Università degli Studi di Perugia
DIPARTIMENTO DI MATEMATICA E INFORMATICA
[LM-18] INFORMATICA

Academic Year: 2023/2024

Course: Artificial Intelligent Systems - Intelligent Model Project

Prof. Alfredo Milani

Ongoing assignment: Machine Learning Basics

Student: Lorenzo Mariotti

ID: 369094

Sommario

1. 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).

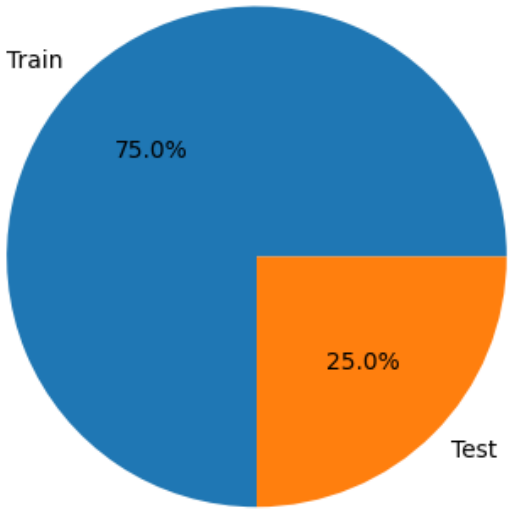
```
for (int i = 1; i < attributes.size(); i++)
{
  double max = stod(*max_element(attributes[i].begin(), attributes[i].end()));

  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.

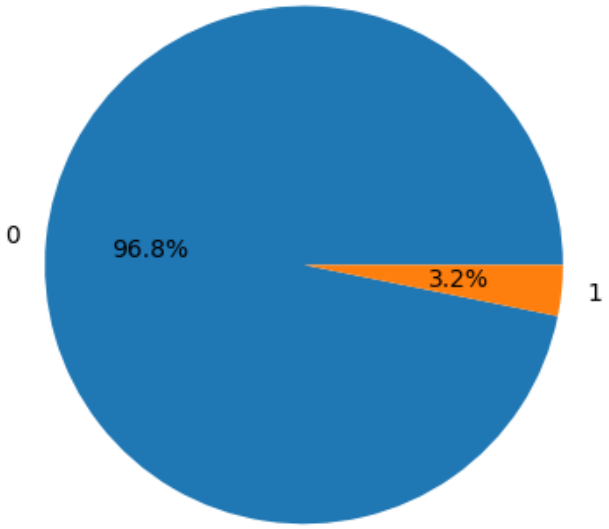
Training:

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



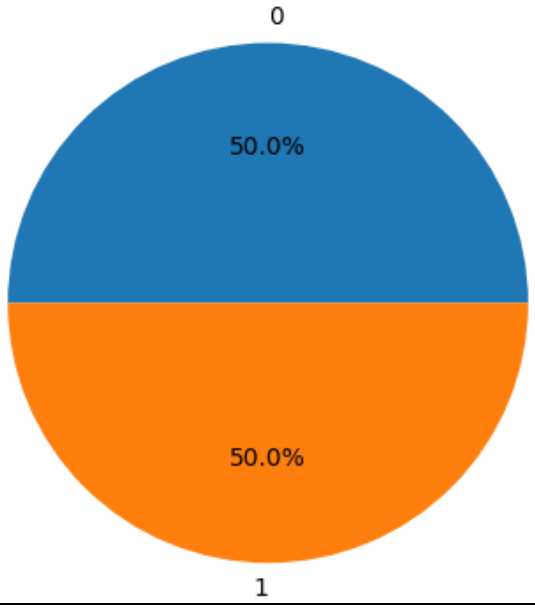
Distribuzione delle classi:

Sampling standard



Classe	Occorrenze
0	4952
1	162

Resampling



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)
```

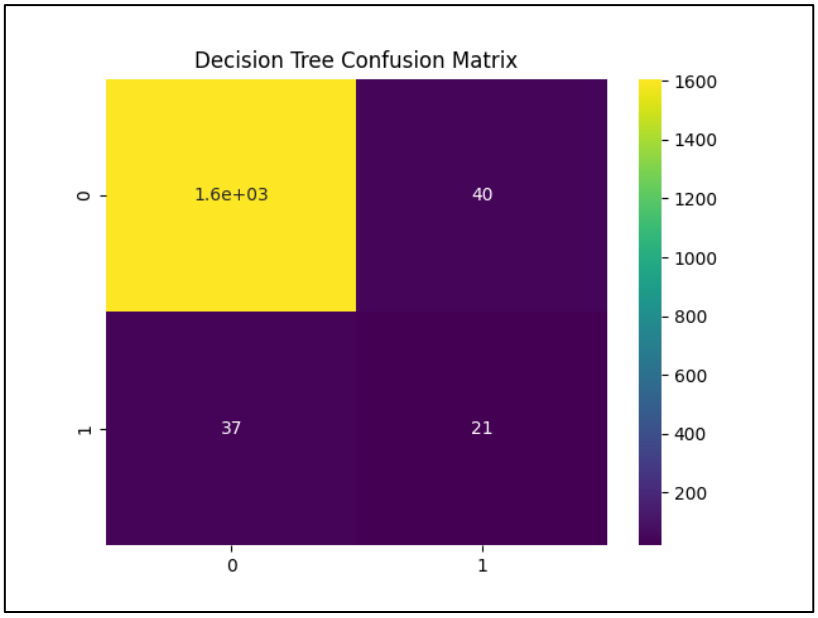
2. Classificatori

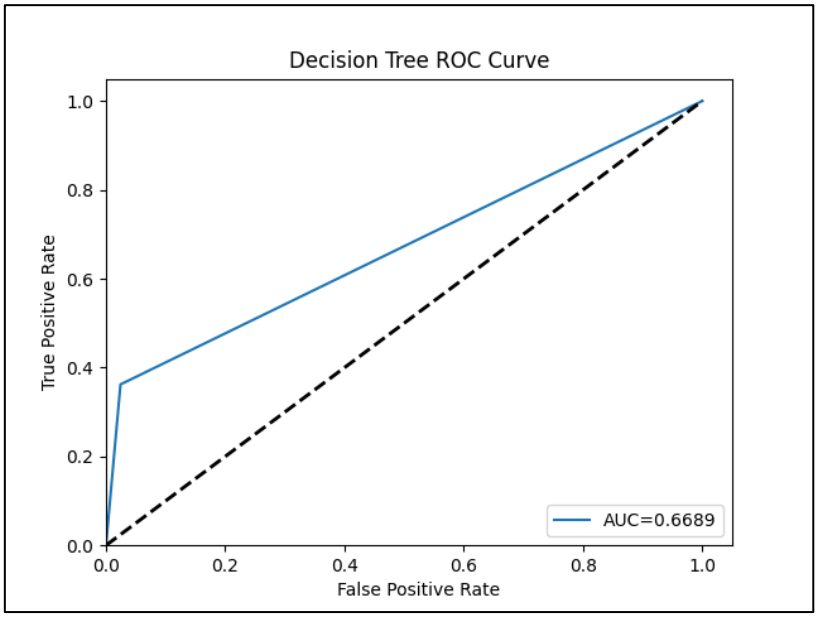
I classificatori analizzati sono stati:

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

2.1. Decision Tree

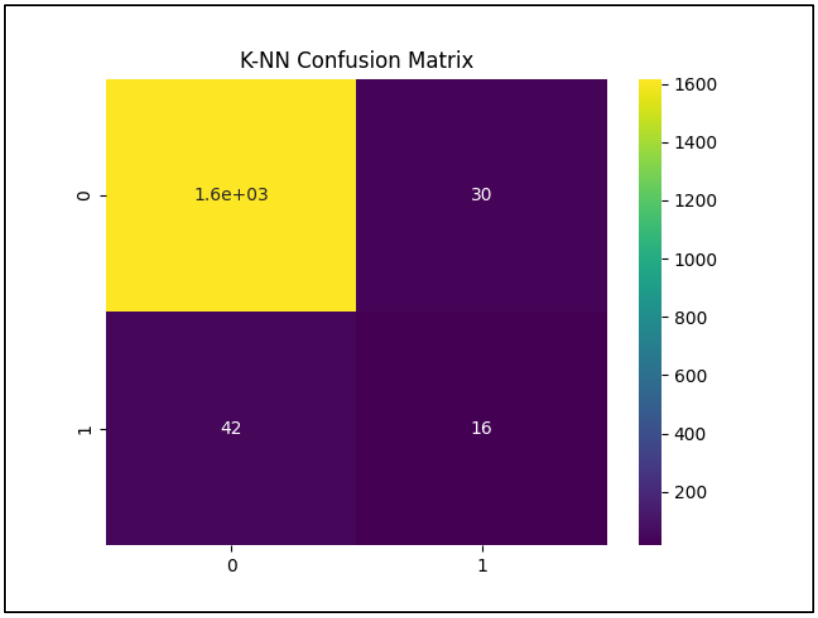
```
def Use_decision_tree_classifier(
        x_train, x_test, y_train, y_test,
        metric,
        export = False):

    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)
```

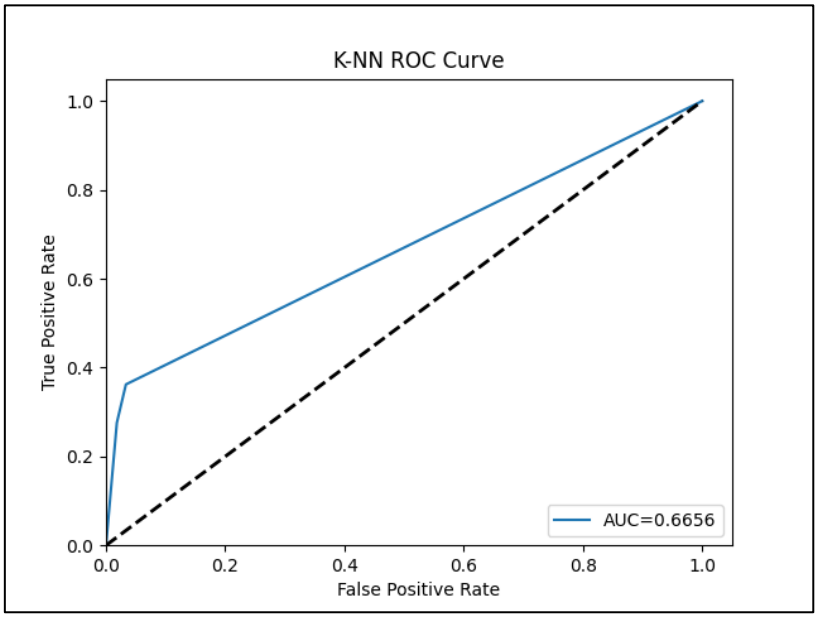




2.2. Nearest Neighbor

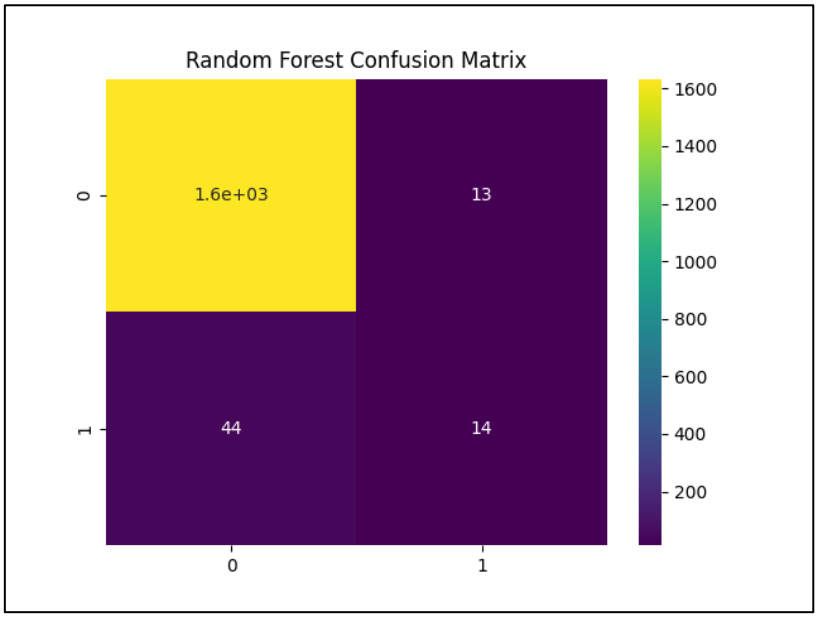
```
def Use_knn_classifier(
        x_train, x_test, y_train, y_test,
        metric,
        export = False):

    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)
```

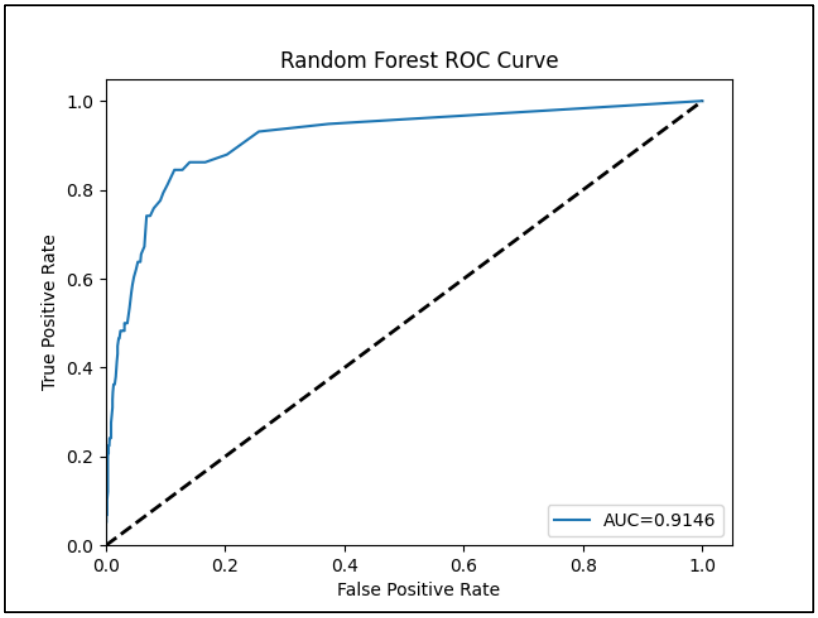




2.3. Random Forest

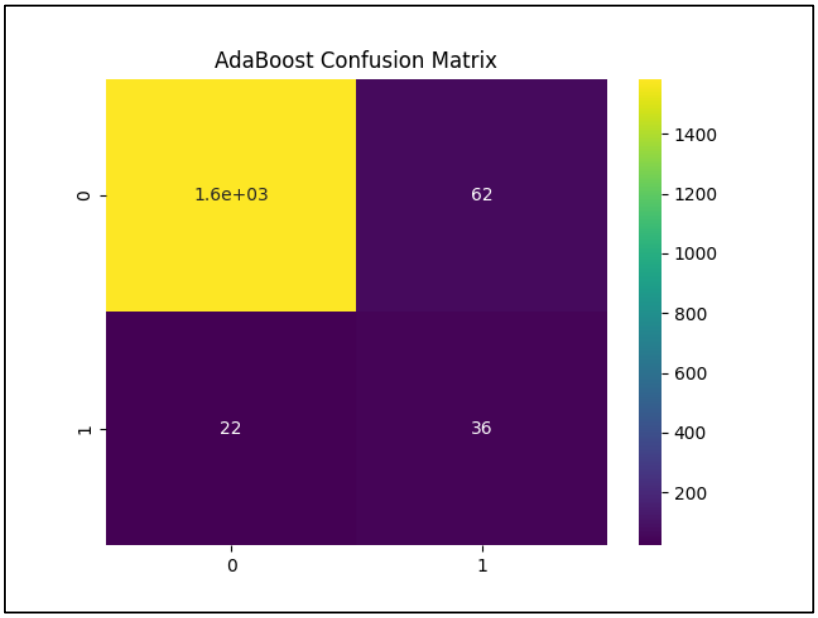
```
def Use_rnd_forest_classifier(
        x_train, x_test, y_train, y_test,
        metric,
        export = False):

    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)
```

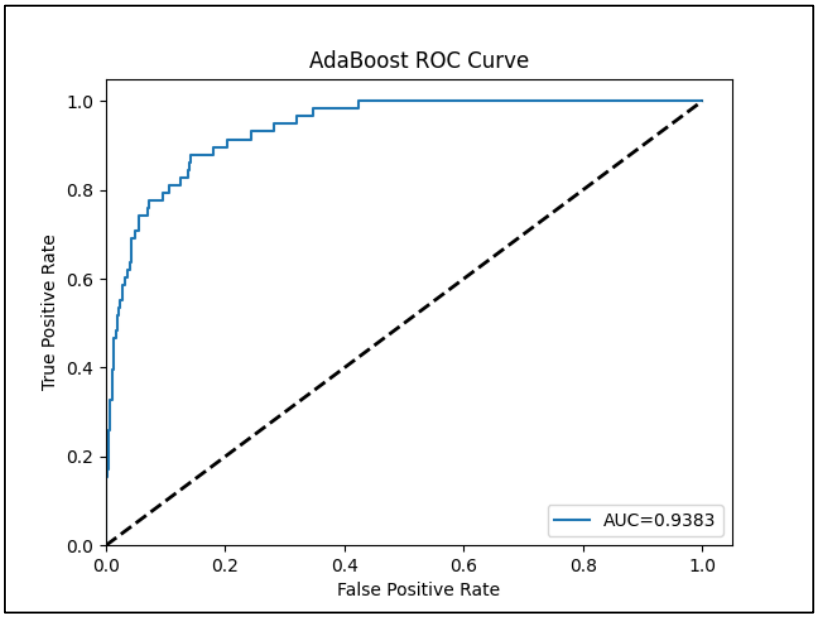




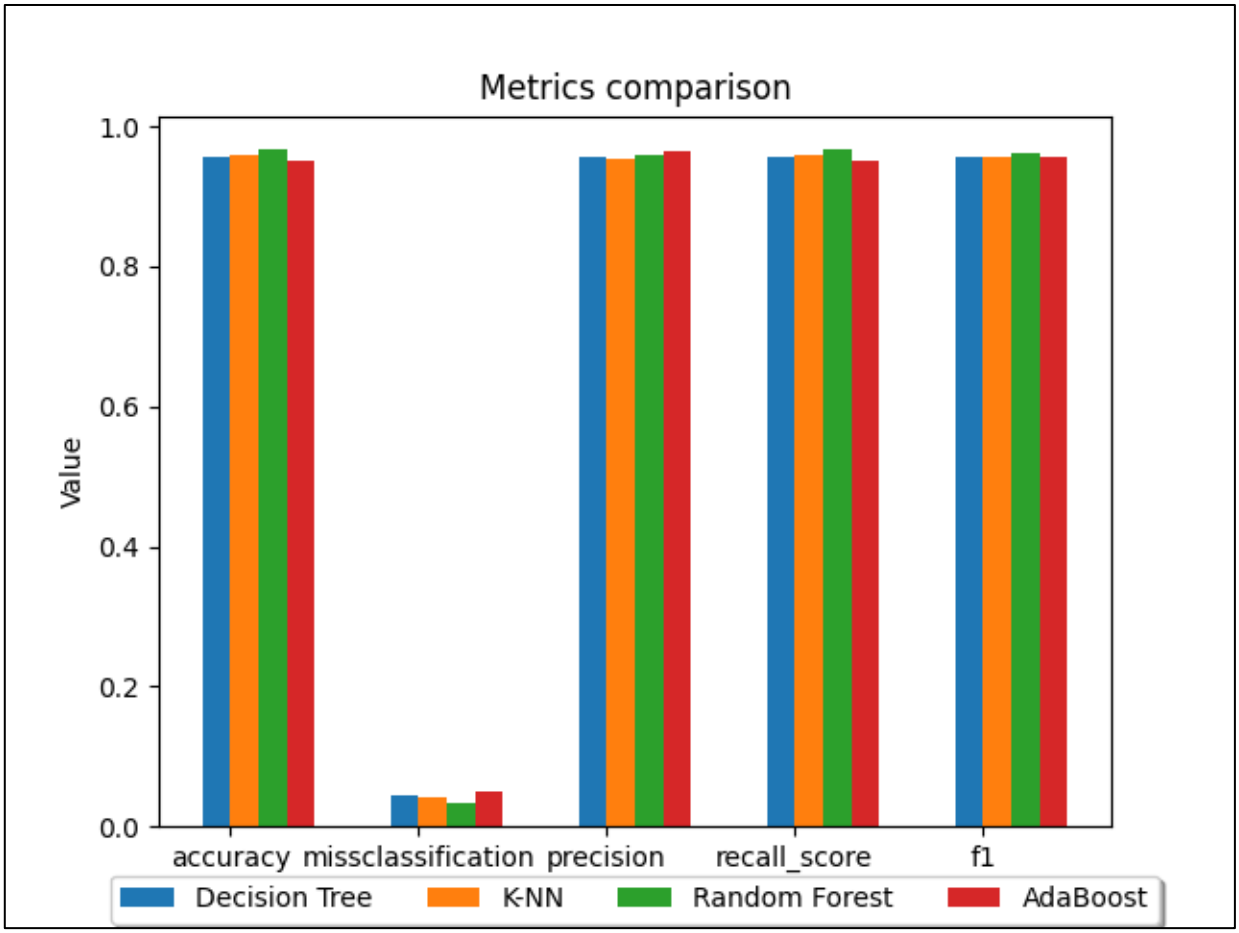
2.4. AdaBoost

```
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)
```





3. 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