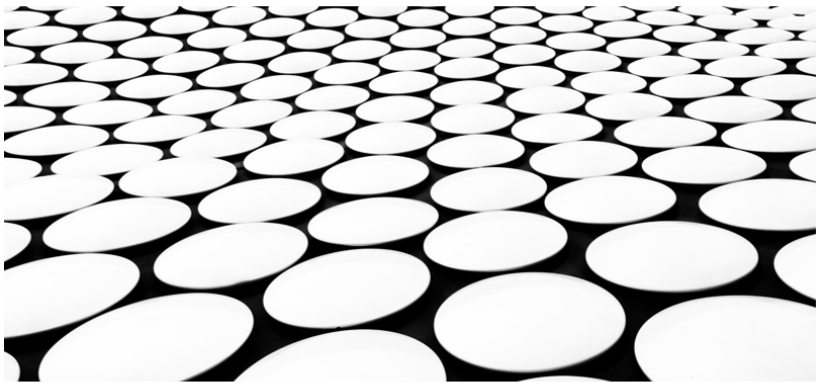


SCHOOL OF AI
ITALIA

MACHINE LEARNING AVANZATO DA ZERO

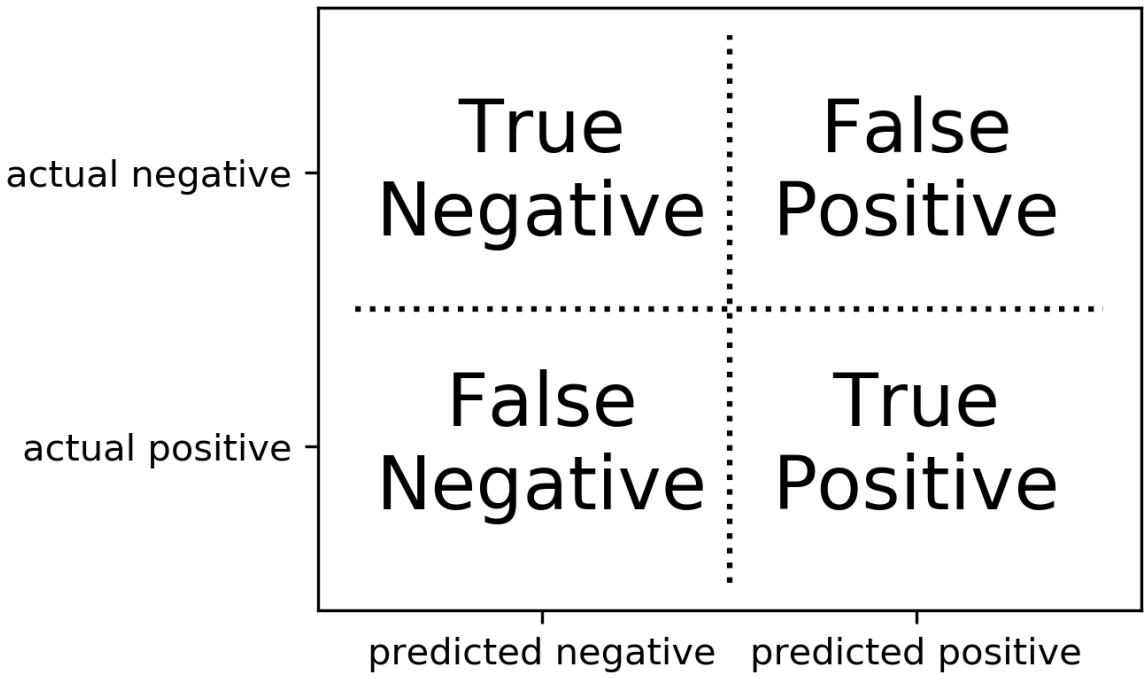
ANTONIO DI CECCO - SCHOOL OF AI

Valutazione dei Modelli

Metriche per la Classificazione binaria

https://scikit-learn.org/stable/modules/model_evaluation.html#confusion-matrix

Confusion matrix

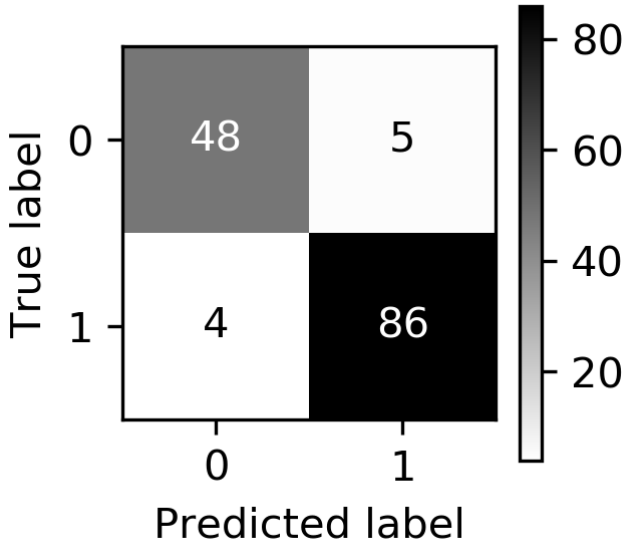


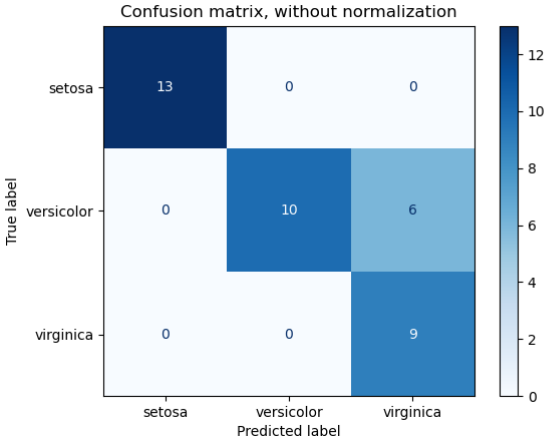
$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$

- Positivo e Negativo sono arbitrari
- Spesso la classe di miniranza è considerata positiva

```
from sklearn.metrics import confusion_matrix, plot_confusion_matrix
data = load_breast_cancer()

X_train, X_test, y_train, y_test = train_test_split(
    data.data, data.target, stratify=data.target, random_state=0)

lr = LogisticRegression().fit(X_train, y_train)
y_pred = lr.predict(X_test)

print(confusion_matrix(y_test, y_pred))
print(lr.score(X_test, y_test))
plot_confusion_matrix(lr, X_test, y_test, cmap='gray_r')
```

```
[[48  5]
 [ 4 86]]
0.94
```



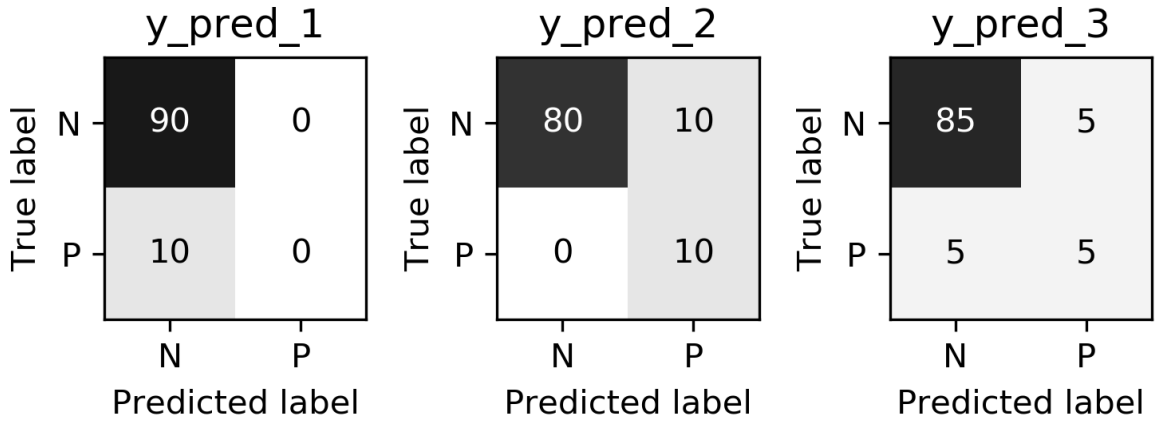


Problemi con l'Accuracy

Dati con il 90% di negativi:

```
from sklearn.metrics import accuracy_score
for y_pred in [y_pred_1, y_pred_2, y_pred_3]:
    print(accuracy_score(y_true, y_pred))
```

```
0.9
0.9
0.9
```



Precision, Recall, f-score

(f1 - score)

$$Precision = \frac{TP}{TP+FP}$$

Positive Predicted Value (PPV)

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

Sensitivity, coverage, true positive rate

$$F_1 = 2\frac{precision \cdot recall}{precision+recall}$$

Media armonica (l'inverso della somma degli inversi) di precisione e recall

$$F_\beta = (1+\beta^2)\frac{precision \cdot recall}{\beta^2 precision+recall}$$

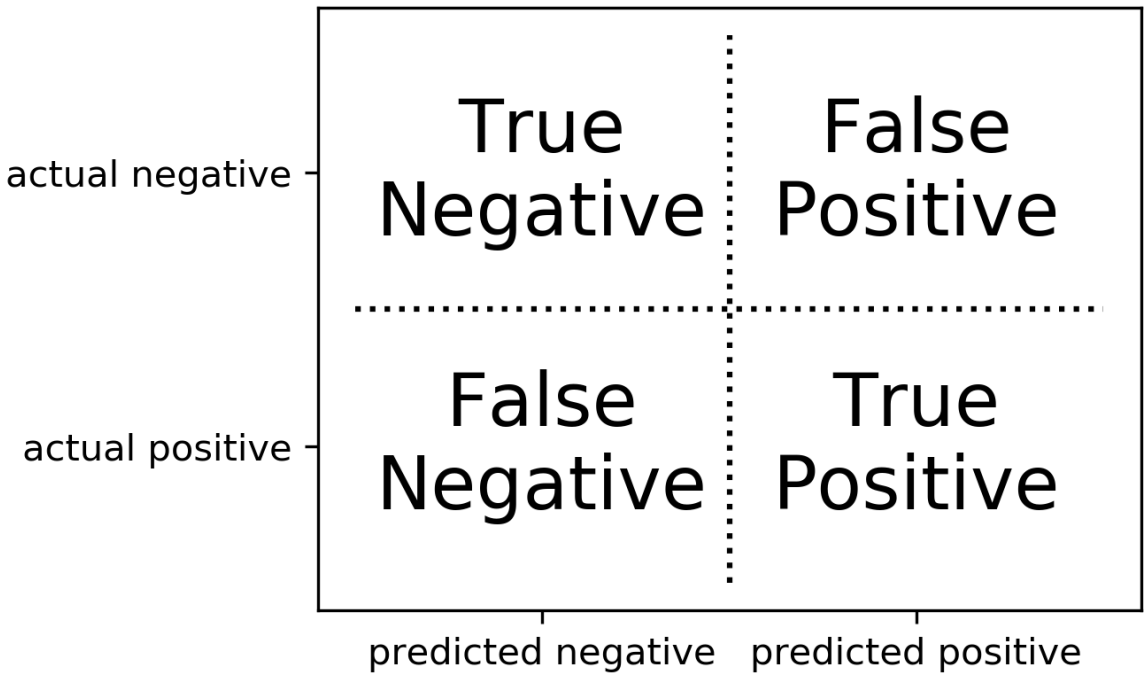
Di tutto

		True condition				
	Total population	Condition positive	Condition negative	Prevalence = $\frac{\Sigma \text{ Condition positive}}{\Sigma \text{ Total population}}$	Accuracy (ACC) = $\frac{\Sigma \text{ True positive} + \Sigma \text{ True negative}}{\Sigma \text{ Total population}}$	
Predicted condition	Predicted condition positive	**True positive**, Power	**False positive**, Type I error	Positive predictive value (PPV), Precision = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Predicted condition positive}}$	False discovery rate (FDR) = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Predicted condition positive}}$	
	Predicted condition negative	**False negative**, Type II error	**True negative**	False omission rate (FOR) = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Predicted condition negative}}$	Negative predictive value (NPV) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Predicted condition negative}}$	
		True positive rate (TPR), Recall, Sensitivity, probability of detection = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Condition positive}}$	False positive rate (FPR), Fall-out, probability of false alarm = $\frac{\Sigma \text{ False positive}}{\Sigma \text{ Condition negative}}$	Positive likelihood ratio (LR+) = $\frac{TPR}{FPR}$	Diagnostic odds ratio (DOR) = $\frac{LR+}{LR-}$	F_1 score = $\frac{2}{\frac{1}{\text{Recall}} + \frac{1}{\text{Precision}}}$
		False negative rate (FNR), Miss rate = $\frac{\Sigma \text{ False negative}}{\Sigma \text{ Condition positive}}$	True negative rate (TNR), Specificity (SPC) = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Condition negative}}$	Negative likelihood ratio (LR−) = $\frac{FNR}{TNR}$		

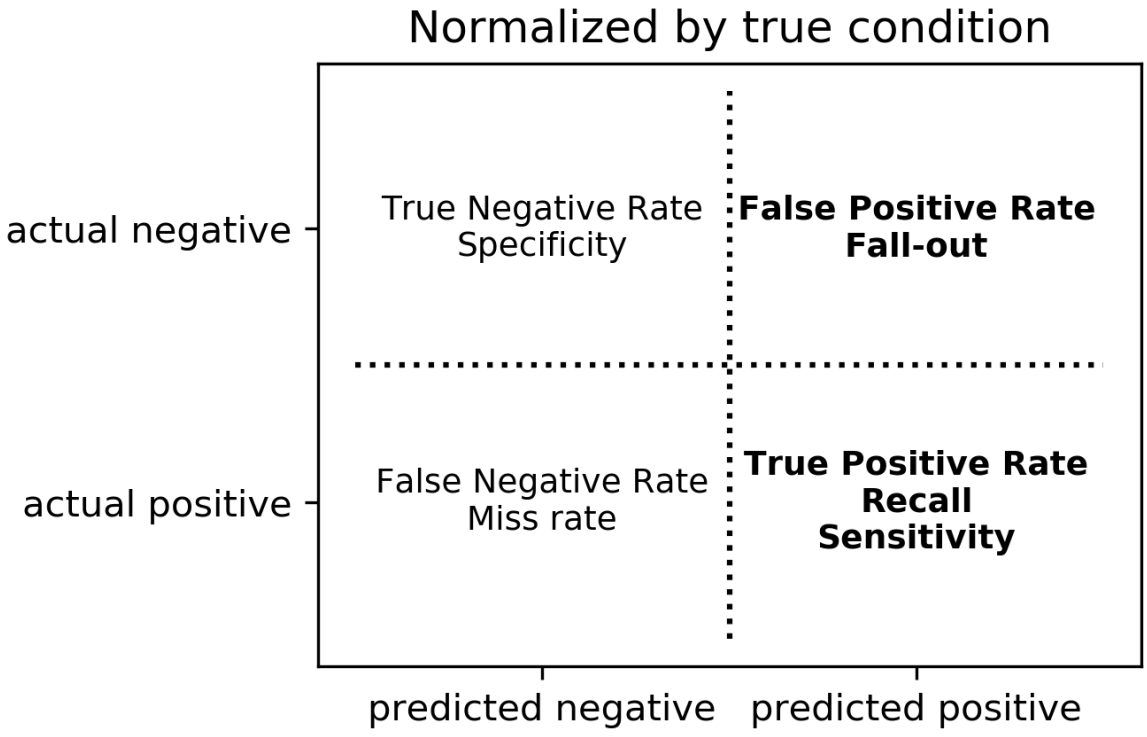
https://en.wikipedia.org/wiki/Precision_and_recall

Normalizzando la confusion matrix

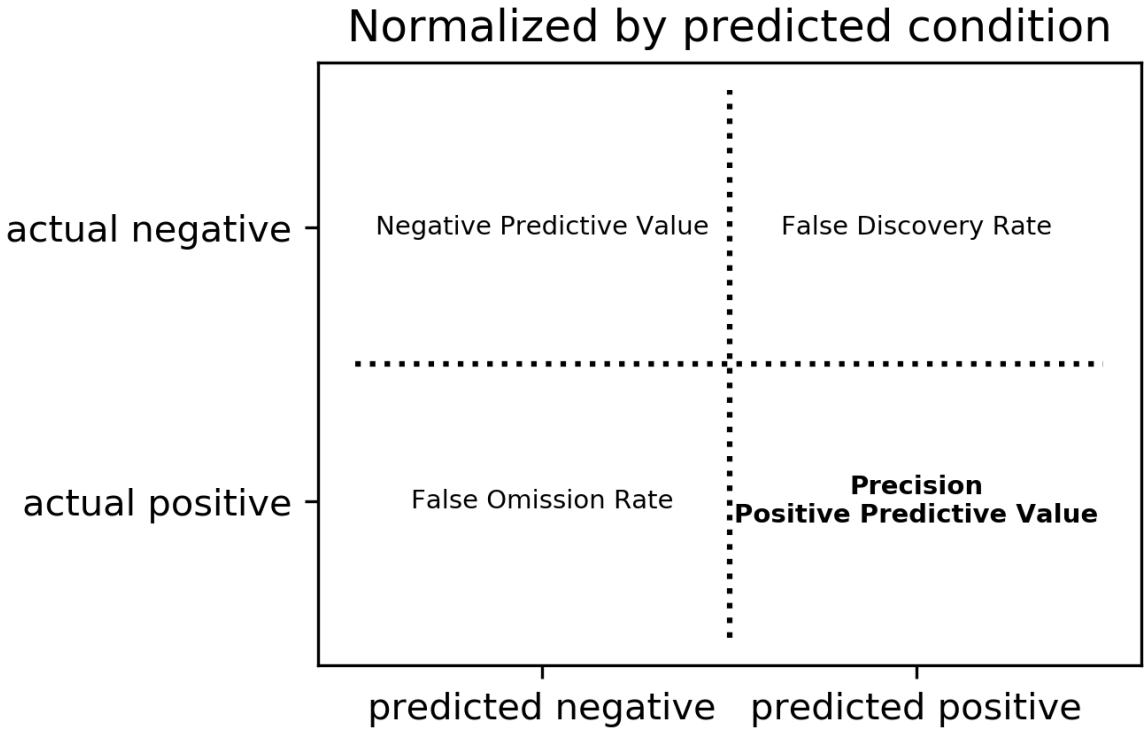
```
confusion_matrix(y_true, y_pred)
```



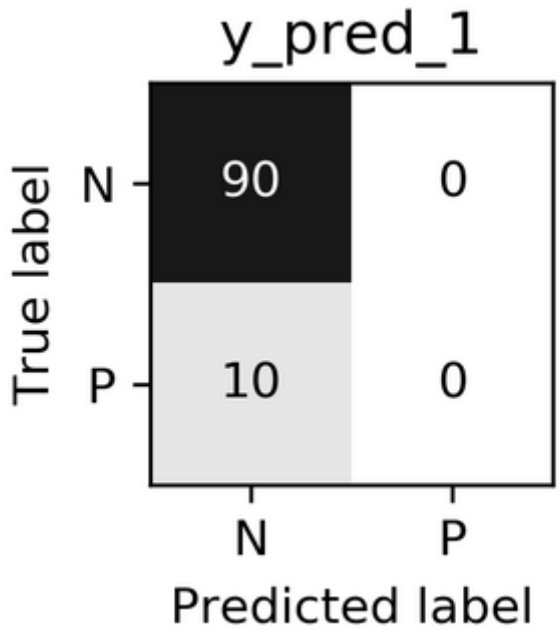

```
confusion_matrix(y_true, y_pred, normalize='true')
```



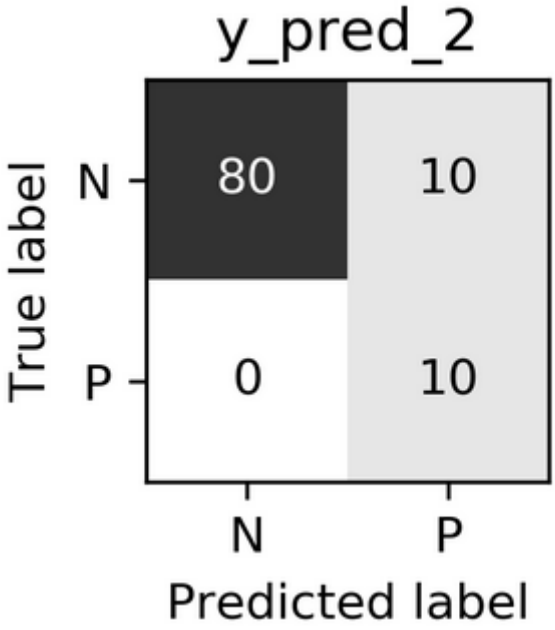

```
confusion_matrix(y_true, y_pred, normalize='pred')
```



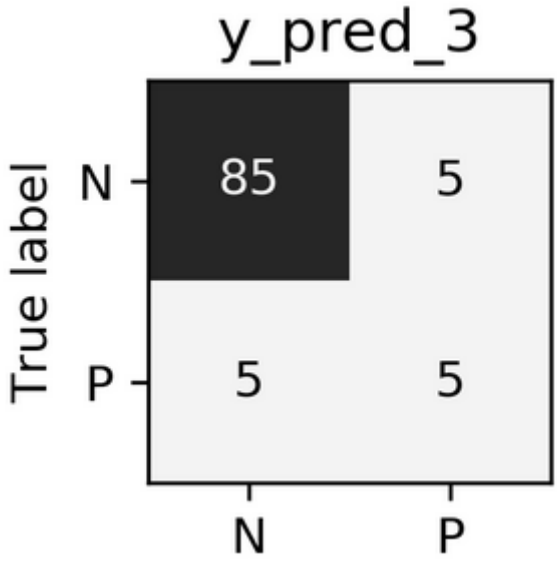

```
from sklearn.metrics import classification_report
classification_report(y_true, y_pred)
```




```
              precision    recall  f1-score   support

           0       0.90      1.00      0.95        90
           1       0.00      0.00      0.00        10

    accuracy                           0.90       100
   macro avg       0.45      0.50      0.47       100
weighted avg       0.81      0.90      0.85       100
```



	precision	recall	f1-score	support
0	1.00	0.89	0.94	90
1	0.50	1.00	0.67	10
accuracy			0.90	100
macro avg	0.75	0.94	0.80	100
weighted avg	0.95	0.90	0.91	100



	precision	recall	f1-score	support
0	0.94	0.94	0.94	90
1	0.50	0.50	0.50	10
accuracy			0.90	100
macro avg	0.72	0.72	0.72	100
weighted avg	0.90	0.90	0.90	100

Strategie di media

$$\text{macro } \frac{1}{|L|} \sum_{l \in L} R(y_l, \hat{y}_l)$$

$$\text{weighted } \frac{1}{n} \sum_{l \in L} n_l R(y_l, \hat{y}_l)$$

```
print("Weighted average: ", recall_score(y_test, y_pred_1, average="weighted"))
print("Macro average: ", recall_score(y_test, y_pred_1, average="macro"))
```

```
Weighted average: 0.90
Macro average: 0.50
```

Balanced Accuracy

$$\text{balanced\_accuracy} = \frac{1}{2}\left(\frac{TP}{TP + FN} + \frac{TN}{TN + FP}\right)$$

```
balanced_accuracy_score(y_true, y_pred)
```

Il valore migliore è 1 il peggiore è 0 quando `adjusted=False`.

- Sempre 0.5 per le predizioni fortuite
- E' uguale all'accuracy per dataset bilanciati

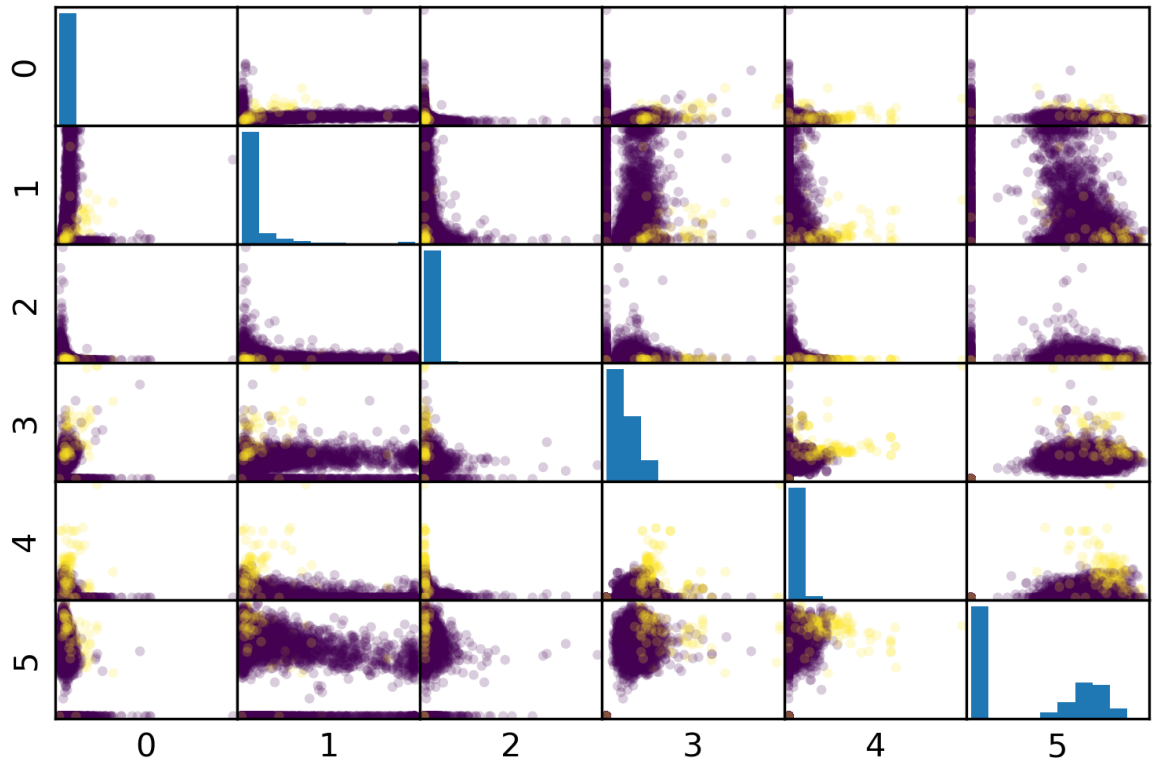
Dati Mammografia

```
from sklearn.datasets import fetch_openml
# mammography https://www.openml.org/d/310
data = fetch_openml('mammography', as_frame=True)
X, y = data.data, data.target
X.shape
```

(11183, 6)

```
y.value_counts()
```

```
-1    10923
1       260
```




```
# make y boolean
# this allows sklearn to determine the positive class more easily
X_train, X_test, y_train, y_test = train_test_split(X, y == '1', random_state=0)
```

```
svc = make_pipeline(StandardScaler(),
                    SVC(C=100, gamma=0.1))
svc.fit(X_train, y_train)
svc.score(X_test, y_test)
```

```
0.986
```

```
rf = RandomForestClassifier()

rf.fit(X_train, y_train)
rf.score(X_test, y_test)
```

```
0.989
```

```
SVC           precision   recall  f1-score  support

       False       0.99     1.00      0.99     2732
        True       0.81     0.53      0.64       64

    accuracy                           0.99     2796
   macro avg       0.90     0.76      0.82     2796
weighted avg       0.98     0.99      0.99     2796
```

```
RF            precision   recall  f1-score  support

       False       0.99     1.00      0.99     2732
        True       0.90     0.56      0.69       64

    accuracy                           0.99     2796
   macro avg       0.94     0.78      0.84     2796
weighted avg       0.99     0.99      0.99     2796
```

Approccio ad obiettivo

- Cosa voglio a cosa sono interessato?
- Posso aggiungere dei costi alla confusion matrix?
- Quali garanzie vogliamo dare?

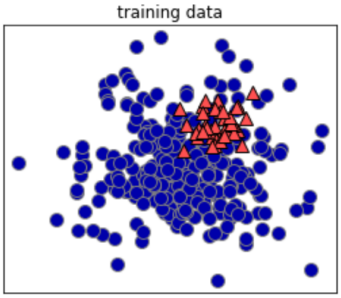
Cambio i Threshold

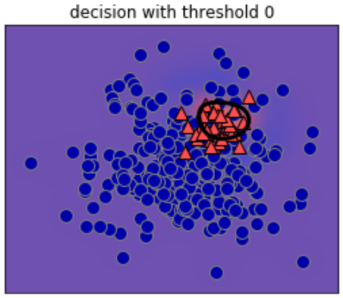
```
y_pred = rf.predict(X_test)
print(classification_report(y_test, y_pred))
```

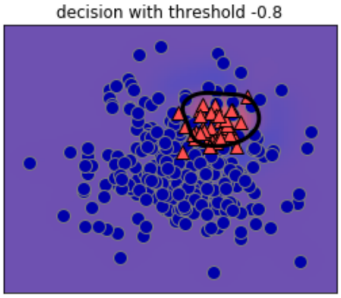
```
              precision    recall  f1-score   support

       False       0.99      1.00      0.99      2732
        True       0.90      0.56      0.69        64

    accuracy                           0.99      2796
   macro avg       0.94      0.78      0.84      2796
weighted avg       0.99      0.99      0.99      2796
```

```
y_pred = rf.predict_proba(X_test)[:, 1] > .30


print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

       False       0.99      0.99      0.99      2732
        True       0.71      0.64      0.67        64

    accuracy                           0.99      2796
   macro avg       0.85      0.82      0.83      2796
weighted avg       0.99      0.99      0.99      2796
```

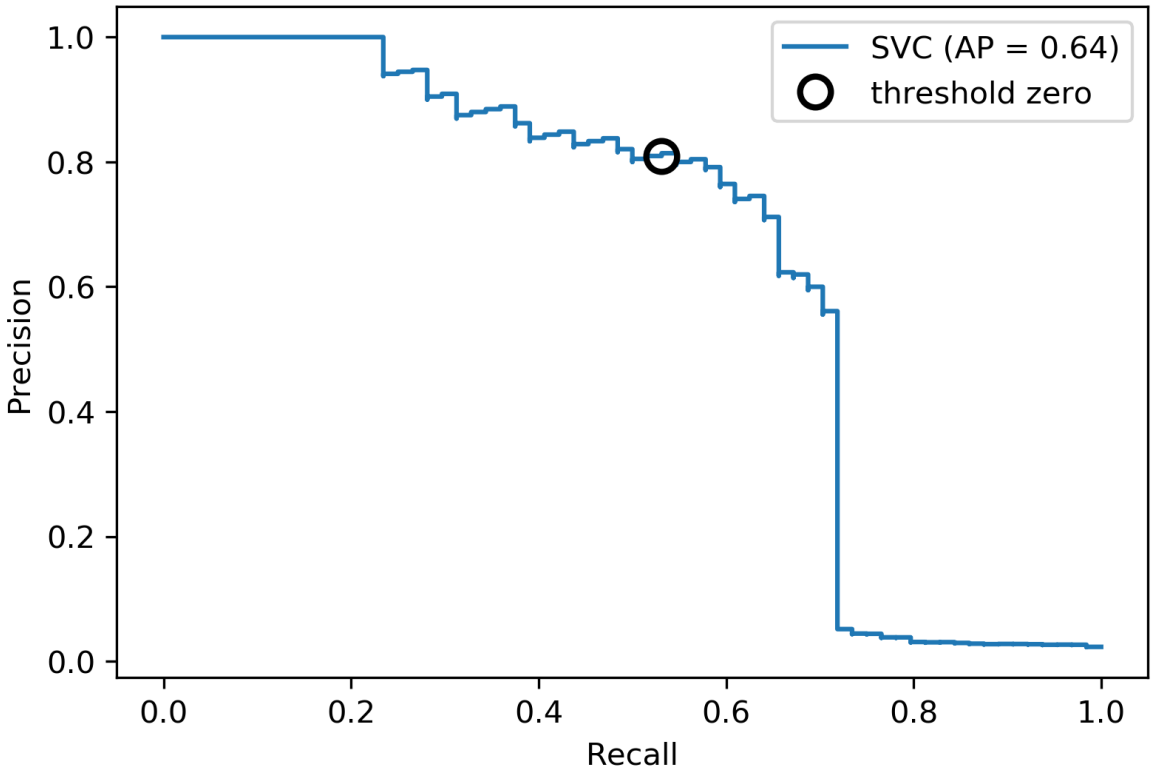






Precision-Recall curve

```
svc = make_pipeline(StandardScaler(), SVC(C=100, gamma=0.1))
svc.fit(X_train, y_train)
plot_precision_recall_curve(svc, X_test, y_test, name='SVC')
```



Scikit-learn plotting API

```
prc = plot_precision_recall_curve(svc, X_test, y_test, name='SVC')
# plot pops up here
prc
```

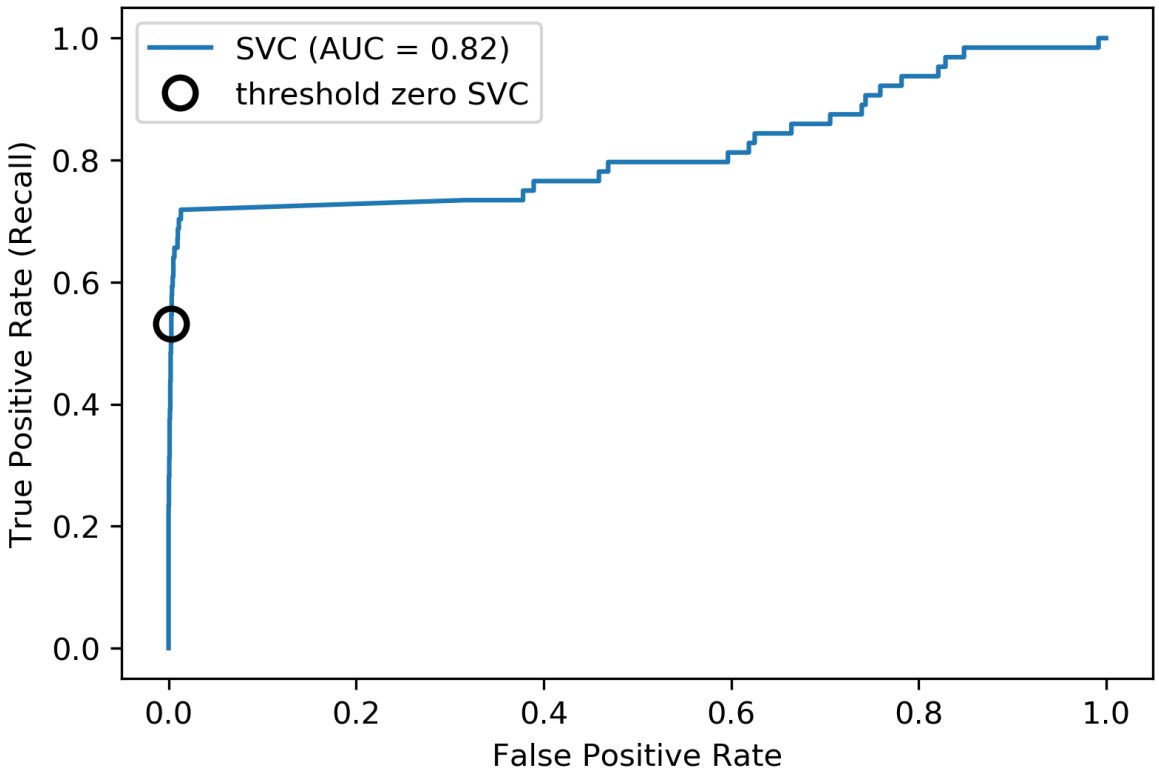
```
vars(pr_svc)
```

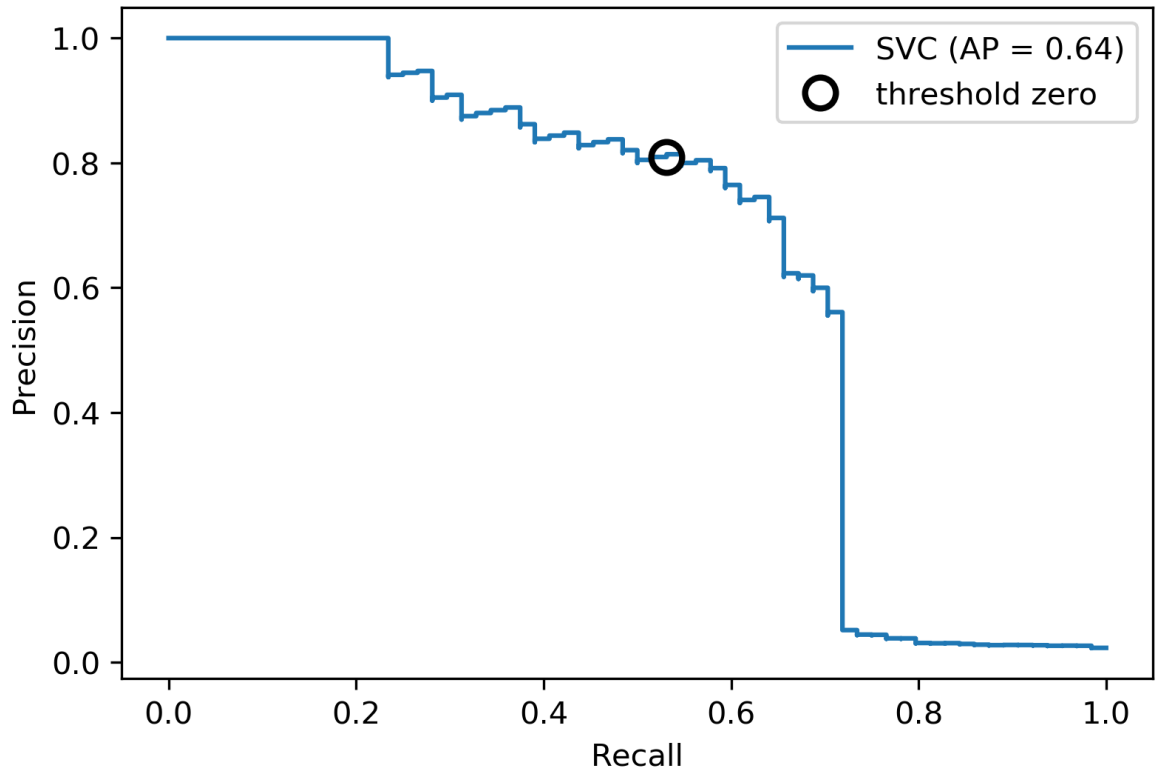
```
{'precision': array([0.023, 0.023, 0.023, ..., 1.   , 1.   , 1.   ]),
 'recall': array([1.   , 0.984, 0.984, ..., 0.031, 0.016, 0.   ]),
 'average_precision': 0.6743452407177641,
 'estimator_name': 'Pipeline',
 'line_': <matplotlib.lines.Line2D at ... >,
 'ax_': <matplotlib.axes._subplots.AxesSubplot at ...>,
 'figure_': <Figure size 432x288 with 1 Axes>}
```

ROC Curve

(Receiver Operating Characteristic)

```
plot_roc_curve(svc, X_test, y_test, name='SVC')
```



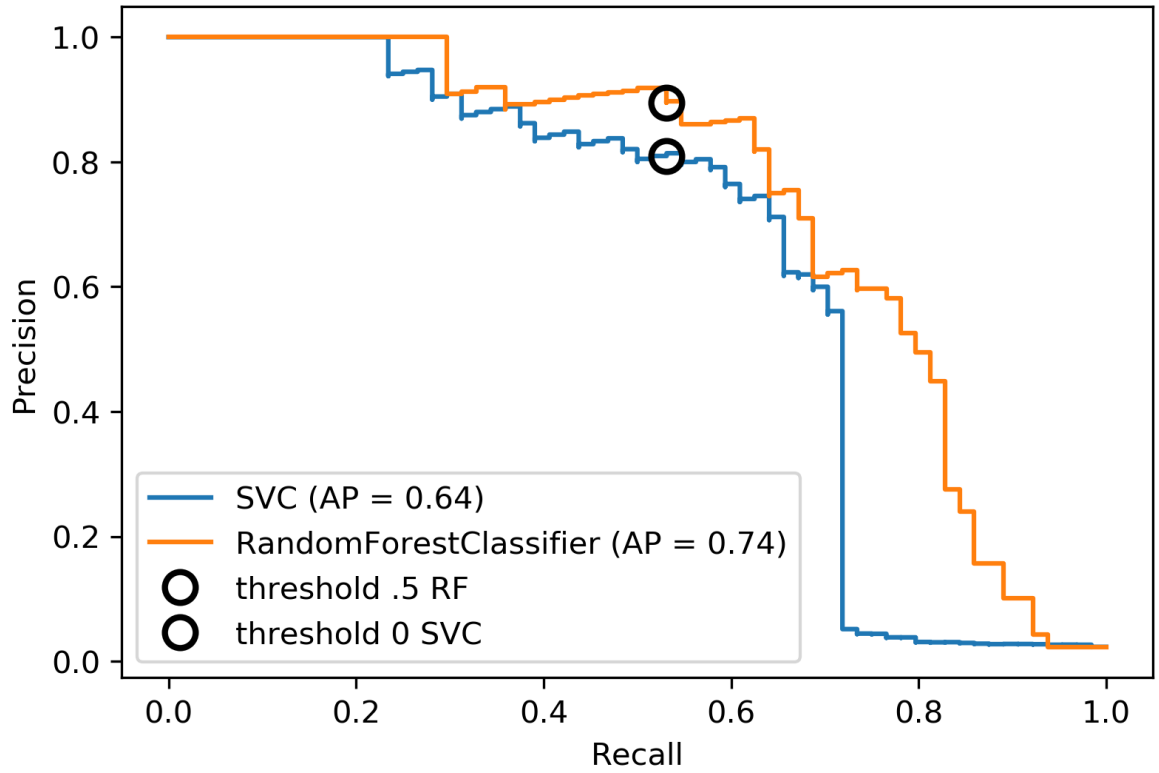


- hanno un asse in comune (trasposto)

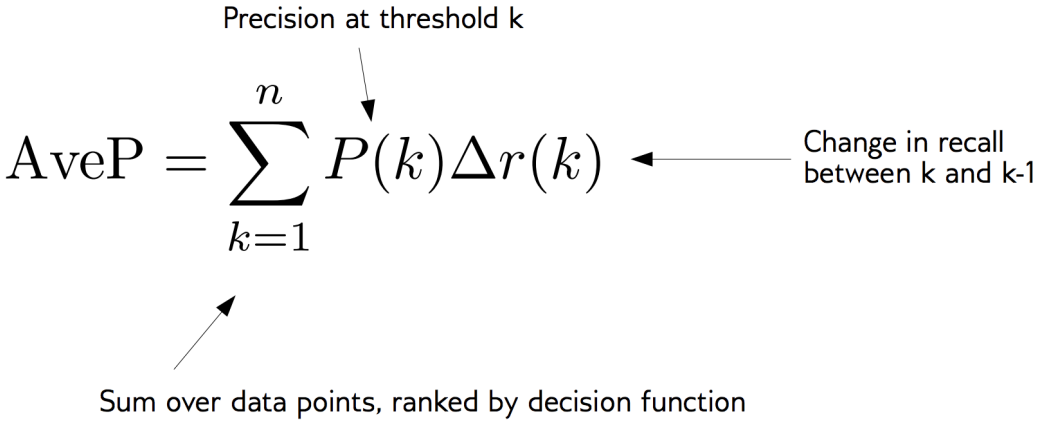
- posso fare interpolatione sulla ROC non sulla PR

Paragonizamo RF e SVC

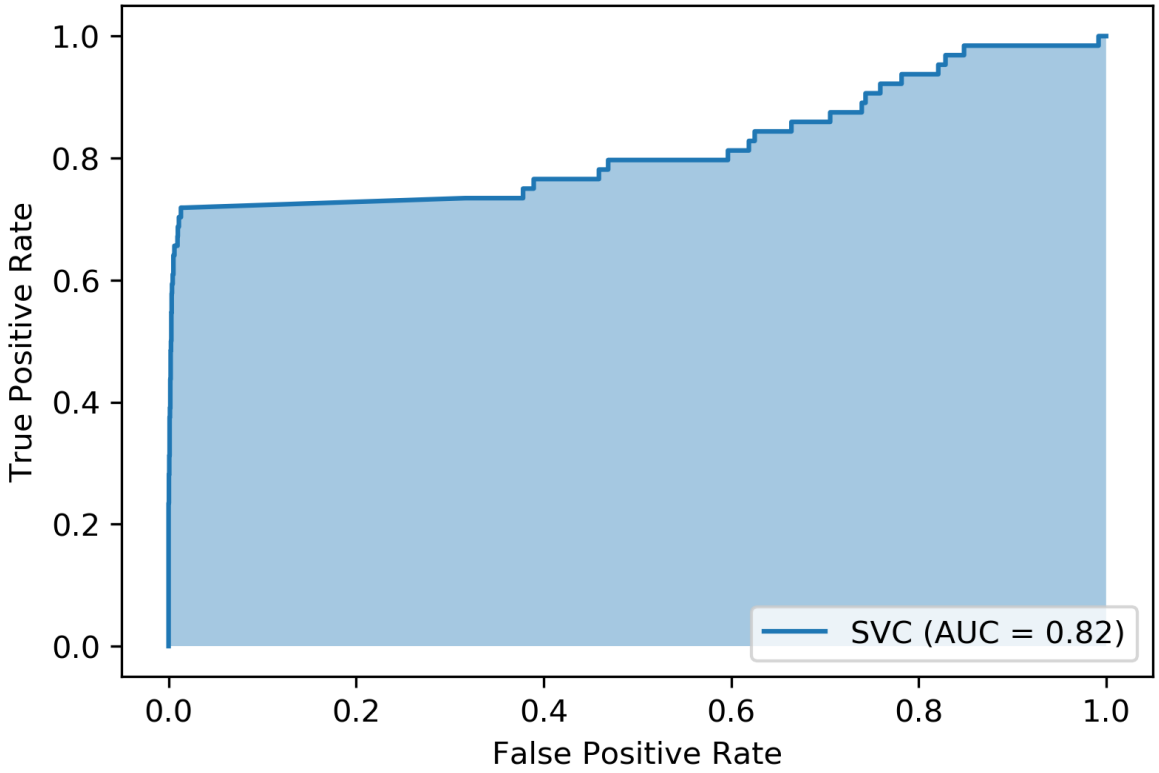
```
pr_svc = plot_precision_recall_curve(svc, X_test, y_test, name='SVC')
# if we used computed before, we could just call pr_svc.plot()
# using ax=plt.gca() will plot into the existing axes instead of creating  new
ones
pr_rf = plot_precision_recall_curve(rf, X_test, y_test, ax=plt.gca())
```



Average Precision



Area Under ROC Curve (AUC)



sempre .5 per predizioni random costanti

Threshold and ranking metrics

```
from sklearn.metrics import f1_score
f1_rf = f1_score(y_test, rf.predict(X_test))
print(f"f1_score of random forest: {f1_rf:.3f}")
f1_svc = f1_score(y_test, svc.predict(X_test))
print(f"f1_score of svc: {f1_svc:.3f}")
```

```
f1_score of random forest: 0.709
f1_score of svc: 0.715
```

```
from sklearn.metrics import balanced_accuracy_score
ba_rf = balanced_accuracy_score(y_test, rf.predict(X_test))
print(f"Balanced accuracy of random forest: {ba_rf:.3f}")
ba_svc = balanced_accuracy_score(y_test, svc.predict(X_test))
print(f"Balanced accuracy of svc: {ba_svc:.3f}")
```

```
Balanced accuracy of random forest: 0.765
Balanced accuracy of svc: 0.764
```

```
from sklearn.metrics import average_precision_score
ap_rf = average_precision_score(
y_test, rf.predict_proba(X_test)[:, 1])
    print(f"Average precision of random forest: {ap_rf:.3f}")
ap_svc = average_precision_score(
    y_test, svc.decision_function(X_test))
print(f"Average precision of svc: {ap_svc:.3f}")
```

```
Average precision of random forest: 0.682
Average precision of svc: 0.693
```

```
from sklearn.metrics import roc_auc_score
auc_rf = roc_auc_score(y_test, rf.predict_proba(X_test)[:, 1])
print(f"Area under ROC curve of random forest: {auc_rf:.3f}")
auc_svc = roc_auc_score(y_test, svc.decision_function(X_test))
print(f"Area under ROC curve of svc: {auc_svc:.3f}")
```

```
Area under ROC curve of random forest: 0.936
Area under ROC curve of svc: 0.817
```

https://www.biostat.wisc.edu/~page/rocpr.pdf

https://papers.nips.cc/paper/5867-precision-recall-gain-curves-pr-analysis-done-right.pdf

Riassunto delle metriche di classificatione binaria

Threshold-based:

- (balanced) accuracy
- precision, recall, f1

Ranking:

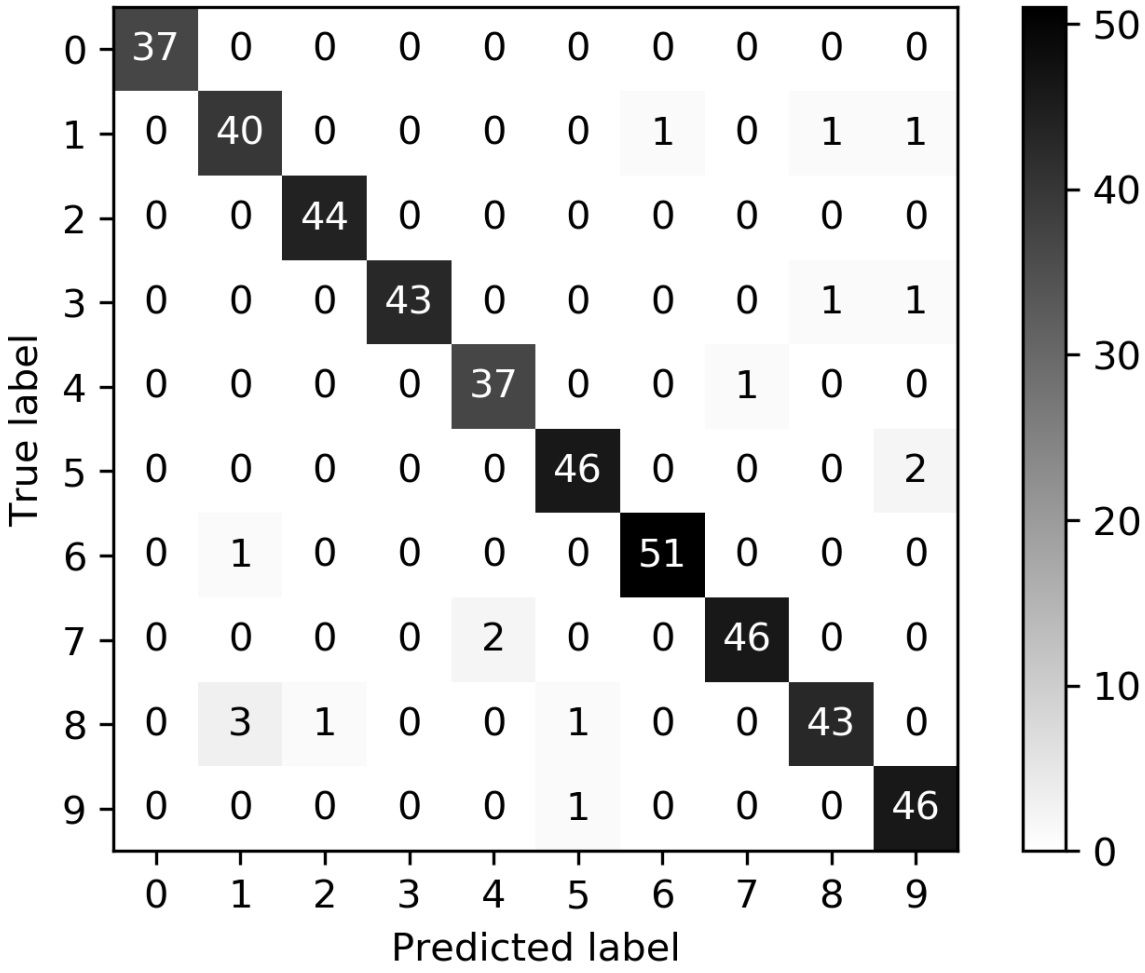
- average precision
- ROC AUC

Multi-class classification

Confusion Matrix

```
from sklearn.datasets import load_digits
from sklearn.metrics import accuracy_score

digits = load_digits()
 # data is between 0 and 16
X_train, X_test, y_train, y_test = train_test_split(
    digits.data / 16., digits.target, random_state=0)
lr = LogisticRegression().fit(X_train, y_train)
pred = lr.predict(X_test)
print("Accuracy: {:.3f}".format(accuracy_score(y_test, pred)))
plot_confusion_matrix(lr, X_test, y_test, cmap='gray_r')
```

```
Accuracy: 0.964
```




```
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        37
           1       0.91      0.93      0.92        43
           2       0.98      1.00      0.99        44
           3       1.00      0.96      0.98        45
```

```
           4       0.95      0.97      0.96        38
           5       0.96      0.96      0.96        48
           6       0.98      0.98      0.98        52
           7       0.98      0.96      0.97        48
           8       0.96      0.90      0.92        48
           9       0.92      0.98      0.95        47

    accuracy                           0.96       450
   macro avg       0.96      0.96      0.96       450
weighted avg       0.96      0.96      0.96       450
```

Multi-class ROC AUC

- Hand & Till, 2001, one vs one

$$\frac{1}{c(c-1)} \sum_{j=1}^{c} \sum_{k \neq j}^{c} AUC(j,k)$$

- Provost & Domingo, 2000, one vs rest

$$\frac{1}{c} \sum_{j=1}^{c} p(j) AUC(j, \text{rest}_j)$$

Riassunto metriche multiclass classification

Threshold-based:

- accuracy
- precision, recall, f1 (macro average, weighted)

Ranking:

- OVR ROC AUC
- OVO ROC AUC

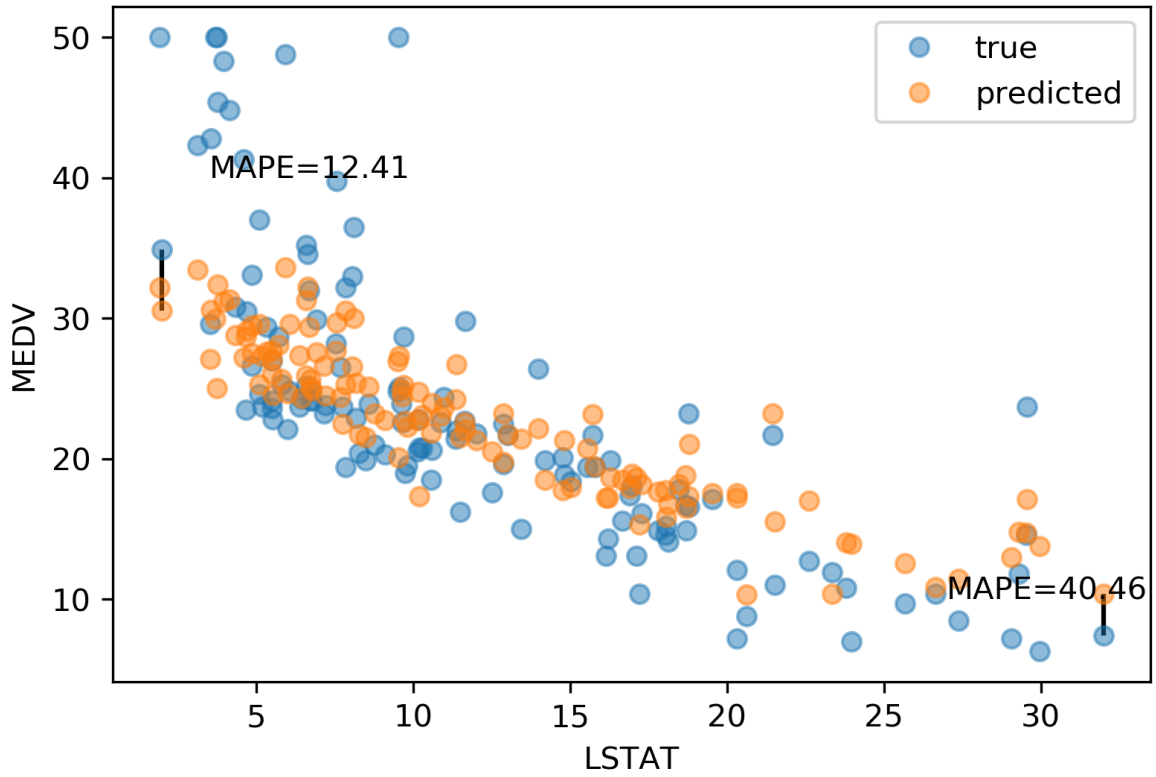
Metrics per i modelli di regressione

Build-in standard metrics

- R2: easy to understand scale
- MSE: easy to relate to input
- Mean absolute error, median absolute error: more robust

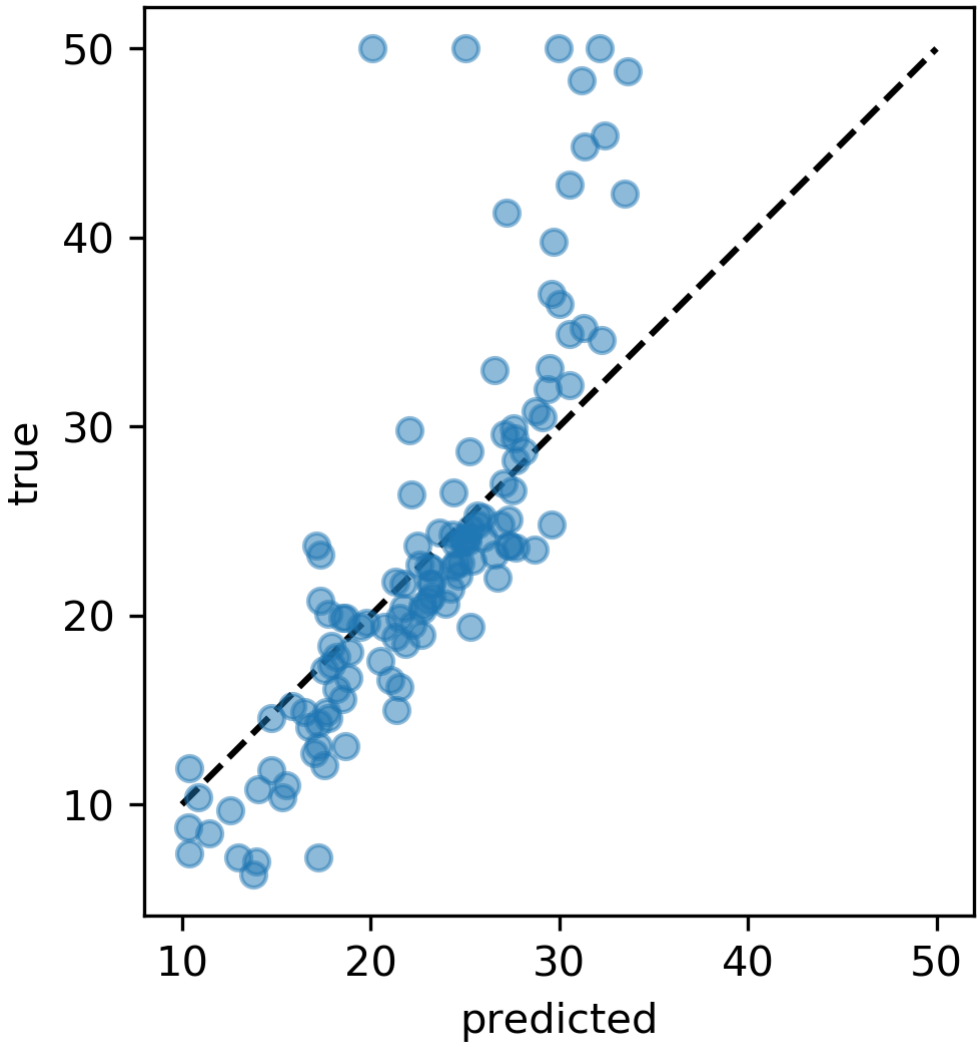
Absolute vs Relative: MAPE

$$\text{MAPE} = \frac{100}{n} \sum_{i=1}^{n} \left| \frac{y - \hat{y}}{y} \right|$$



Prediction plots

```
from sklearn.linear_model import Ridge
from sklearn.datasets import load_boston
boston = load_boston()
X_train, X_test, y_train, y_test = train_test_split(
    boston.data, boston.target)
ridge = Ridge(normalize=True)
ridge.fit(X_train, y_train)
pred = ridge.predict(X_test)
plt.plot(pred, y_test, 'o')
```

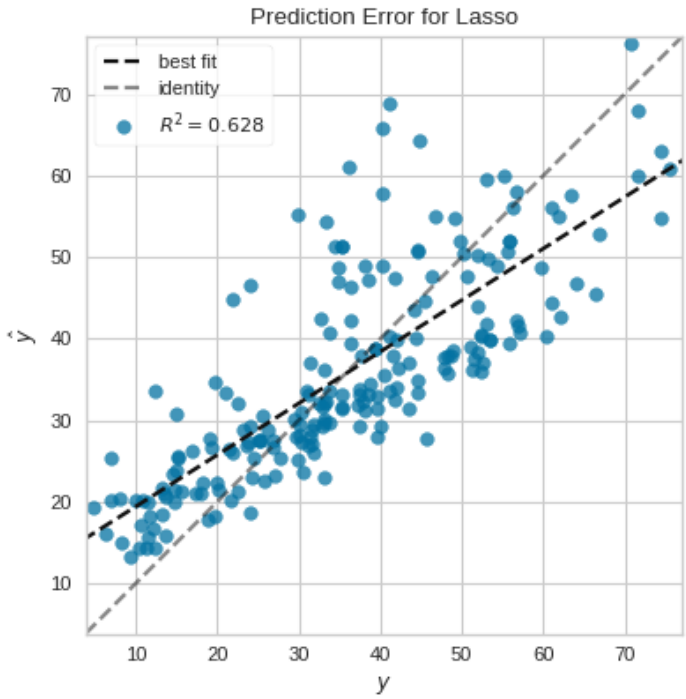


case di Boston

dovrebbe essere sulla diagonale

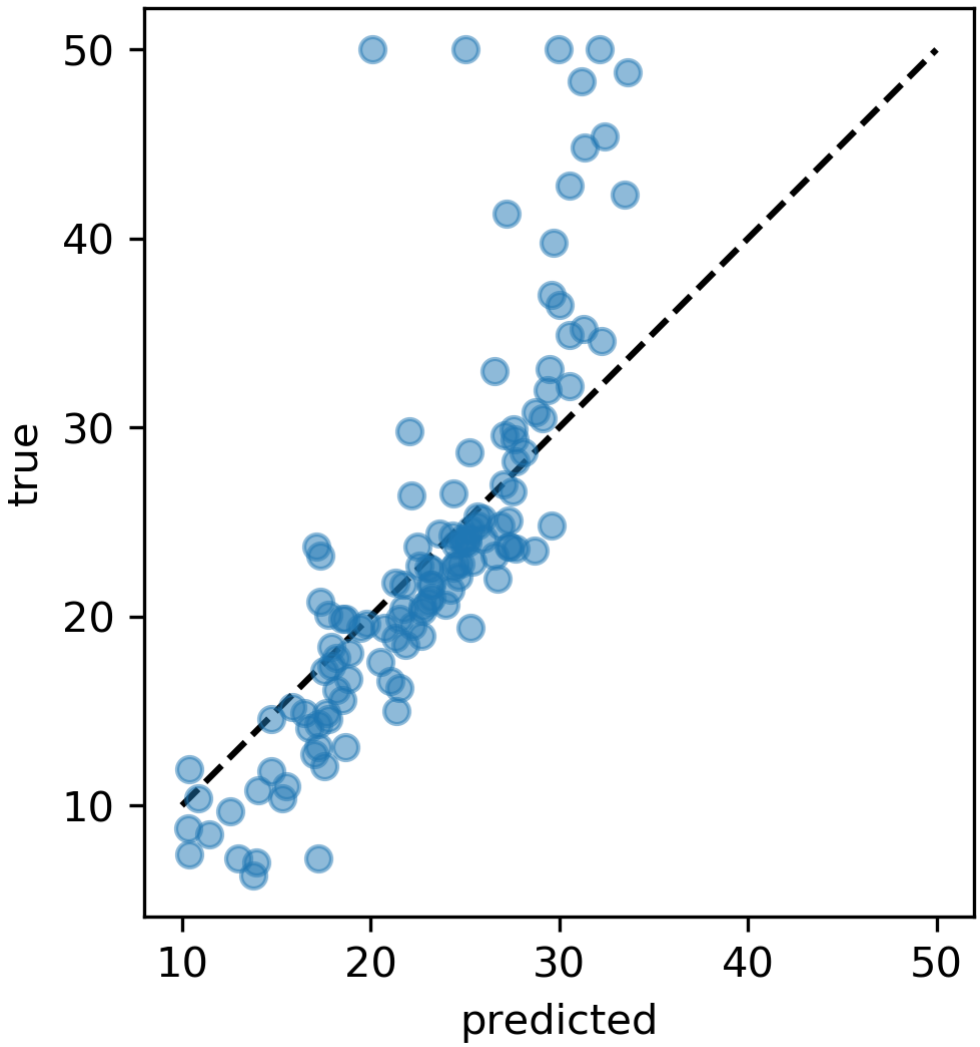
- predico troppo alto per case che costano poco
- predico troppo basso per case che costano molto

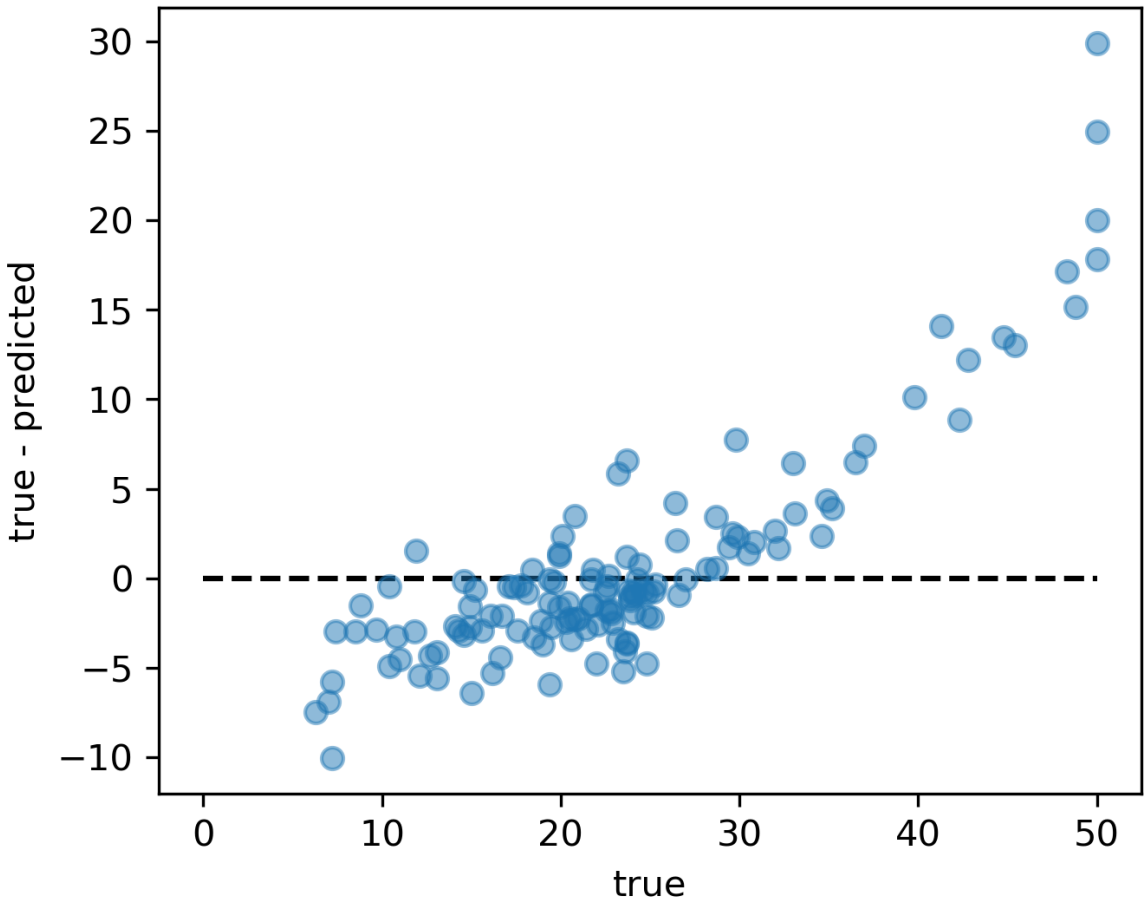
- Posso vedere se ci sono particolari trend



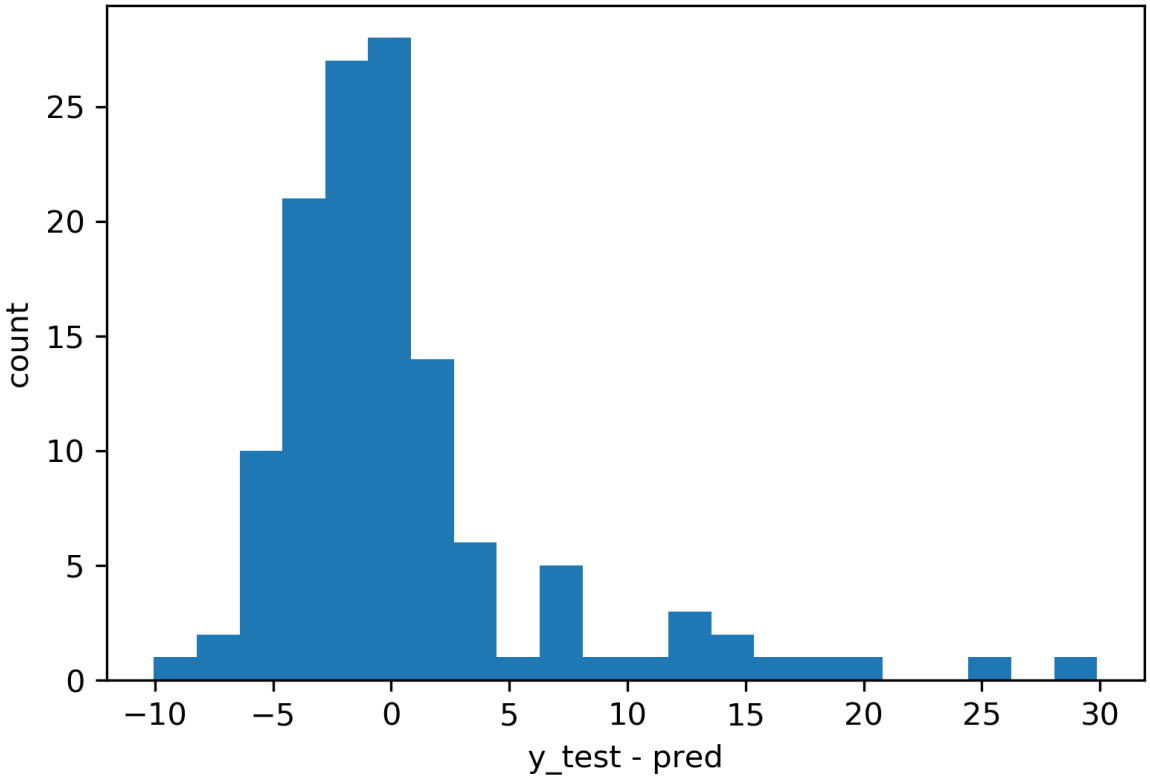
da yellowbrick https://www.scikit-yb.org/en/latest/api/regressor/peplot.html#residuals-plot

Residual Plot

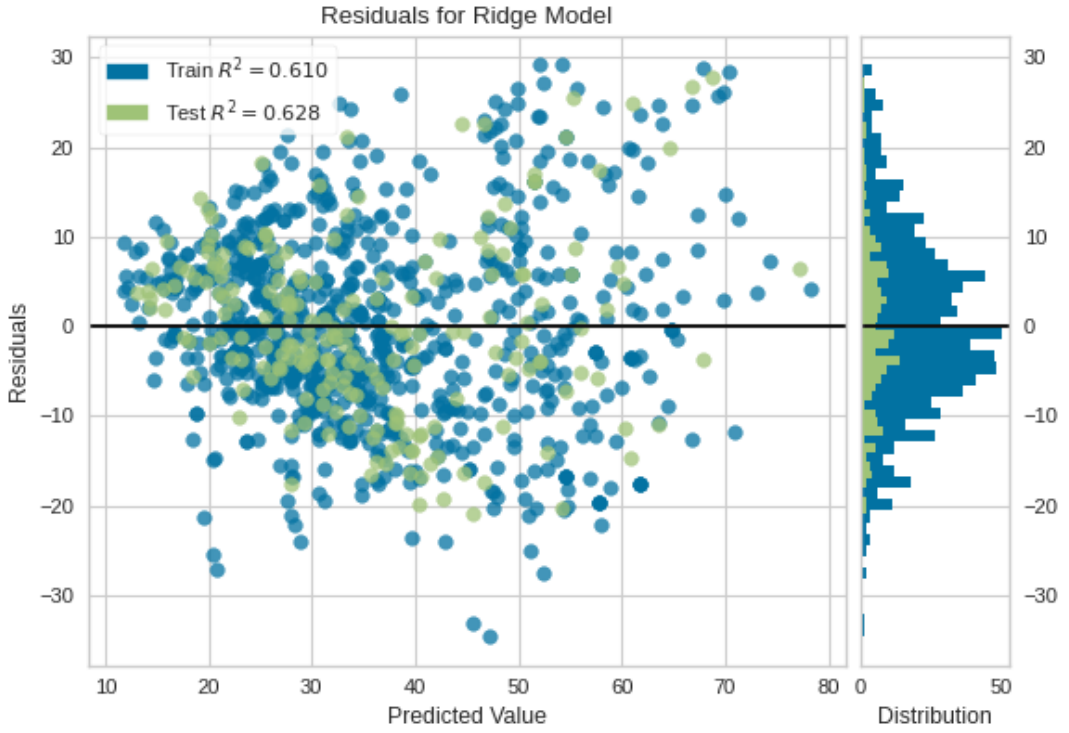




- Il grafico dei residui assomiglia al precedente ruotato di 90°
- dovrebbe essere una retta orizzontale con spread allargamento gaussiano
- mi mostra delle dipendenze residue non spiegate

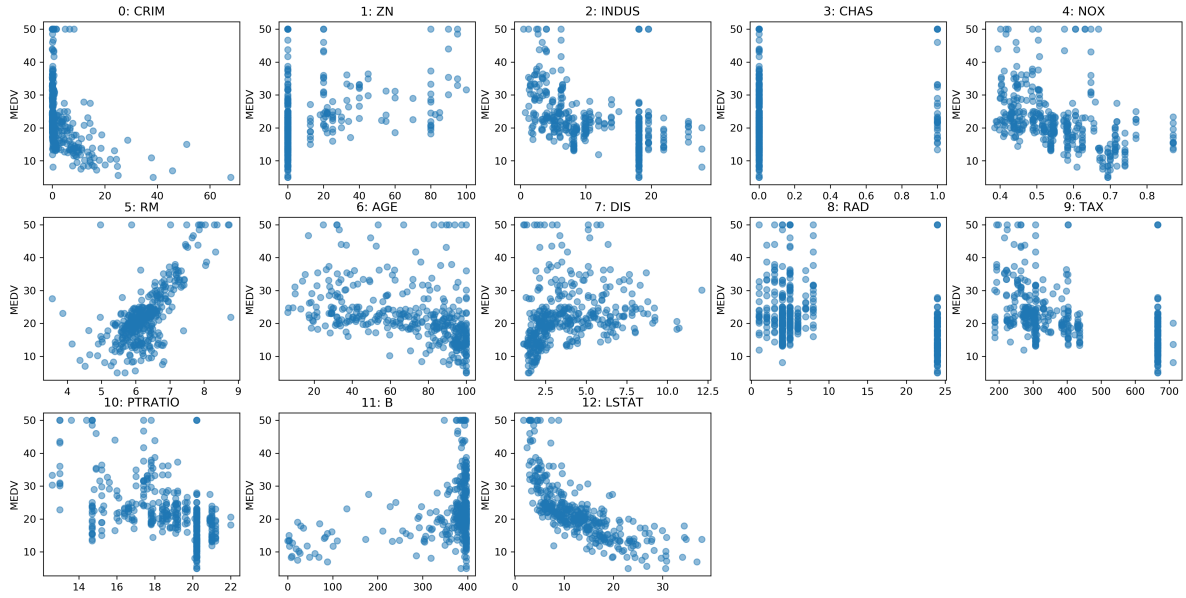


- dovrebbe essere una campana
- una cosa buona di questi grafici è che sono indipendenti dalla dimensionalità del dataset

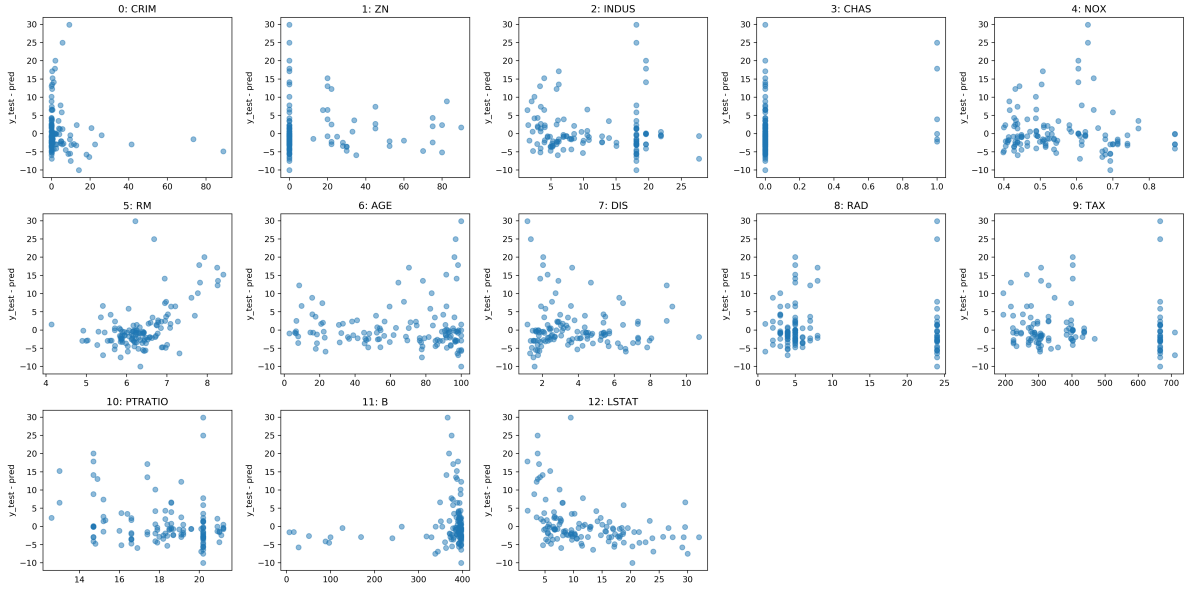


(sempre yellowbrick)

Target vs Feature



Residual vs Feature



Come scegliere le metriche

- Accuracy rarely what you want
- Problems are rarely balanced
- Find the right criterion for the task
- OR pick a substitude, but at least think about it
- Emphasis on recall or precision?
- Which classes are the important ones?

Using metrics in cross-validation

```
cancer = load_breast_cancer()
X, y = cancer.data, cancer.target

# default scoring for classification is accuracy
rf = RandomForestClassifier(random_state=0)
print("default scoring ", cross_val_score(rf, X, y))

# providing scoring="accuracy" doesn't change the results
explicit_accuracy =  cross_val_score(rf, X, y,  scoring="accuracy")
print("explicit accuracy scoring ", explicit_accuracy)

ap =  cross_val_score(rf, X, y, scoring="average_precision")
print("average precision", ap)
```

```
default scoring  [0.93  0.947 0.991 0.974 0.973]
explicit accuracy scoring  [0.93  0.947 0.991 0.974 0.973]
average precision [0.992 0.973 0.999 0.995 0.999]
```

Multiple Metrics

```
from sklearn.model_selection import cross_validate
res = cross_validate(RandomForestClassifier(), X, y,
                     scoring=["accuracy", "average_precision",
                     "recall_macro"],
                     return_train_score=True, cv=5)
pd.DataFrame(res)
```

	fit_time	score_time	test_accuracy	train_accuracy	test_average_precision	train_average_precision	test_recall_macro	train_recall_mac
0	0.171312	0.016973	0.921053	1.0	0.992115	1.0	0.918277	1
1	0.133090	0.017563	0.938596	1.0	0.972293	1.0	0.923190	1
2	0.110232	0.015200	0.982456	1.0	0.999098	1.0	0.981151	1
3	0.110211	0.015076	0.956140	1.0	0.992924	1.0	0.950397	1
4	0.124239	0.019509	0.973451	1.0	0.998858	1.0	0.974011	1

Built-in scoring

```
from sklearn.metrics import SCORERS
print("\n".join(sorted(SCORERS.keys())))
```

```
accuracy                        adjusted_mutual_info_score
adjusted_rand_score
average_precision               balanced_accuracy               completeness_score
explained_variance              f1                              f1_macro
f1_micro                        f1_samples                      f1_weighted
fowlkes_mallows_score           homogeneity_score               jaccard
jaccard_macro                   jaccard_micro                   jaccard_samples
jaccard_weighted                max_error                       mutual_info_score
neg_brier_score                 neg_log_loss
neg_mean_absolute_error
neg_mean_gamma_deviance         neg_mean_poisson_deviance
neg_mean_squared_error
neg_mean_squared_log_error      neg_median_absolute_error
neg_root_mean_squared_error
normalized_mutual_info_score    precision                       precision_macro
precision_micro                 precision_samples               precision_weighted
r2                              recall                          recall_macro
recall_micro                    recall_samples                  recall_weighted
roc_auc                         roc_auc_ovo
roc_auc_ovo_weighted
roc_auc_ovr                     roc_auc_ovr_weighted            v_measure_score
```

The Scorer interface (vs the metrics interface)

```
# metric function interface:
y_probs = rf.predict_proba(X_test)
ap_rf = average_precision_score(y_test, y_probs[:, 1])
y_pred = rf.predict(X_test)
ba_rf = balanced_accuracy_score(y_test, y_pred)
# scorer interface:
from sklearn.metrics import get_scorer
ap_scorer = get_scorer('average_precision')
ap_rf = ap_scorer(rf, X_test, y_test)
ab_scorer = get_scorer('balanced_accuracy')
ba_rf = ab_scorer(rf, X_test, y_test)
```

Fate da voi il vostro estimatore

Takes estimator, X, y

Returns score – higher is better (always!)

```
def accuracy_scoring(est, X, y):
    return (est.predict(X) == y).mean()
```

Ho reimplementato l'accuracy

Puoi accedere al modello!

```
def nonzero(est, X, y):
    return np.sum(est.coef_ != 0)

param_grid = {'alpha': np.logspace(-5, 0, 10)}
# scoring can be string, single scorer, list or dict
grid = GridSearchCV(Lasso(), param_grid, return_train_score=True,
                    scoring={'r2': 'r2', 'num_nonzero': nonzero}, refit='r2')
grid.fit(X_train, y_train)
a = results.plot('param_alpha', 'mean_train_r2')
b = results.plot('param_alpha', 'mean_test_r2', ax=plt.gca())
ax2 = plt.gca().twinx()
results.plot('param_alpha', 'mean_train_num_nonzero', ax=ax2, c='k')
```