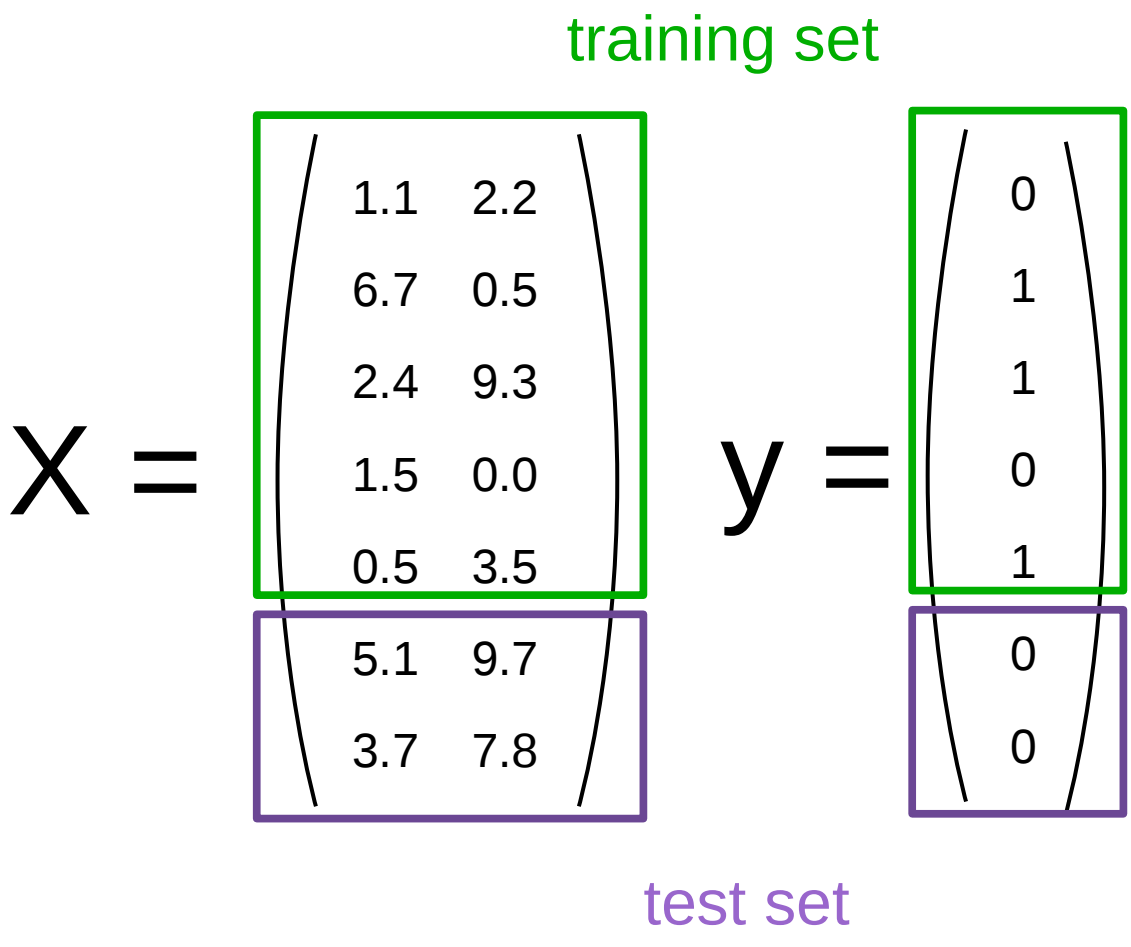
Model evaluation and metrics

Andreas Müller


training set
X =
1.1 2.2
6.7 0.5
2.4 9.3
1.5 0.0
0.5 3.5
5.1 9.7
3.7 7.8
y =
0
1
1
0
1
0
0
test set

Overfitting the test set

```
from sklearn.datasets import load_breast_cancer
from sklearn.preprocessing import scale

data = load_breast_cancer()
X, y = data.data, data.target
X = scale(X)

X_trainval, X_test, y_trainval, y_test = train_test_split(X, y)
X_train, X_val, y_train, y_val = train_test_split(X_trainval, y_trainval)
```

```
knn = KNeighborsClassifier(n_neighbors=5).fit(X_train, y_train)

print("Validation: {:.3f}".format(knn.score(X_val, y_val)))
print("Test: {:.3f}".format(knn.score(X_test, y_test)))
```

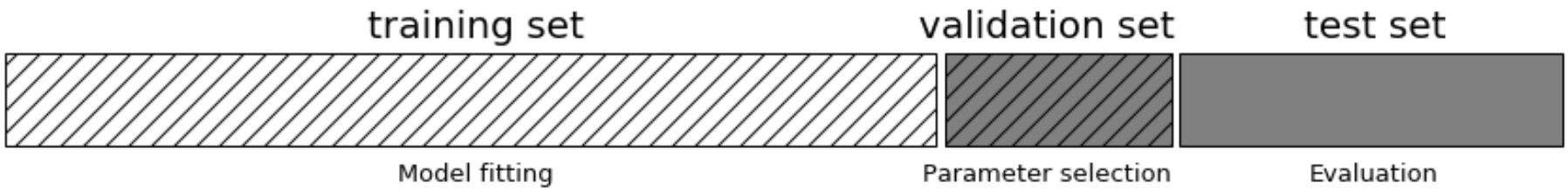
```
Validation: 0.963
Test: 0.972
```

Overfitting the test set

```
val = []
test = []

for i in range(1000):
    rng = np.random.RandomState(i)
    noise = rng.normal(scale=.1, size=X_train.shape)
    knn = KNeighborsClassifier(n_neighbors=5).fit(X_train + noise, y_train)
    val.append(knn.score(X_val, y_val))
    test.append(knn.score(X_test, y_test))

print("Validation: {:.3f}".format(np.max(val)))
print("Test: {:.3f}".format(test[np.argmax(val)]))
```

```
Validation: 0.991
Test: 0.958
```

Three-fold split

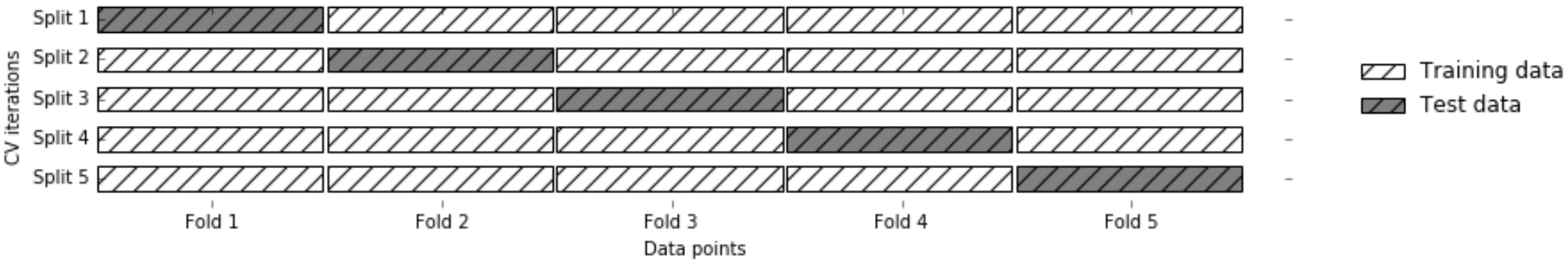


pro: fast, simple
con: high variance, bad use of data.

```
val_scores = []
neighbors = np.arange(1, 15, 2)
for i in neighbors:
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    val_scores.append(knn.score(X_val, y_val))
print("best validation score: {:.3f}".format(np.max(val_scores)))
best_n_neighbors = neighbors[np.argmax(val_scores)]
print("best n_neighbors: {}".format(best_n_neighbors))

knn = KNeighborsClassifier(n_neighbors=best_n_neighbors)
knn.fit(X_trainval, y_trainval)
print("test-set score: {:.3f}".format(knn.score(X_test, y_test)))
```

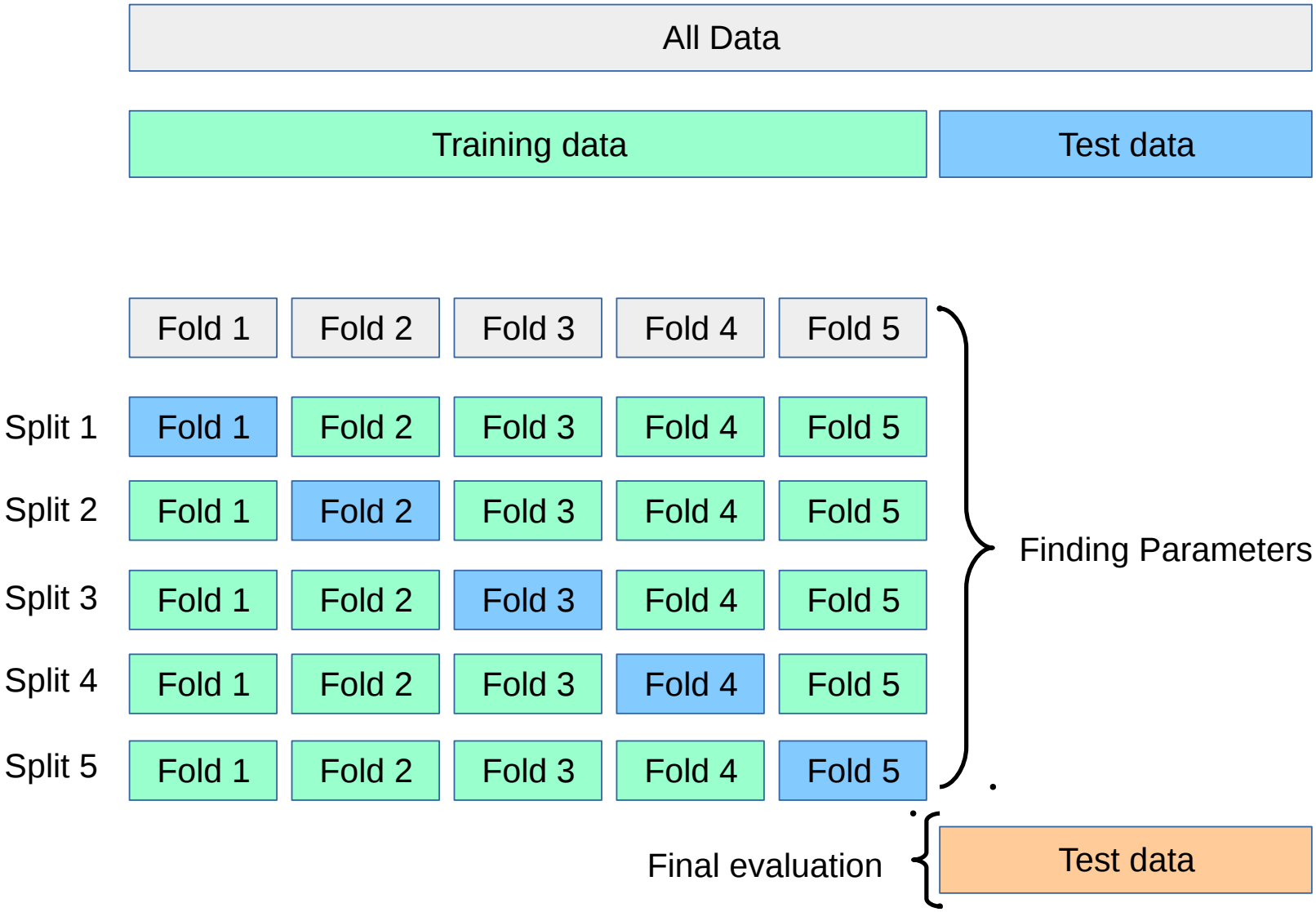
```
best validation score: 0.972
best n_neighbors: 3
test-set score: 0.965
```

Cross-validation



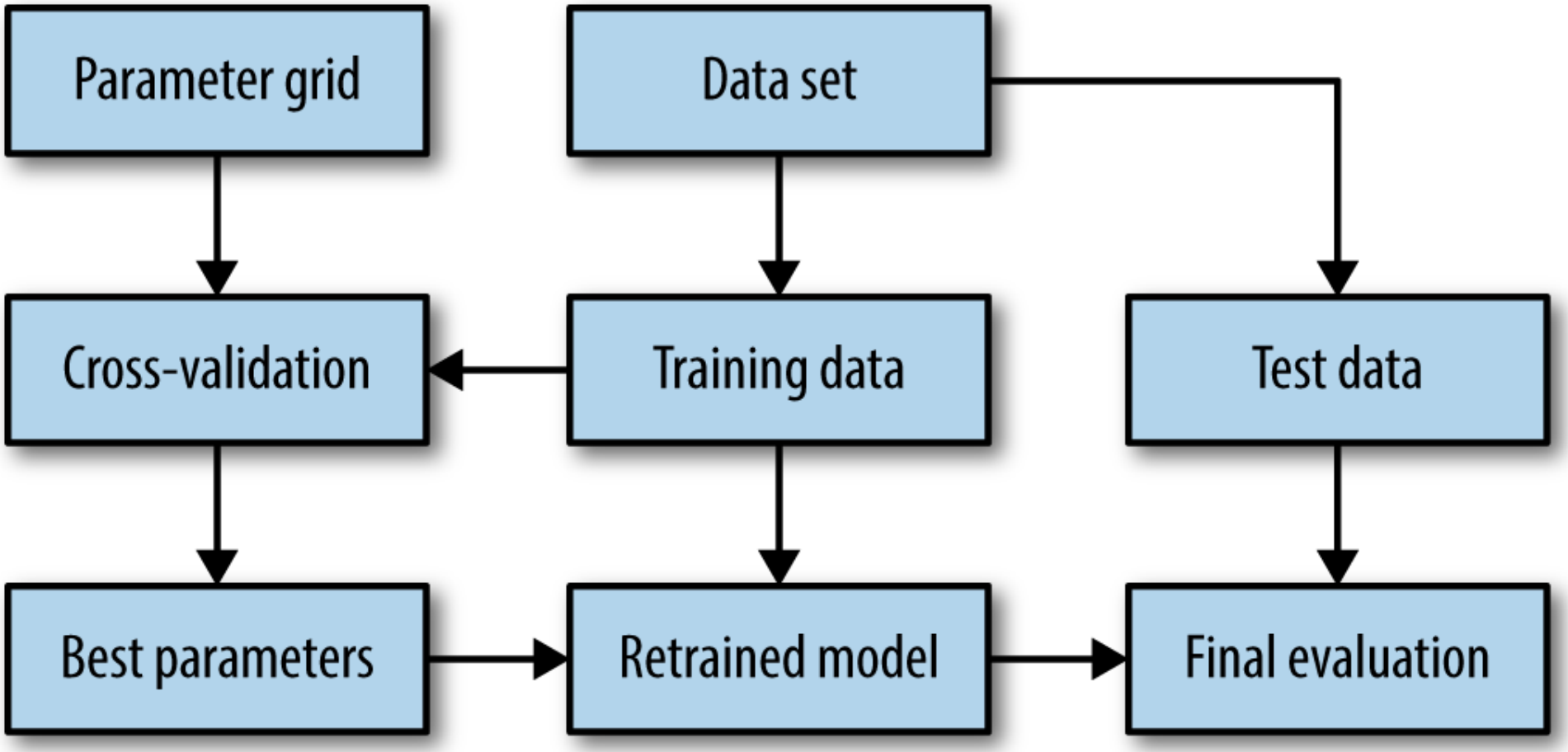
Pro: more stable, more data
con: slower

Cross-validation + test-set

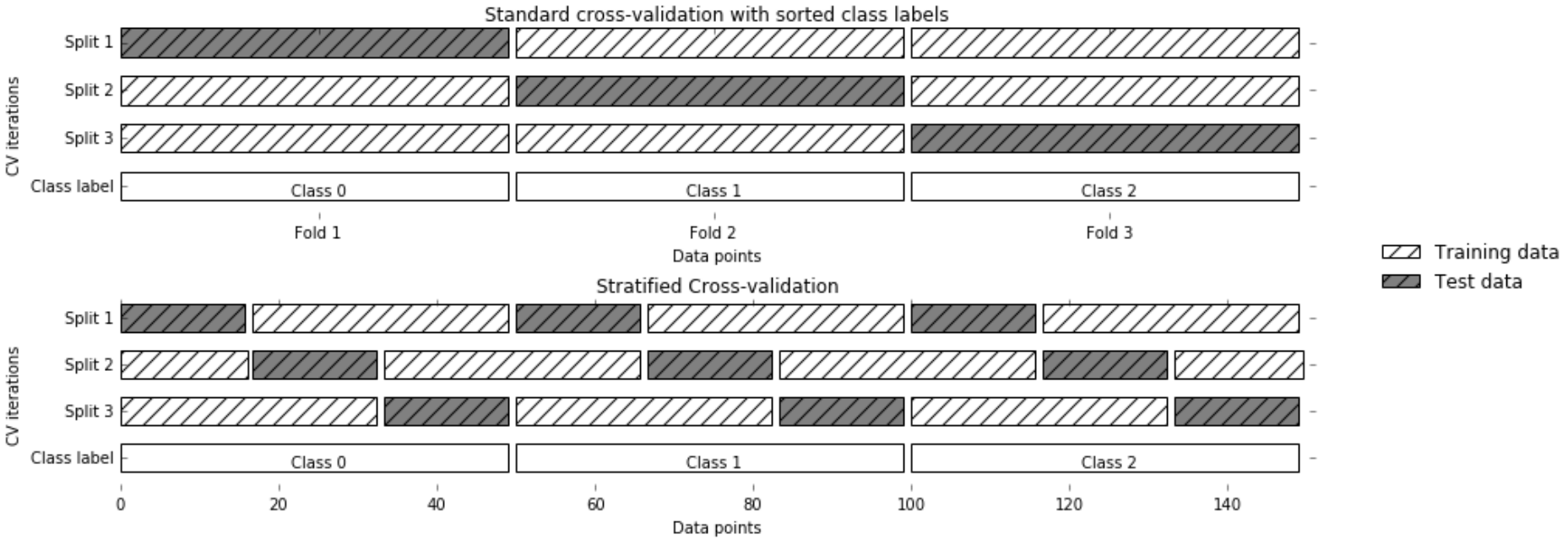


```
from sklearn.model_selection import cross_val_score

X_train, X_test, y_train, y_test = train_test_split(X, y)

cross_val_scores = []

for i in neighbors:
    knn = KNeighborsClassifier(n_neighbors=i)
    scores = cross_val_score(knn, X_trainval, y_trainval, cv=10)
    cross_val_scores.append(np.mean(scores))

print("best cross-validation score: {:.3f}".format(np.max(cross_val_scores)))
best_n_neighbors = neighbors[np.argmax(cross_val_scores)]
print("best n_neighbors: {}".format(best_n_neighbors))

knn = KNeighborsClassifier(n_neighbors=best_n_neighbors)
knn.fit(X_train, y_train)
print("test-set score: {:.3f}".format(knn.score(X_test, y_test)))
```

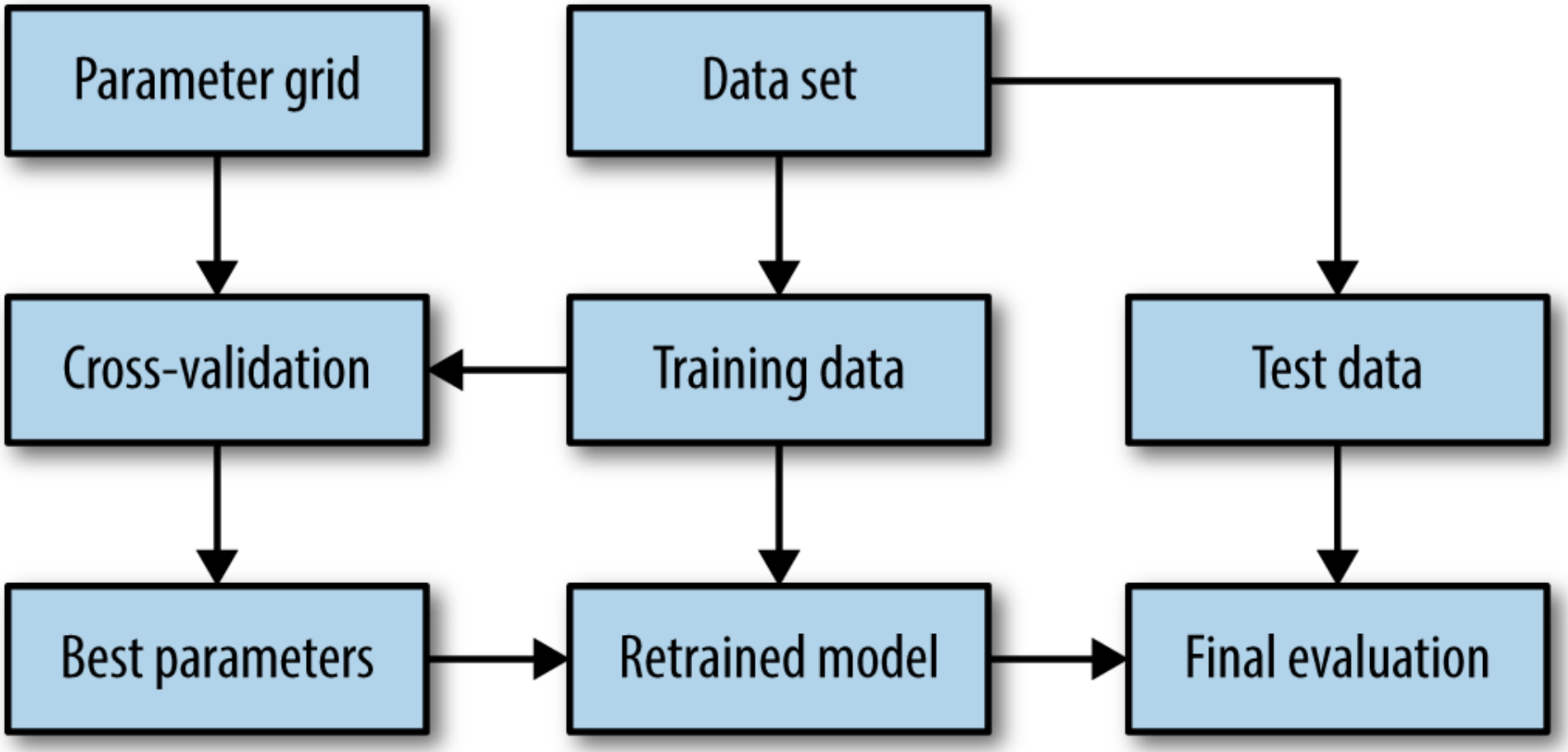
```
best cross-validation score: 0.972
best n_neighbors: 3
test-set score: 0.972
```


Parameter grid
Data set
Cross-validation
Training data
Test data
Best parameters
Retrained model
Final evaluation

StratifiedKFold



Stratified:
Ensure relative class frequencies in each fold reflect relative class frequencies on the whole dataset.


Parameter grid
Data set
Cross-validation
Training data
Test data
Best parameters
Retrained model
Final evaluation

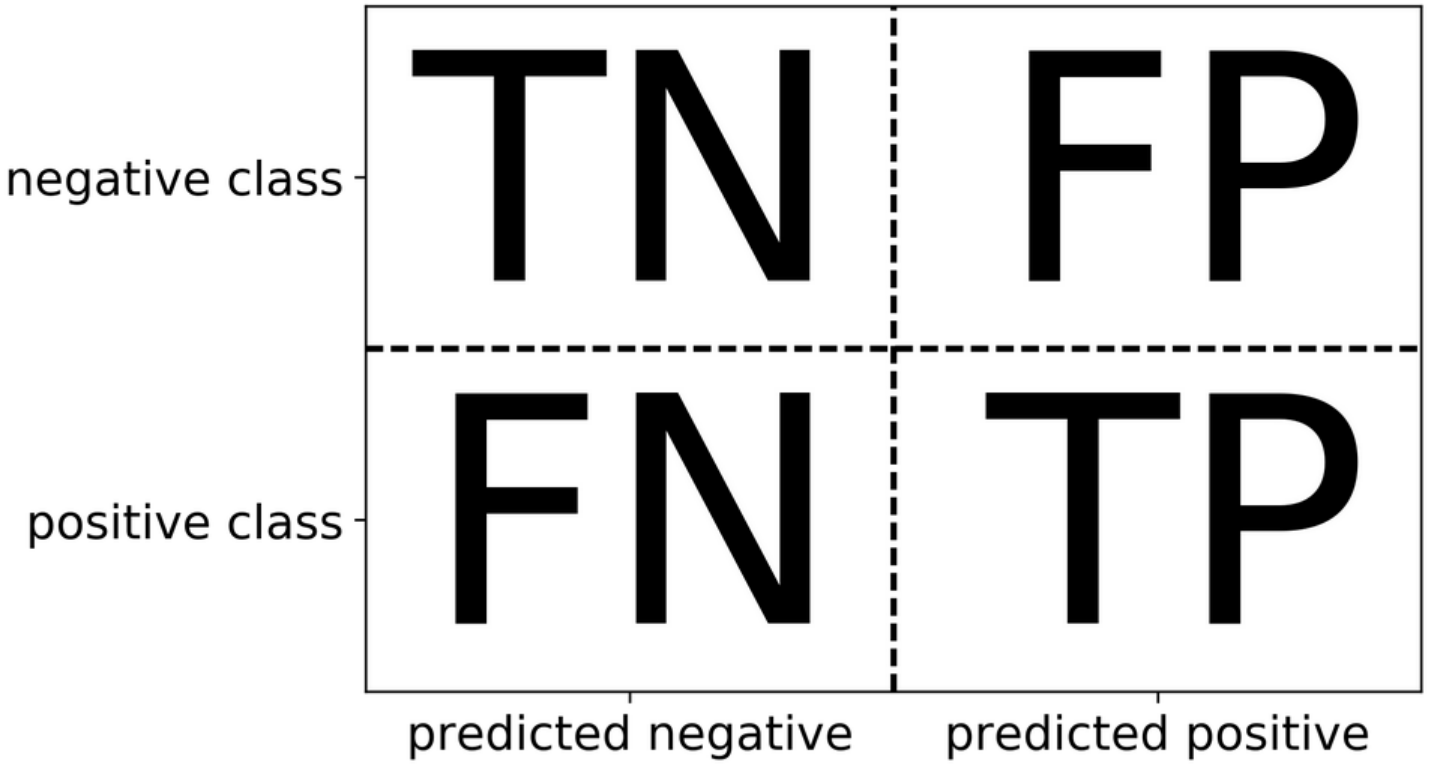
Grid-Search

```
from sklearn.model_selection import GridSearchCV

X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y)


param_grid = {'n_neighbors':  np.arange(1, 15, 2)}
grid = GridSearchCV(KNeighborsClassifier(), param_grid=param_grid, cv=10)
grid.fit(X_train, y_train)
print("best mean cross-validation score: {:.3f}".format(grid.best_score_))
print("best parameters: {}".format(grid.best_params_))

print("test-set score: {:.3f}".format(grid.score(X_test, y_test)))
```

```
best mean cross-validation score: 0.967
best parameters: {'n_neighbors': 7}
test-set score: 0.951
```

Metrics for Binary Classification

Review: confusion matrix



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

Diagonal divided by everything.

```python
from sklearn.datasets import load_breast_cancer
from sklearn.linear_model import LogisticRegression
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)

from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test, y_pred))
print(lr.score(X_test, y_test))
```

```
[[49  4]
 [ 5 85]]
0.937062937063
```

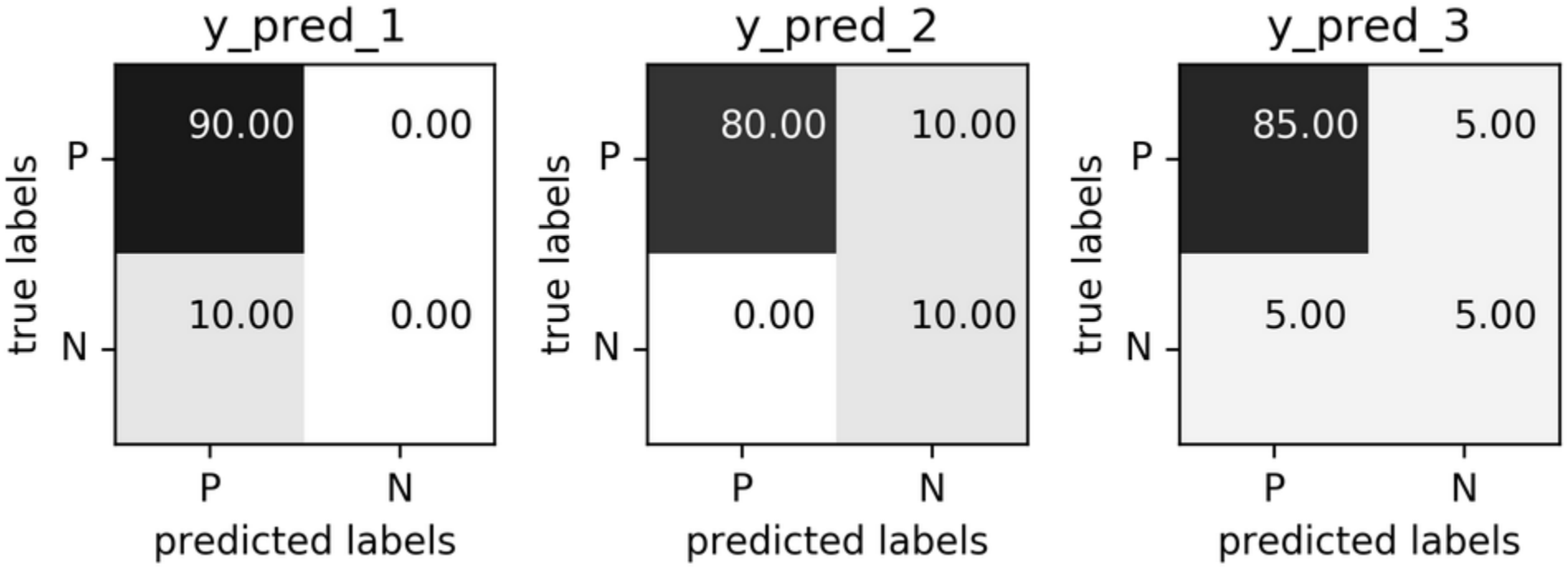
Problems with accuracy

- Imbalanced classes lead to hard-to-interpret accuracy:

```
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
```

Data with 90% negatives



Precision, Recall, f-score

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

Positive Predicted Value (PPV)

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

Sensitivity, coverage, true positive rate.

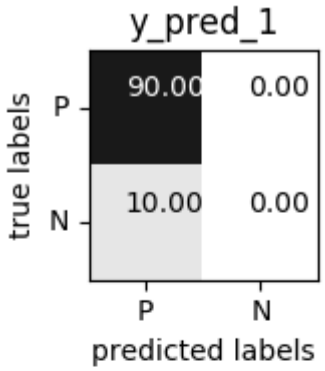
$$\text{F} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

All depend on definition of positive and negative!

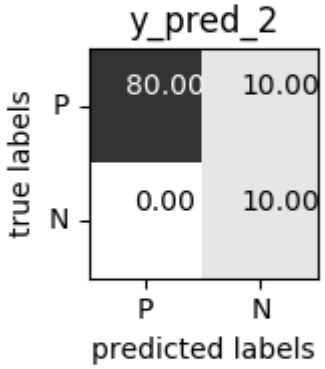
The zoo

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

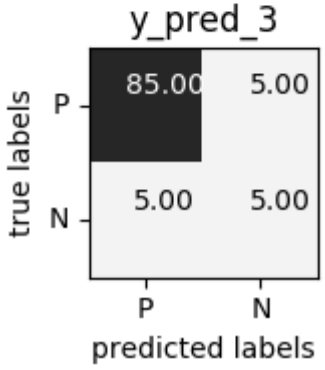
https://en.wikipedia.org/wiki/Precision_and_recall



	precision	recall	f1-score	support
0	0.90	1.00	0.95	90
1	0.00	0.00	0.00	10
avg / total	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
avg / total	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
avg / total	0.90	0.90	0.90	100

Goal setting!

- What do I want? What do I care about?

  (precision, recall, something else)
- Can I assign costs to the confusion matrix?

  (i.e. a false positive costs me $10, a false negative $100)
- What guarantees do we want to give?

Changing Thresholds

```
# logistic regresson on breast cancer, but change threshold:
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(classification_report(y_test, y_pred))
```

```
             precision    recall  f1-score   support

          0       0.91      0.92      0.92        53
          1       0.96      0.94      0.95        90

avg / total       0.94      0.94      0.94       143
```

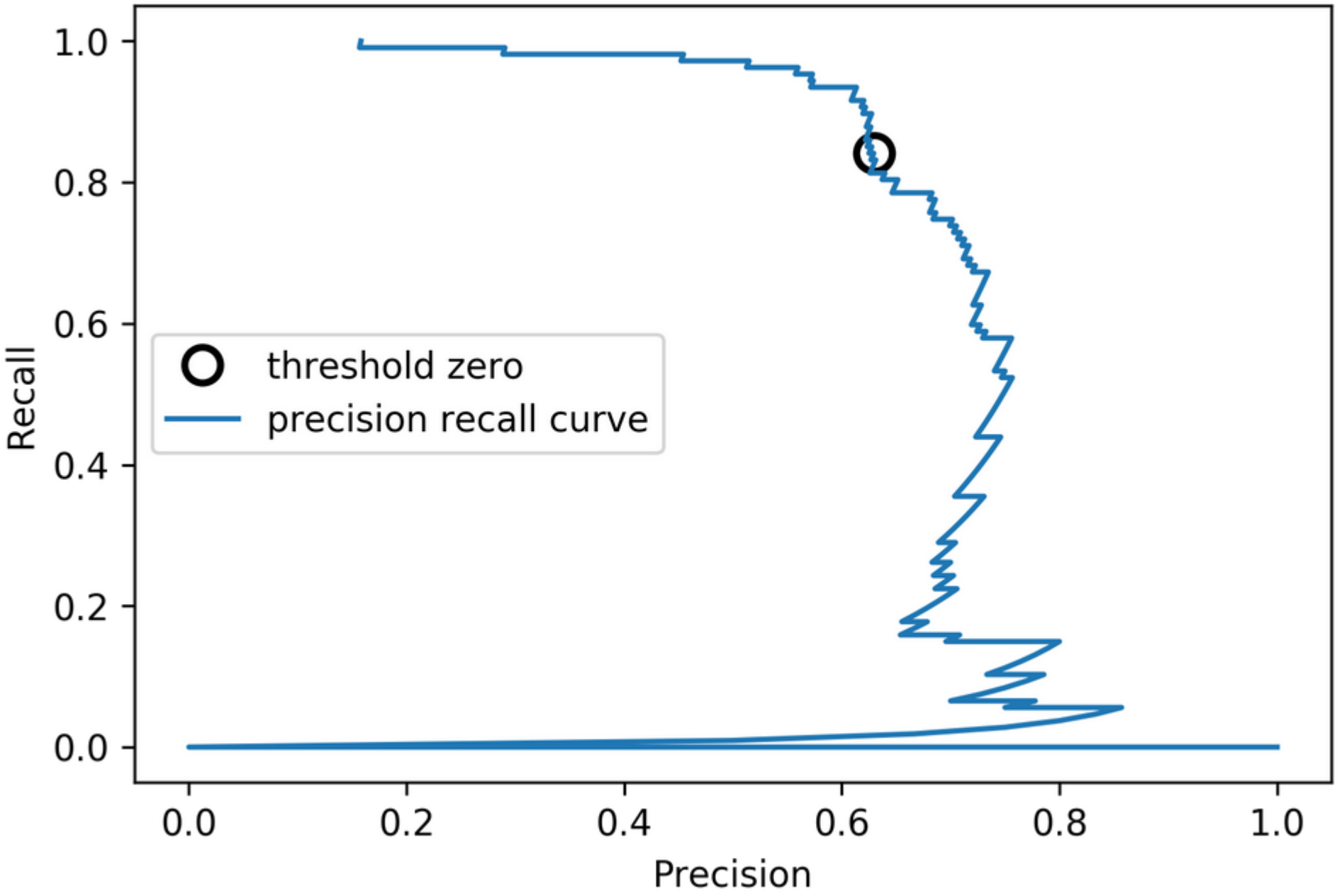
```
y_pred = lr.predict_proba(X_test)[:, 1] > .85

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

```
             precision    recall  f1-score   support

          0       0.84      1.00      0.91        53
          1       1.00      0.89      0.94        90

avg / total       0.94      0.93      0.93       143
```

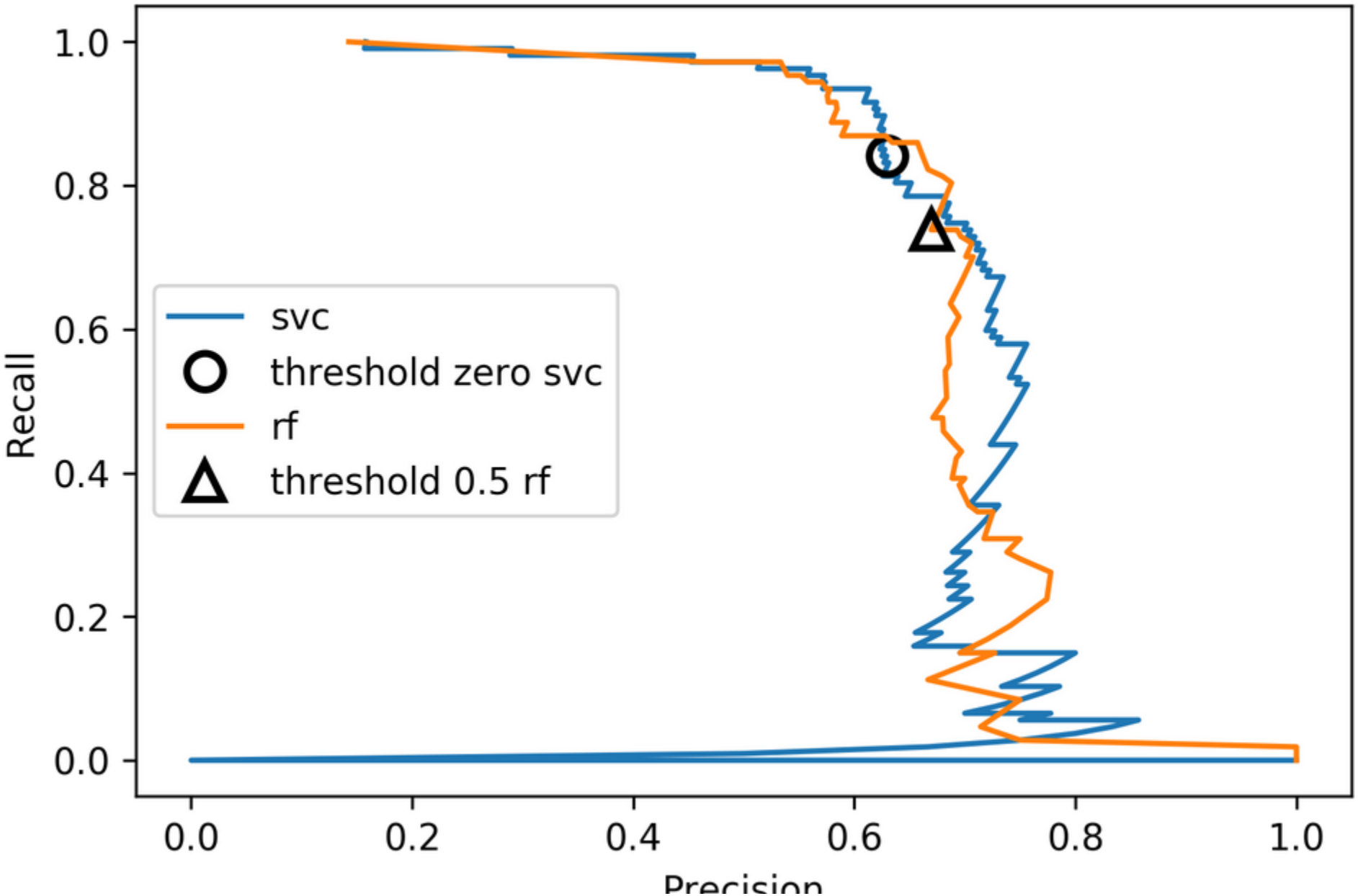
Precision-Recall Curve

```
X, y = make_blobs(n_samples=5000, centers=2, cluster_std=[7.0, 2],
                  random_state=22, shuffle=False)
# make classes imbalanced
X, y = X[:3000], y[:3000]

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

svc = SVC(gamma=.05).fit(X_train, y_train)

precision, recall, thresholds = precision_recall_curve(
    y_test, svc.decision_function(X_test))
# find threshold closest to zero:
close_zero = np.argmin(np.abs(thresholds))
plt.plot(precision[close_zero], recall[close_zero], 'o', markersize=10,
         label="threshold zero", fillstyle="none", c='k', mew=2)
```

Precision-Recall Curve



Comparing RF and SVC



Average Precision

Precision at threshold k

$$\mathrm{AveP} = \sum_{k=1}^{n} P(k)\Delta r(k)$$

Change in recall between k and k-1

Sum over data points, ranked by decision function

Same as area under the precision-recall curve
(depending on how you treat edge-cases)

F1 vs average precision

```
from sklearn.metrics import f1_score

print("f1_score of random forest: {:.3f}".format(
      f1_score(y_test, rf.predict(X_test))))

print("f1_score of svc: {:.3f}".format(f1_score(y_test, svc.predict(X_test))))
```

```
f1_score of random forest: 0.709
f1 score of svc: 0.715
```

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

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

AP only considers ranking!

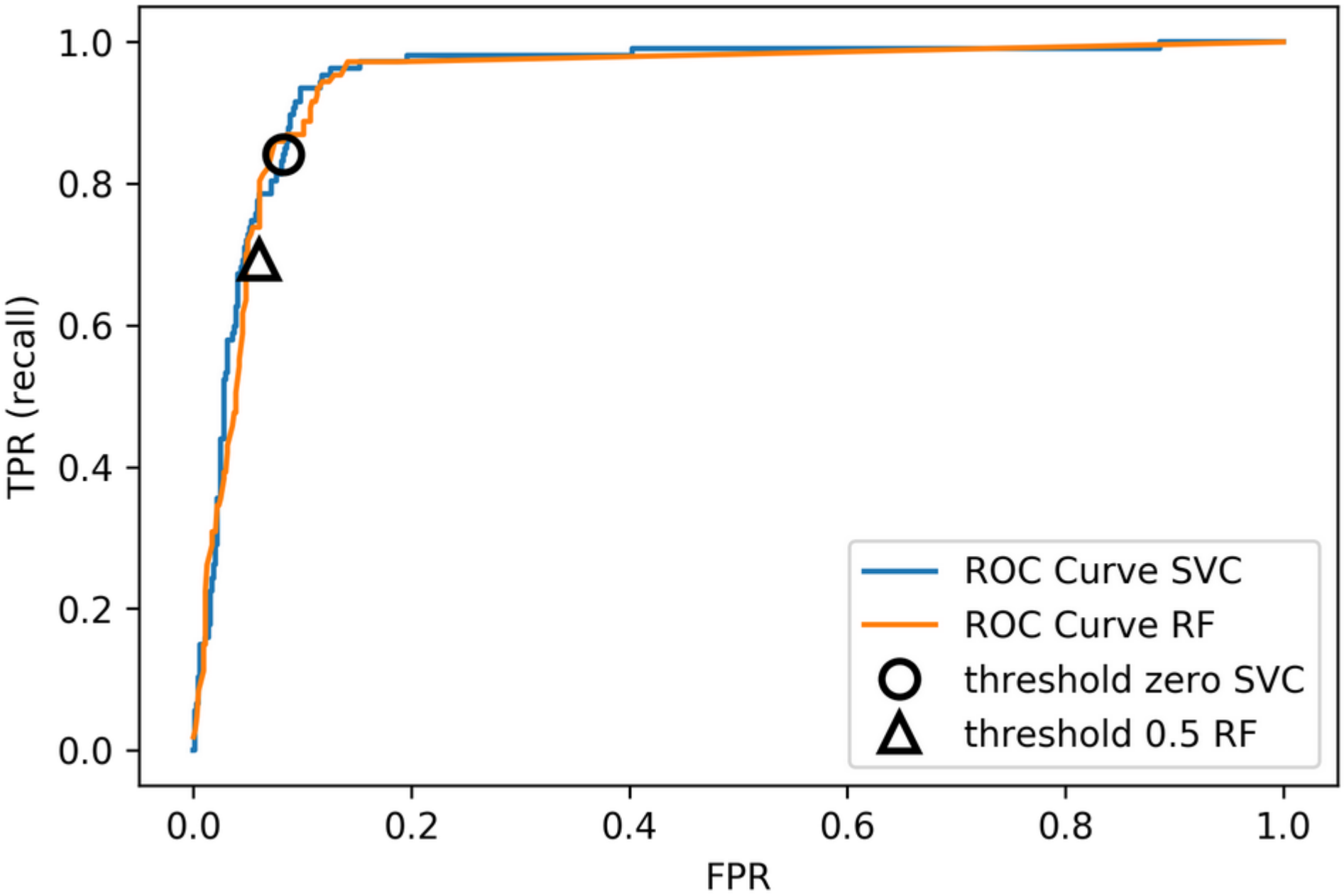
ROC Curve

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

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

= recall


1.0
0.8
0.6
0.4
0.2
0.0
TPR (recall)
0.0
0.2
0.4
0.6
0.8
1.0
FPR
ROC Curve SVC
ROC Curve RF
threshold zero SVC
threshold 0.5 RF

ROC AUC

- Area under ROC Curve
- Always .5 for random / constant prediction

```
from sklearn.metrics import roc_auc_score
rf_auc = roc_auc_score(y_test, rf.predict_proba(X_test)[:, 1])
svc_auc = roc_auc_score(y_test, svc.decision_function(X_test))
print("AUC for Random Forest: {:.3f}".format(rf_auc))
print("AUC for SVC: {:.3f}".format(svc_auc))
```

```
AUC for Random Forest: 0.937
AUC for SVC: 0.916
```

The Relationship Between Precision-Recall and ROC Curves
https://www.biostat.wisc.edu/~page/rocpr.pdf

Multi-class classification

Confusion Matrix

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

digits = load_digits()

X_train, X_test, y_train, y_test = train_test_split(
    digits.data, 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)))
print("Confusion matrix:\n{}".format(confusion_matrix(y_test, pred)))
```

```
Accuracy: 0.953
Confusion matrix:
[[37  0  0  0  0  0  0  0  0  0]
 [ 0 39  0  0  0  0  2  0  2  0]
 [ 0  0 41  3  0  0  0  0  0  0]
 [ 0  0  1 43  0  0  0  0  0  1]
 [ 0  0  0  0 38  0  0  0  0  0]
 [ 0  1  0  0  0 47  0  0  0  0]
 [ 0  0  0  0  0  0 52  0  0  0]
 [ 0  1  0  1  1  0  0 45  0  0]
 [ 0  3  1  0  0  0  0  0 43  1]
 [ 0  0  0  1  0  1  0  0  1 44]]
```

Normalizing confusion matrix (by rows) can be helpful

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

	precision	recall	f1-score	support
0	1.00	1.00	1.00	37
1	0.89	0.91	0.90	43
2	0.95	0.93	0.94	44
3	0.90	0.96	0.92	45
4	0.97	1.00	0.99	38
5	0.98	0.98	0.98	48
6	0.96	1.00	0.98	52
7	1.00	0.94	0.97	48
8	0.93	0.90	0.91	48
9	0.96	0.94	0.95	47
avg / total	0.95	0.95	0.95	450

Micro and Macro F1

- Macro-average f1: Average f1 scores over classes
- Micro-average f1: Average binary confusion matrices, then compute recall, precision once

```
print("Micro average f1 score: {:.3f}".format(f1_score(y_test, pred, average="micro")))
print("Macro average f1 score: {:.3f}".format(f1_score(y_test, pred, average="macro")))
```

```
Micro average f1 score: 0.953
Macro average f1 score: 0.954
```

Macro: “all classes are equally important”
Micro: “all samples are equally important” - same for other metric averages

Picking metrics?

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

Using metrics in cross-validation

```
X = digits.data
y = digits.target == 9

# default scoring for classification is accuracy
scores_default = cross_val_score(SVC(), X, y)

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

# using ROC AUC
roc_auc =  cross_val_score(SVC(), X, digits.target == 9, scoring="roc_auc")

print("Default scoring: {}".format(scores_default))
print("Explicit accuracy scoring: {}".format(explicit_accuracy))
print("AUC scoring: {}".format(roc_auc))
```

```
Default scoring: [ 0.9  0.9  0.9]
Explicit accuracy scoring: [ 0.9  0.9  0.9]
AUC scoring: [ 0.994  0.99   0.996]
```

Same for GridSearchCV
Will make GridSearchCV.score use your metric!

Built-in scoring

- “scoring” can be string or callable.
- Strings:

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

```
accuracy
adjusted_mutual_info_score
adjusted_rand_score
average_precision
completeness_score
f1
f1_macro
f1_micro
f1_samples
f1_weighted
fowlkes_mallows_score
homogeneity_score
log_loss
mean_absolute_error
mean_squared_error
median_absolute_error
mutual_info_score
neg_log_loss
neg_mean_absolute_error
neg_mean_squared_error
neg_mean_squared_log_error
neg_median_absolute_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
v_measure_score
```

Metrics for regression models

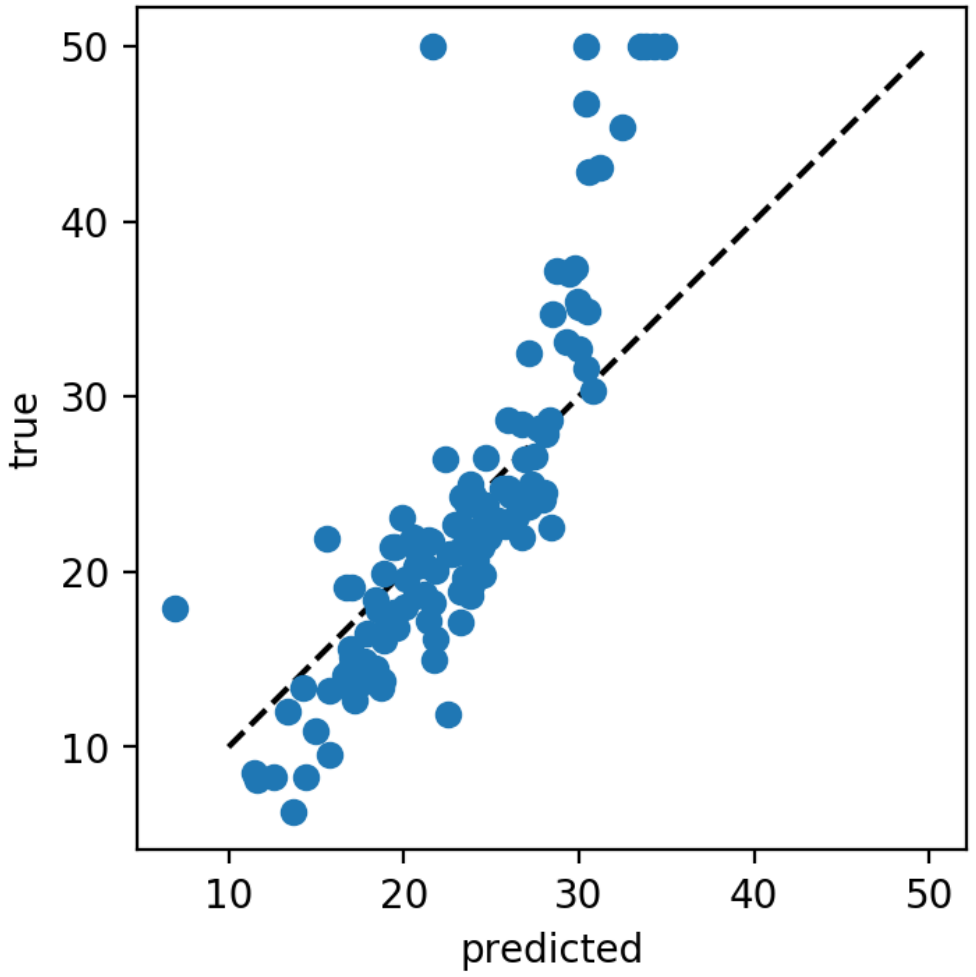
Build-in standard metrics

- R^2 : easy to understand scale
- MSE : easy to relate to input
- Mean absolute error, median absolute error: more robust.

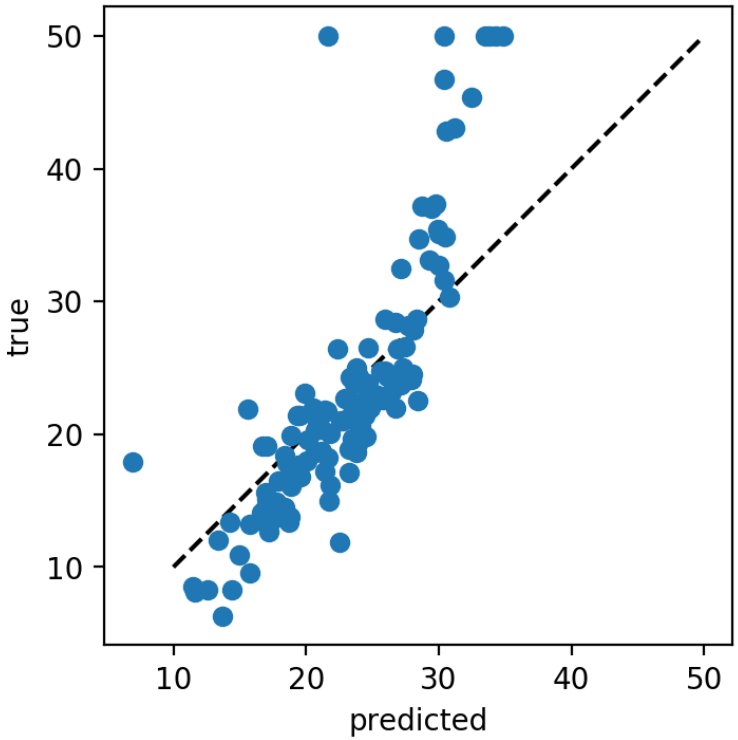
- When using “scoring” use “neg_mean_squared_error” etc

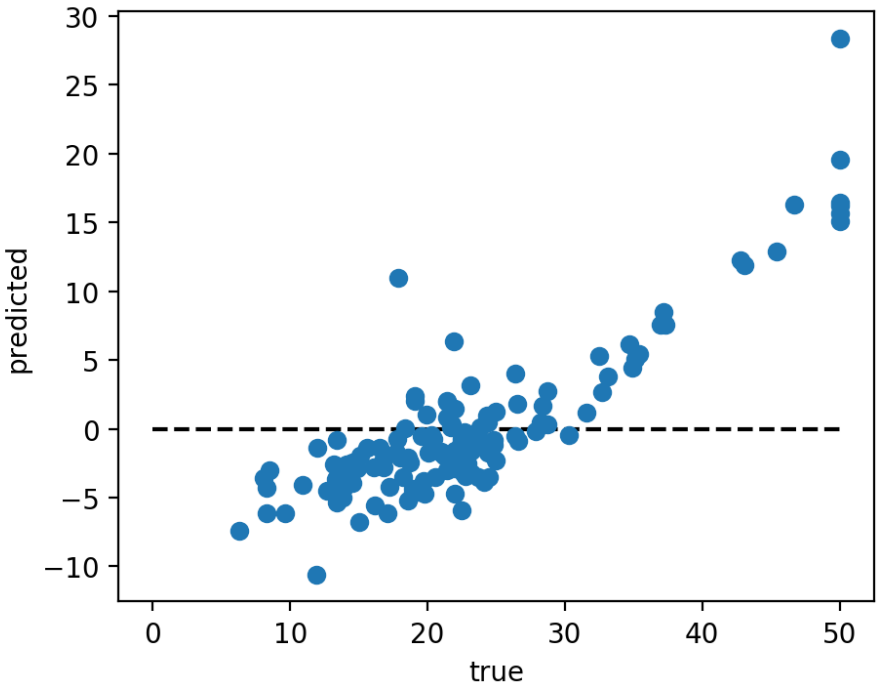
Prediction plots

```
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).fit(X_train, y_train)
pred = ridge.predict(X_test)
plt.plot(pred, y_test, 'o')
```

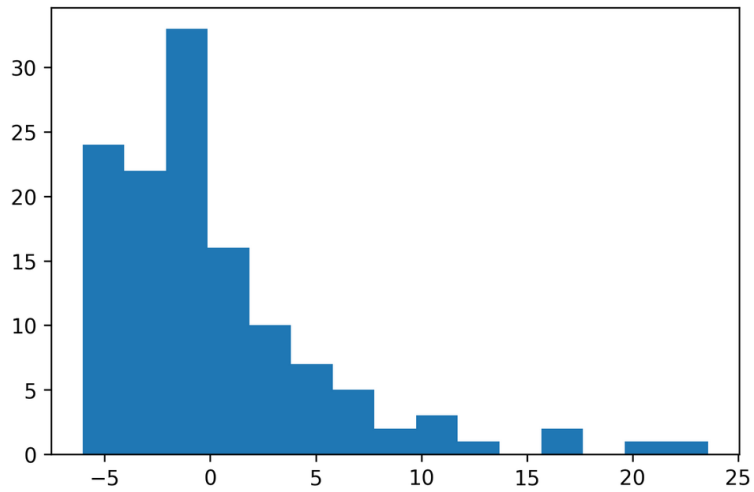


Residual Plots

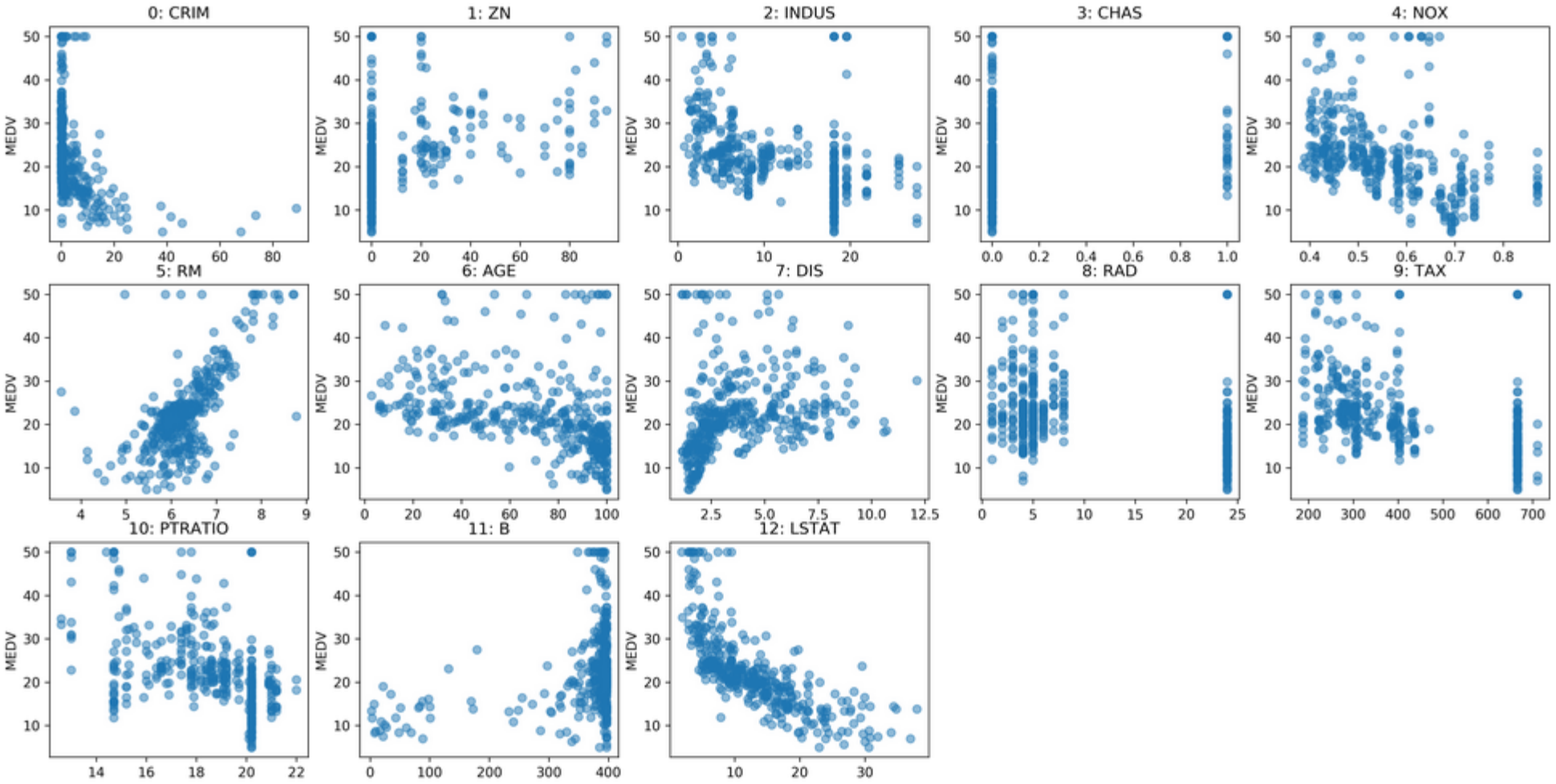





```
plt.hist(y_test - pred, bins="auto");
```



Target vs Feature



Absolute vs relative: MAPE

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

Over vs under

- Overprediction and underprediction can have different cost.
- Try to create cost-matrix: how much does overprediction and underprediction cost?
- Is it linear?