataset non presenta alcun attributo categorico quindi non è necessaria alcuna fattorizzazione.

Va tenuto però da conto della scala degli attributi, mentre la maggior parte di essi sono rappresentati da valori numerici compresi nell’intervallo [0, 1] altri come ad esempio:

* “Operating Expense Rate”,
* “Research and development expense rate”
* ”Cash flow rate Interest-bearing debt interest rate”
* …

sono rappresentati da valori compresi nell’intervallo [0, +Inf]. Tale differenza influenza in modo estremamente negativo classificatori come il K-NN pertanto si è deciso di ridurne la scala riportandola a dei valori congrui al resto degli attributi.

**Resize** **degli attributi:**

Lo script “*Preprocessing.cpp*” si occupa di scalare ogni attributo portando ogni suo valore nell’intervallo (0, 1).

for (int i = 1; i < attributes.size(); i++)

{

double max = stod(\*max\_element(attributes[i].begin(), attributes[i].end()));

if (max > 1)

{

for (int j = 1; j < attributes[i].size(); j++)

{

double x = stod(attributes[i][j]);

x /= max;

attributes[i][j] = to\_string(x);

}

}

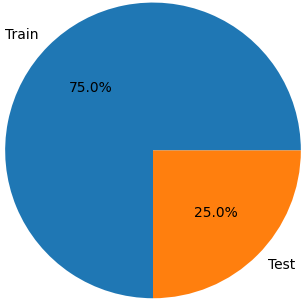
}

**Classi:**

Le classi d’interesse sono rappresentate dalla colonna *“Bankrupt?”* che riporta con valori booleani se una data condizione ha portato o meno alla bancarotta dell’azienda.

**Training:**

Il training del modello è stato eseguito utilizzando il 75% dei campioni per il training e il restante 25% per il test.

****

**Distribuzione delle classi:**

|  |  |
| --- | --- |
| Sampling standard | Resampling |
|  |  |
| |  |  | | --- | --- | | Classe | Occorrenze | | 0 | 4952 | | 1 | 162 | | |  |  | | --- | --- | | Classe | Occorrenze | | 0 | 5114 | | 1 | 5114 | |

Con uno sbilanciamento tale di classi metriche come l’accuratezza risultano forvianti sono quindi da favorire metriche come la **precisione** o il **recall score**.

Per arginare questo sbilanciamento si è deciso di eseguire un **resampling** dei dati in ingresso, tale tecnica ci permette di migliorare la rappresentazione delle classi con occorrenze minori.

**Resempling utilizzato:**

1. **Oversampling** della classe con meno occorrenze;
2. **Undersampling** della classe con più occorrenze;
3. Concatenazione dei risultati dei punti 1-2;

x\_train, x\_test, y\_train, y\_test = train\_test\_split(

data,

target,

test\_size=test\_perc,

random\_state=42

)

# Check for sampling condition

if(sampling == "resample"):

re\_samp = RandomOverSampler(

sampling\_strategy = 'minority',

random\_state=42

)

x\_over, y\_over = re\_samp.fit\_resample(x\_train, y\_train)

re\_samp = RandomUnderSampler(

sampling\_strategy = 'majority',

random\_state=42

)

x\_under, y\_under = re\_samp.fit\_resample(x\_train, y\_train)

x\_train = pd.concat([x\_over, x\_under], ignore\_index=True)

y\_train = pd.concat([y\_over, y\_under], ignore\_index=True)

# Classificatori

I classificatori analizzati sono stati:

* Decision Tree
* Nearest-neighbor
* Random forest
* AdaBoost

## Decision Tree

def Use\_decision\_tree\_classifier(

x\_train, x\_test, y\_train, y\_test,

metric,

export = False):

classifier = DecisionTreeClassifier(

class\_weight = Get\_classes\_wgt(y\_train),

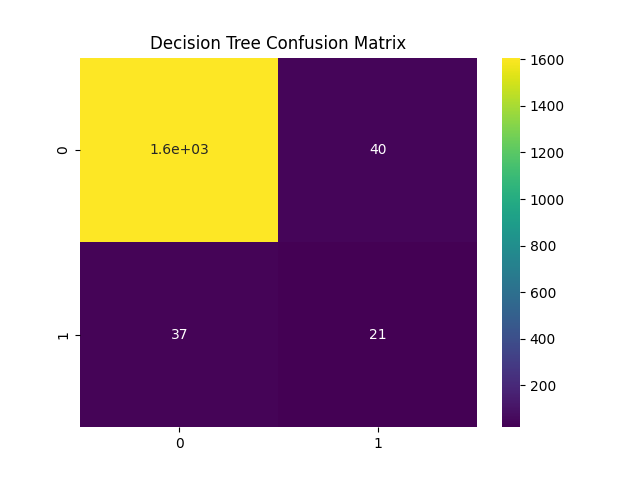
random\_state = 42

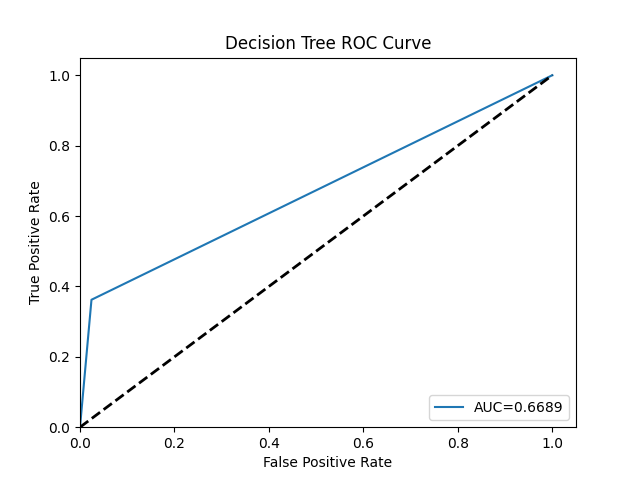
)

classifier.fit(x\_train, y\_train)

predictions = classifier.predict(x\_test)

Compute\_metrics(classifier, predictions, x\_test, y\_test, metric, export)





## Nearest Neighbor

def Use\_knn\_classifier(

x\_train, x\_test, y\_train, y\_test,

metric,

export = False):

classifier = KNeighborsClassifier(

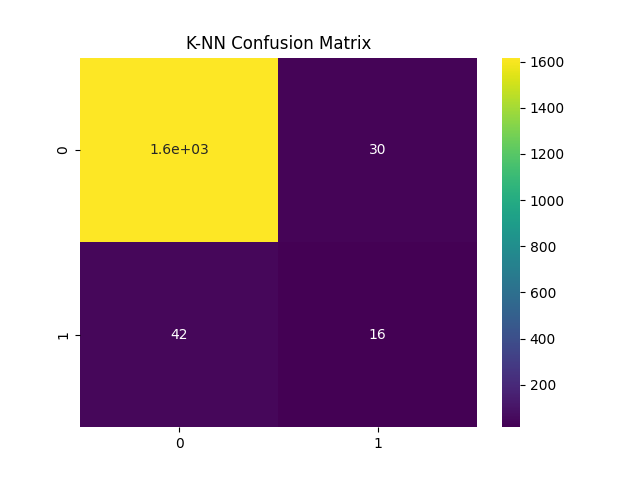
n\_neighbors=2

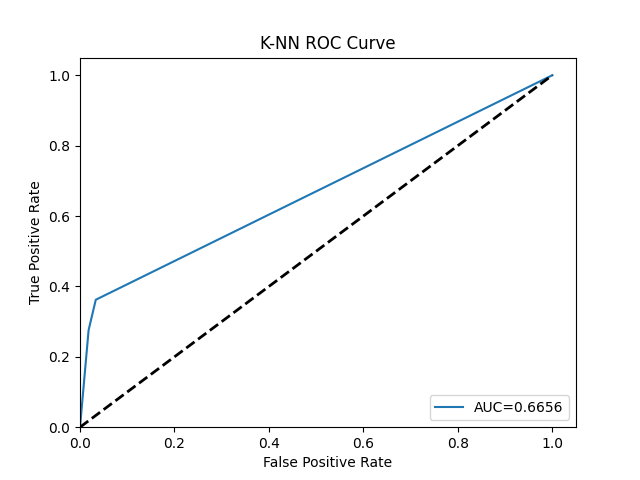
)

classifier.fit(x\_train, y\_train)

predictions = classifier.predict(x\_test)

Compute\_metrics(classifier, predictions, x\_test, y\_test, metric, export)





## Random Forest

def Use\_rnd\_forest\_classifier(

x\_train, x\_test, y\_train, y\_test,

metric,

export = False):

classifier = RandomForestClassifier(

n\_estimators=100,

random\_state = 42,

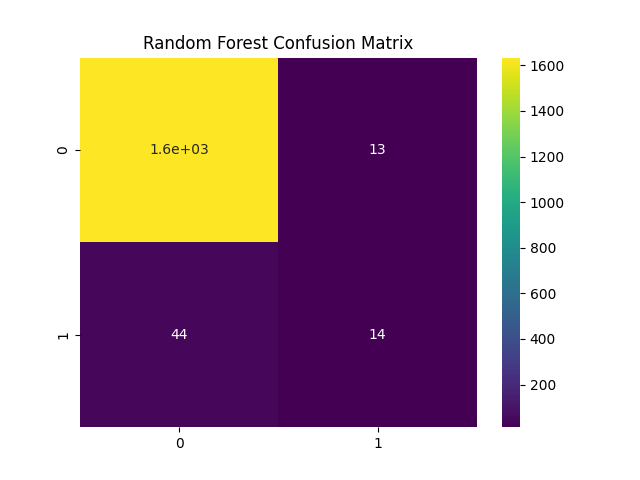
class\_weight = Get\_classes\_wgt(y\_train),

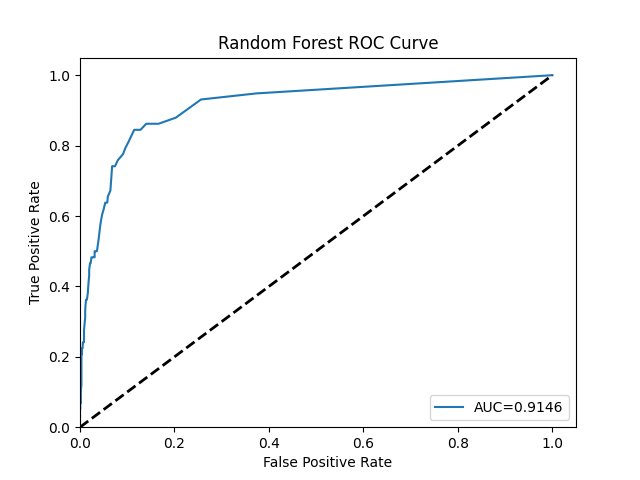
)

classifier.fit(x\_train, y\_train)

predictions = classifier.predict(x\_test)

Compute\_metrics(classifier, predictions, x\_test, y\_test, metric, export)





## AdaBoost

def Use\_adaboost\_classifier(

x\_train, x\_test, y\_train, y\_test,

metric,

export = False):

weak\_learner = RandomForestClassifier(estimators=100, random\_state = 42,

max\_depth = 4, class\_weight = Get\_classes\_wgt(y\_train),

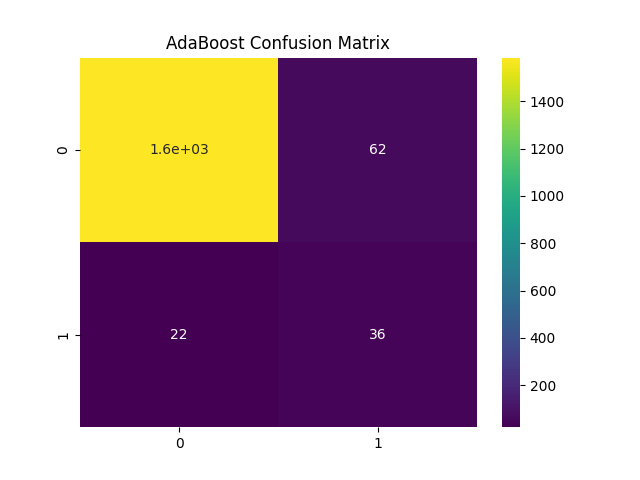
)

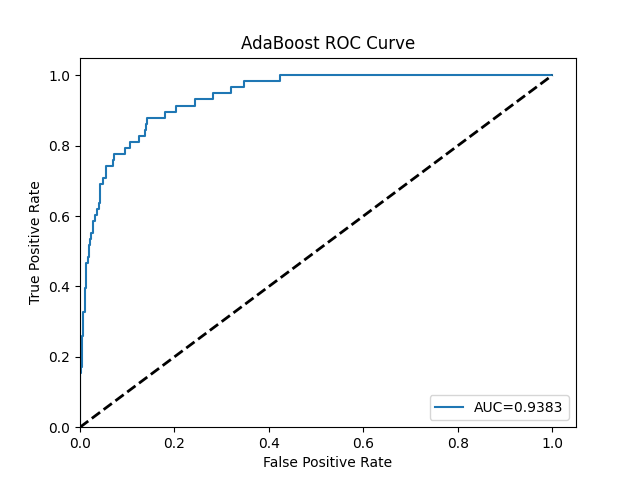
classifier = AdaBoostClassifier(estimator=weak\_learner)

classifier.fit(x\_train, y\_train)

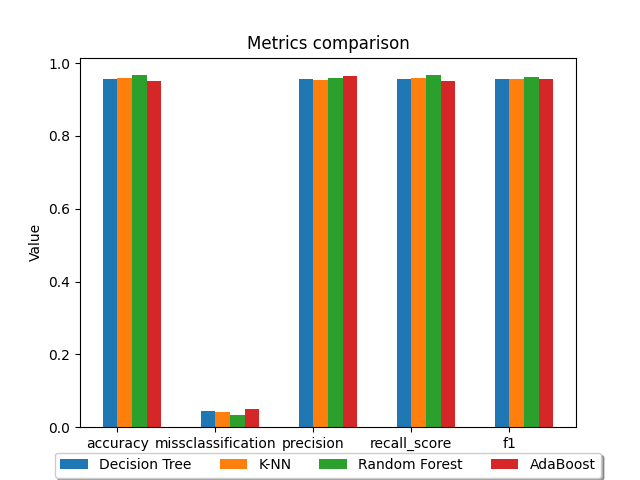
predictions = classifier.predict(x\_test)

Compute\_metrics(classifier, predictions, x\_test, y\_test, metric, export)





# Comparazione



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Classification | accuracy | missclassification | precision | recall\_score | f1 |
| Decision Tree | 0,954839 | 0,045161 | 0,955953 | 0,954839 | 0,955387 |
| K-Nearest Neighbors | 0,957771 | 0,042229 | 0,953359 | 0,957771 | 0,955412 |
| Random Forest | 0,966569 | 0,033431 | 0,958291 | 0,966569 | 0,960629 |
| AdaBoost | 0,950733 | 0,049267 | 0,965254 | 0,950733 | 0,956747 |