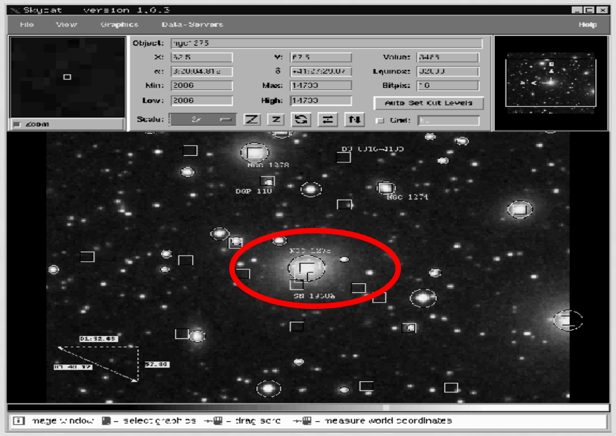
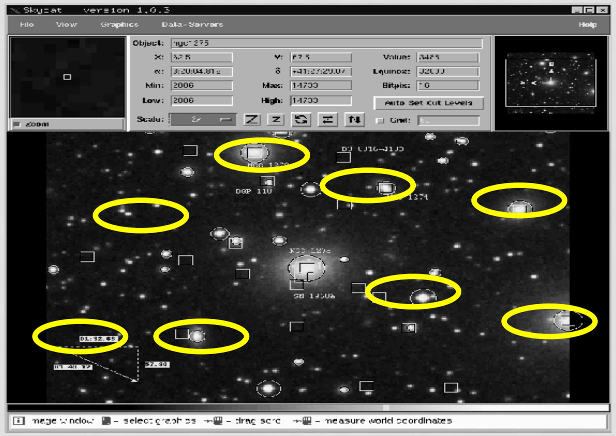
We started with this workflow:



?

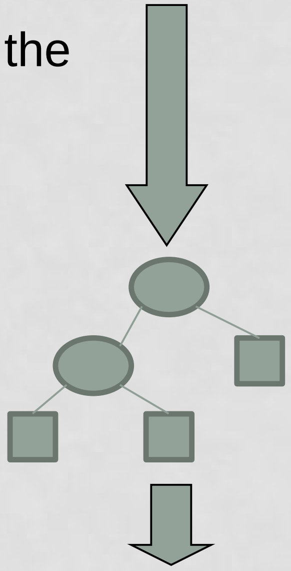
Use of the model

Training data



Method

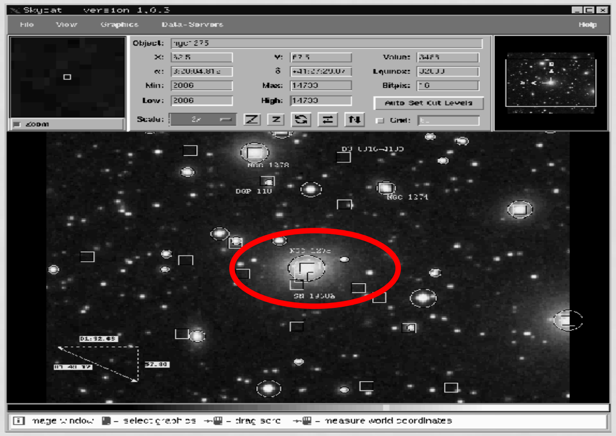
Model



Spiral galaxy

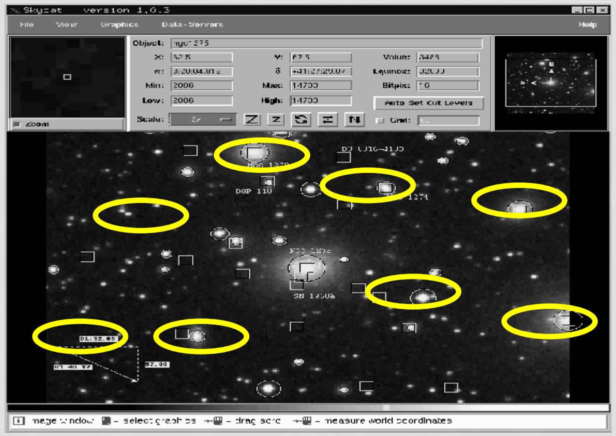
MODEL EVALUATION: ESTIMATION OF FUTURE MODEL PERFORMANCE

We started with this workflow:



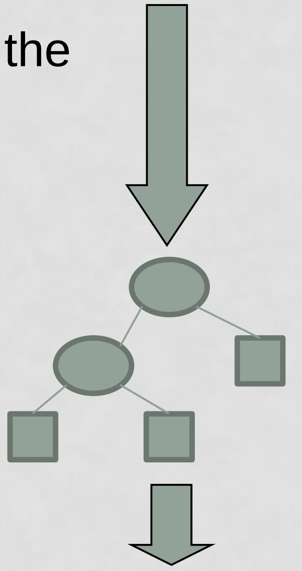
Use of the model

Training data



Method

Model



Spiral galaxy

The model must **generalize** beyond the training data and work well for new instances, different to the ones in the training data

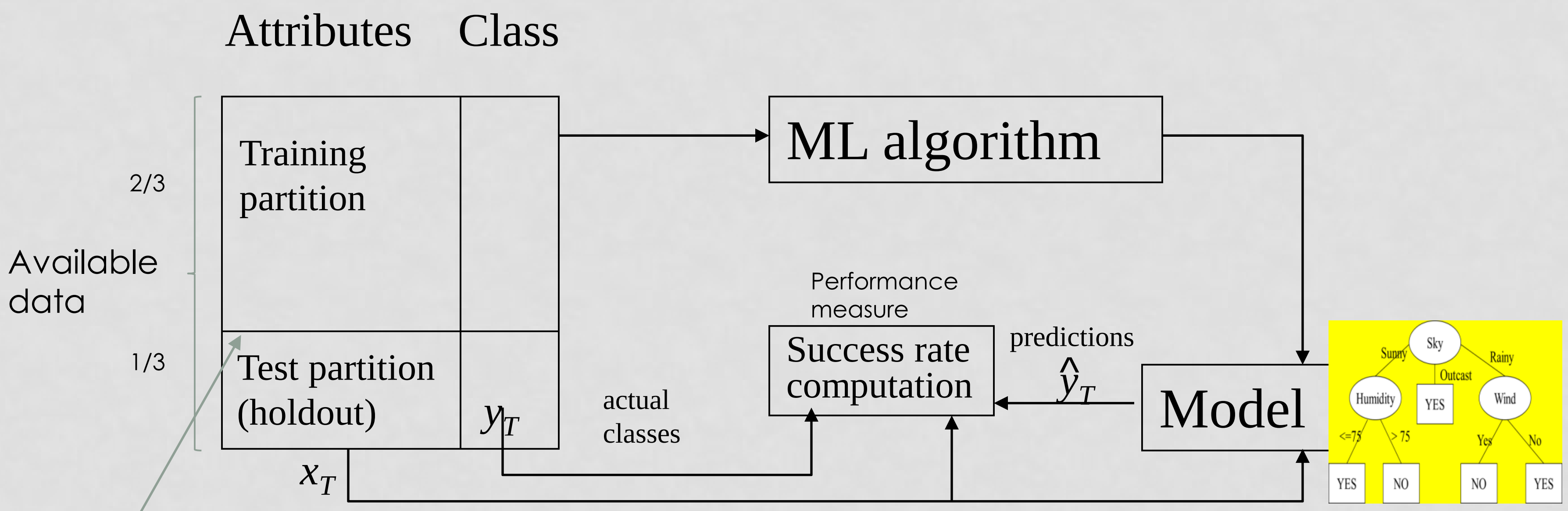
ESTIMATION OF FUTURE MODEL PERFORMANCE (EVALUATION)

1. Performance: a measure of how well your model does (e.g. classification accuracy for classification tasks)
2. Why? It is not guaranteed that the model will generalize well (because of overfitting, which can be solved to some extent by hyper-parameter tuning).
3. Why? Even if the model generalizes well, the data might have some irreducible error (for instance, if there is noise / class overlap in the data).
4. Why? Your company may want to know how well your model is going to perform in the future

MODEL EVALUATION

- In summary, we want two results:
  - A model
  - An estimation of its future performance / its performance on new data
- There are several strategies for estimating future performance (model evaluation), but the most widely used are:
  - Train / test (holdout)
  - Crossvalidation

TRAIN / TEST EVALUATION METHOD (A.K.A. HOLDOUT METHOD)

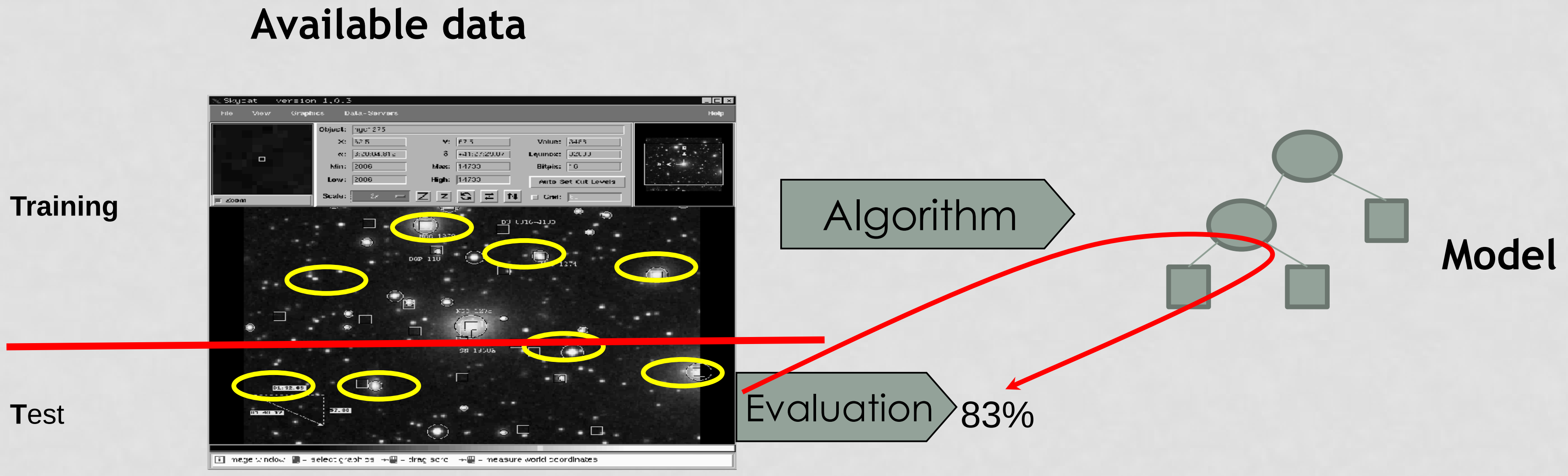


random split!
(== <u>shuffle</u> av. data before splitting)

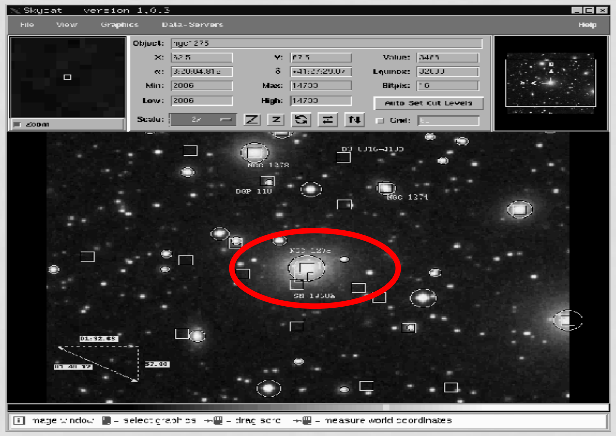
$$Success\ rate = \ Accuracy = \frac{1}{n}\sum_{k=1}^{n} y_k == \hat{y}_k$$

Rule: never evaluate a model with the same data used for training it

Train/test:
Now, in addition to training the model with the training partition, the model is evaluated with the test partition.



- Now, we get both the model **and** an estimation of its future performance
- Then, we can use the model.

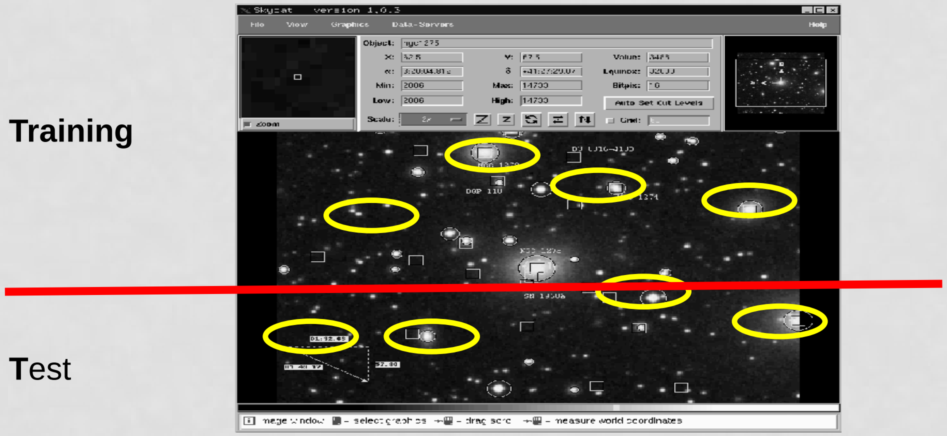


?

Use the model

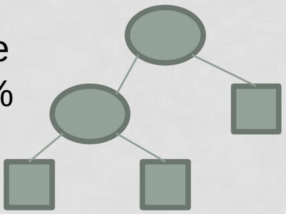
Available data

Training



Test

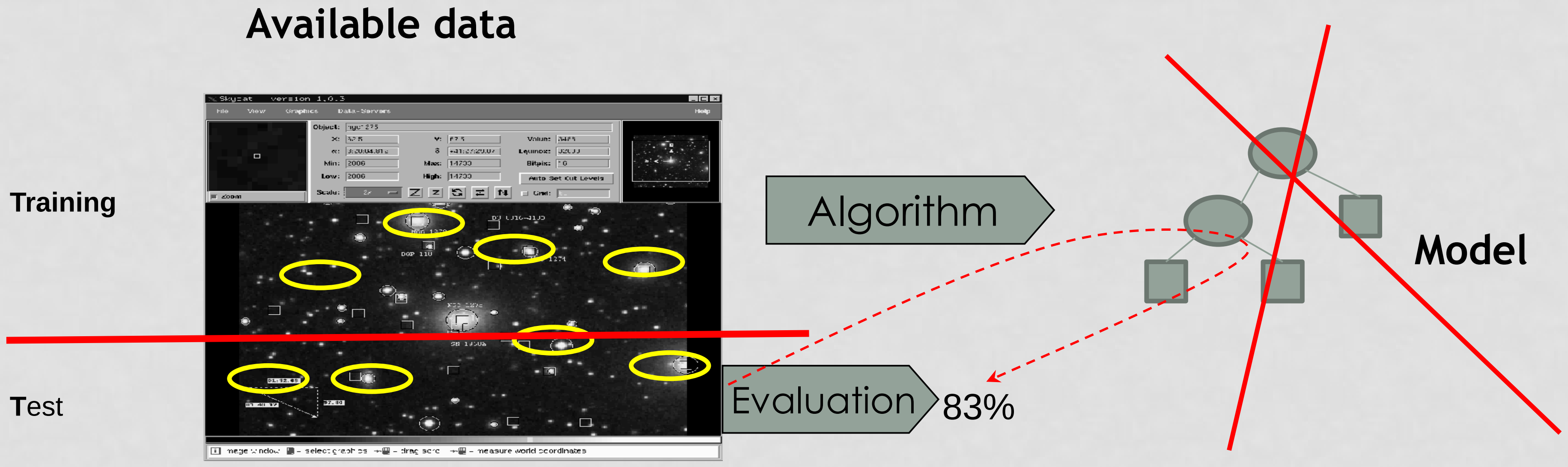
Estimation of future performance = 83%



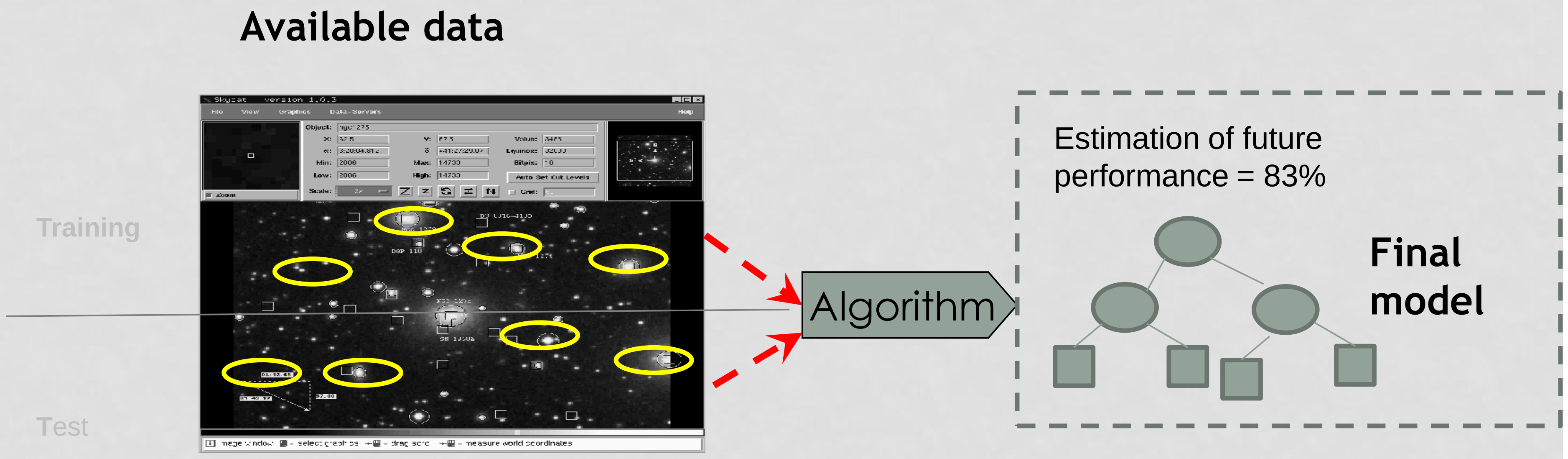
Model

Spiral galaxy

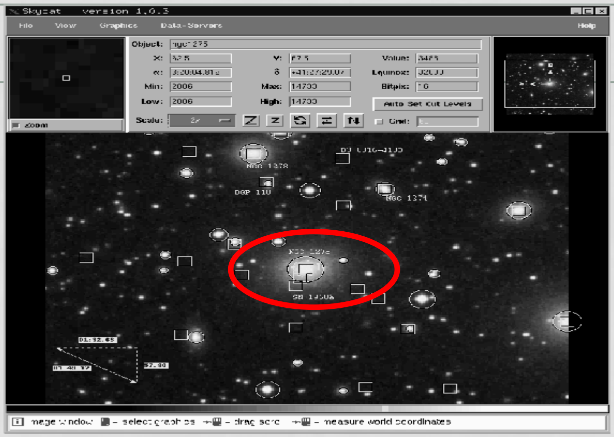
- However, it is common practice that, after the estimation of future performance (83%) has been obtained …
- The model used to obtain it is discarded and …



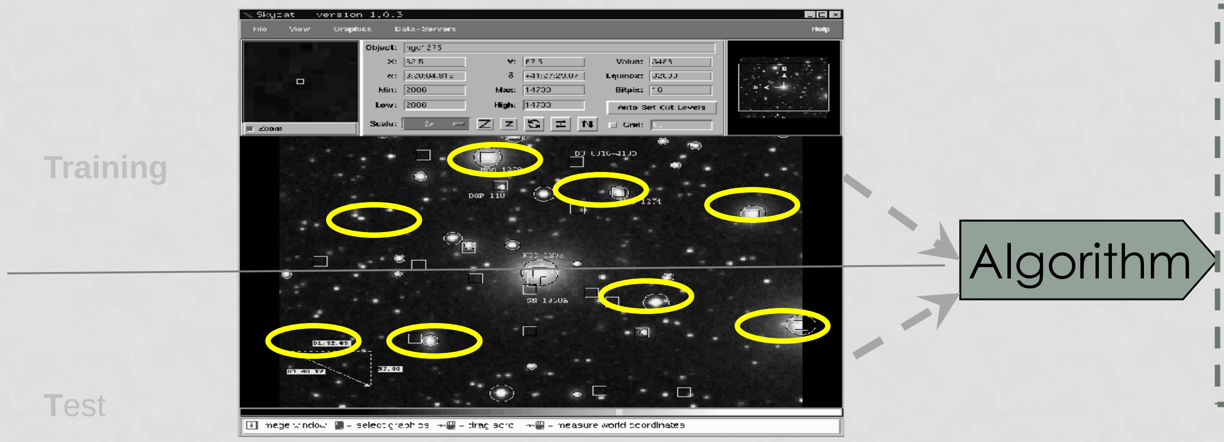
- … and a **final model** is trained with the **complete dataset** (available data = training and test partitions).
- The reason is that the more data is used to train the model, the better the model should be (at least, it shouldn't be worse).
- It is considered that the evaluation computed previously (83%), is also a good estimation of this new final model, and it is kept.
- In fact, it is considered that 83% is a **pessimistic evaluation**, because the final model is trained with a larger dataset than the one used previously for evaluation



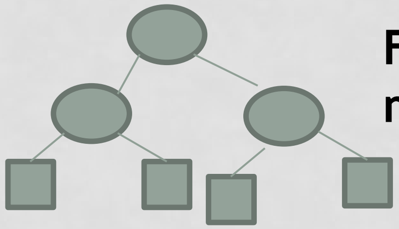
- Then, we can use this final model



Available data



Estimation of future performance = 83%

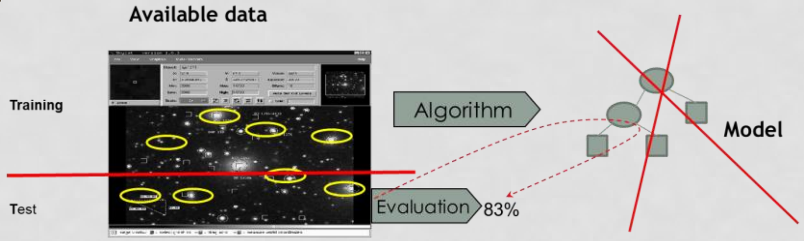


Spiral galaxy

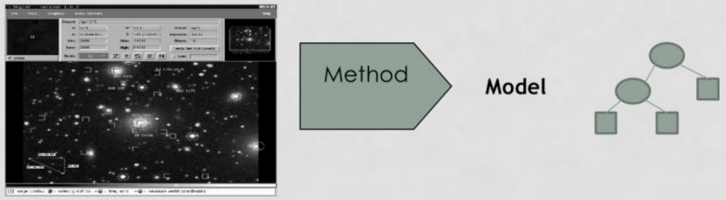
Summary

- Two ideas:

1. The goal of estimation of future performance / estimation of performance on new data / model evaluation is not getting a model, but getting a performance measure.

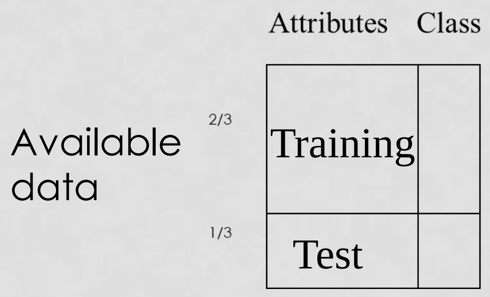


2. The final model is always trained using all available data.
   - In fact, if you are not interested in model evaluation, this is exactly what you would do: use all data for training the model.



On the size of the test partition

- 2/3 vs. 1/3 for train and test is arbitrary but commonly used.



- Good for thousands of instances on the testing partition. Probably, for millions of instances, 5% for test should be enough.
- Dilemma:
  - The larger the test, the more accurate model evaluation
  - But then, fewer instances are available for training the model

On the size of the test partition

- In order to estimate the required size of the test partition, can we estimate the uncertainty of the accuracy on the test set?
- Let's suppose that the test partition contains N instances, and that $\{x_i \mid i = 1:N\}$ is $x_i == 1$ if correct prediction by the model and $x_i == 0$ if wrong prediction
- Estimated accuracy of the model (success rate) for this test partition can be estimated as :

$$\hat{f} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

On the size of the test partition

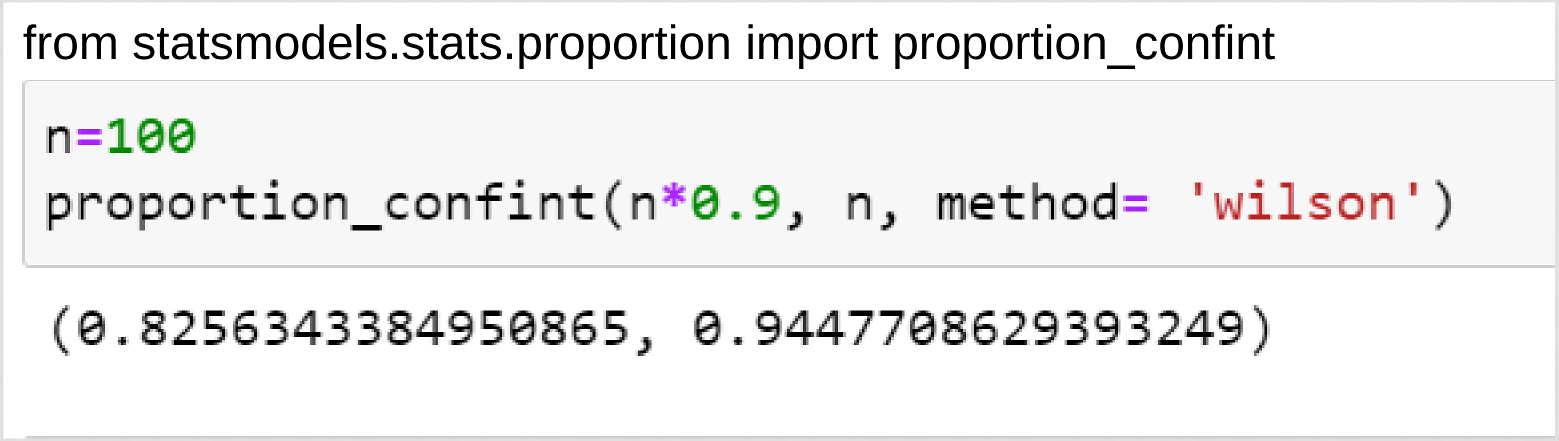
- Estimated accuracy (success rate) for this test partition can be estimated as :

$$\hat{f} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

- If the true accuracy is *f* ... ($N \to \infty \Rightarrow \hat{f} \to f$)
- then the number of observed successes of the model on the test partition ($N * \hat{f}$) follows a binomial with probability = *f*
- and confidence intervals around $\hat{f}$ can be estimated
- f ∈ [$\hat{f}$-l, $\hat{f}$+u] with 95% probability

On the size of the test partition

- For example, if $\hat{f} = 0.9$ (estimated on test set T) and the test set contains 100 instances, then the 95% confidence interval is [0.83 0.94]



```
from statsmodels.stats.proportion import proportion_confint
```

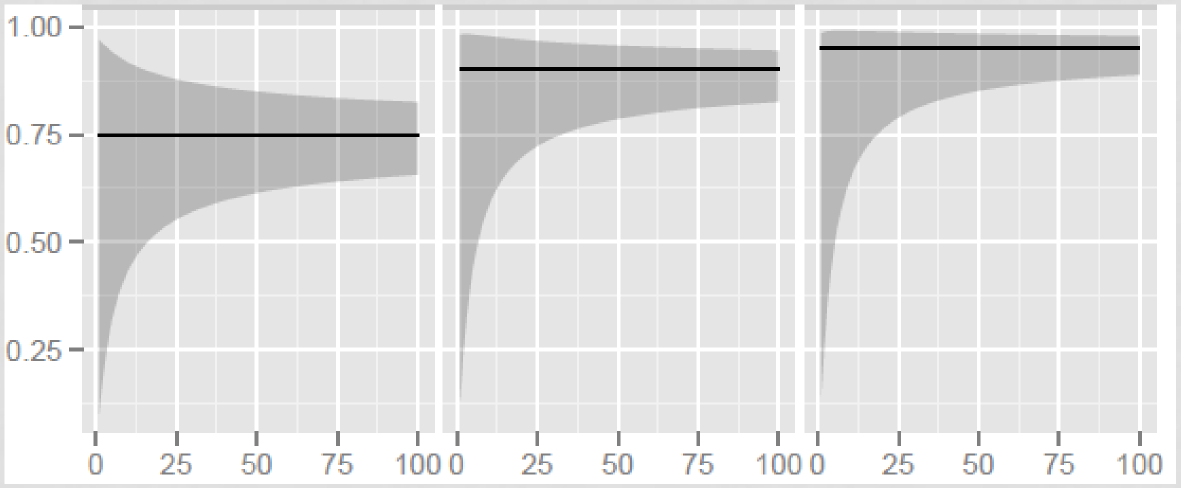
```
n=100
proportion_confint(n*0.9, n, method= 'wilson')
```

```
(0.8256343384950865, 0.9447708629393249)
```

On the size of the test partition

- The confidence interval size depends on $\hat{f}$ and N

$\hat{f} = 0.75$ $\hat{f} = 0.90$ $\hat{f} = 0.95$



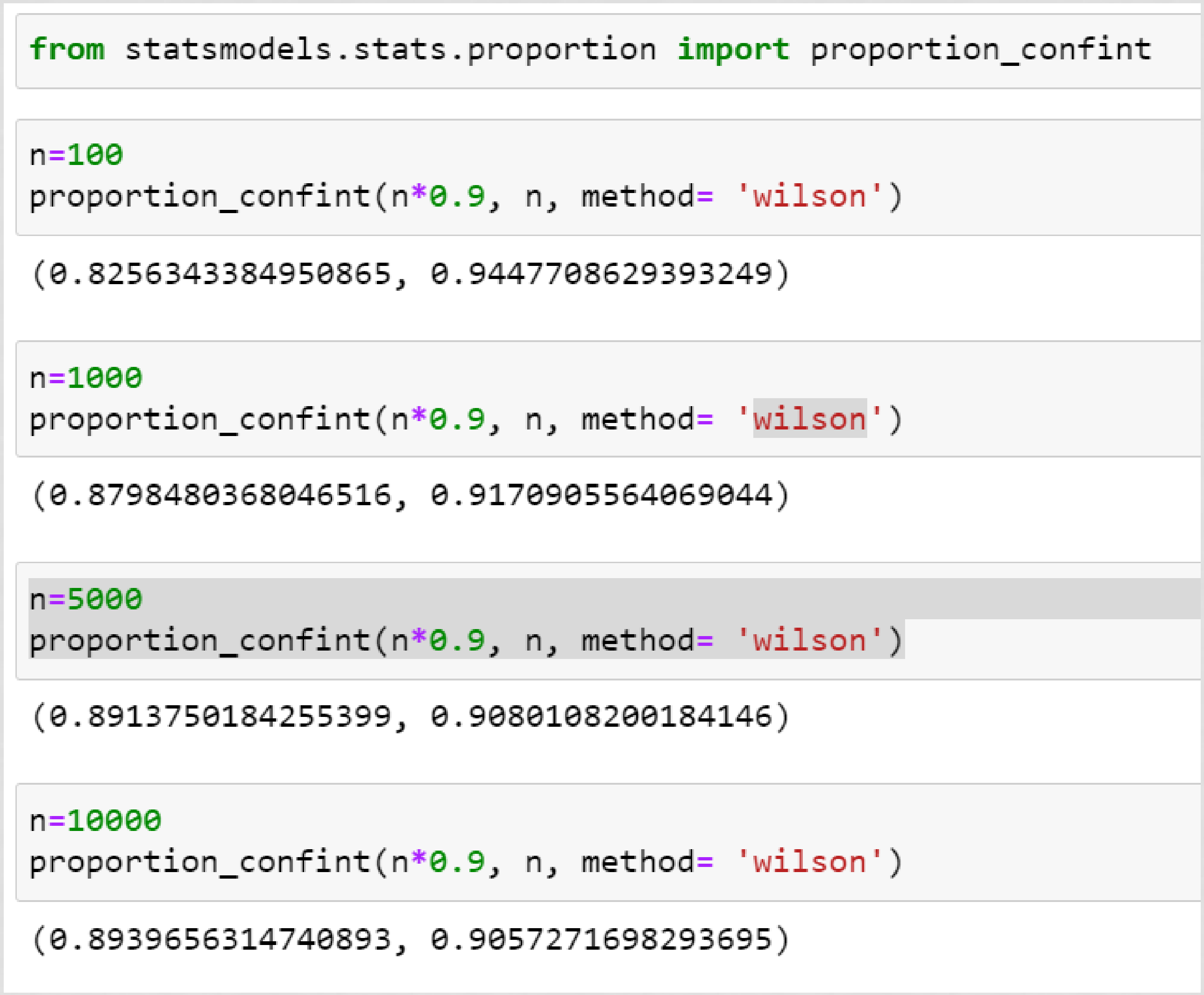
- More information: Beleites, C., Neugebauer, U., Bocklitz, T., Krafft, C., & Popp, J. (2013). Sample size planning for classification models. *Analytica chimica acta*, *760*, 25-33.

https://www.sciencedirect.com/science/article/pii/S0003267012016479?via%3Dihub

- And R code:

https://ars.els-cdn.com/content/image/1-s2.0-S0003267012016479-mmc3.pdf

RELATION BETWEEN THE WIDTH OF THE CONFIDENCE INTERVAL AND THE SIZE OF THE TEST PARTITION



```
from statsmodels.stats.proportion import proportion_confint
```

```
n=100
proportion_confint(n*0.9, n, method= 'wilson')
```

```
(0.8256343384950865, 0.9447708629393249)
```

```
n=1000
proportion_confint(n*0.9, n, method= 'wilson')
```

```
(0.8798480368046516, 0.9170905564069044)
```

```
n=5000
proportion_confint(n*0.9, n, method= 'wilson')
```

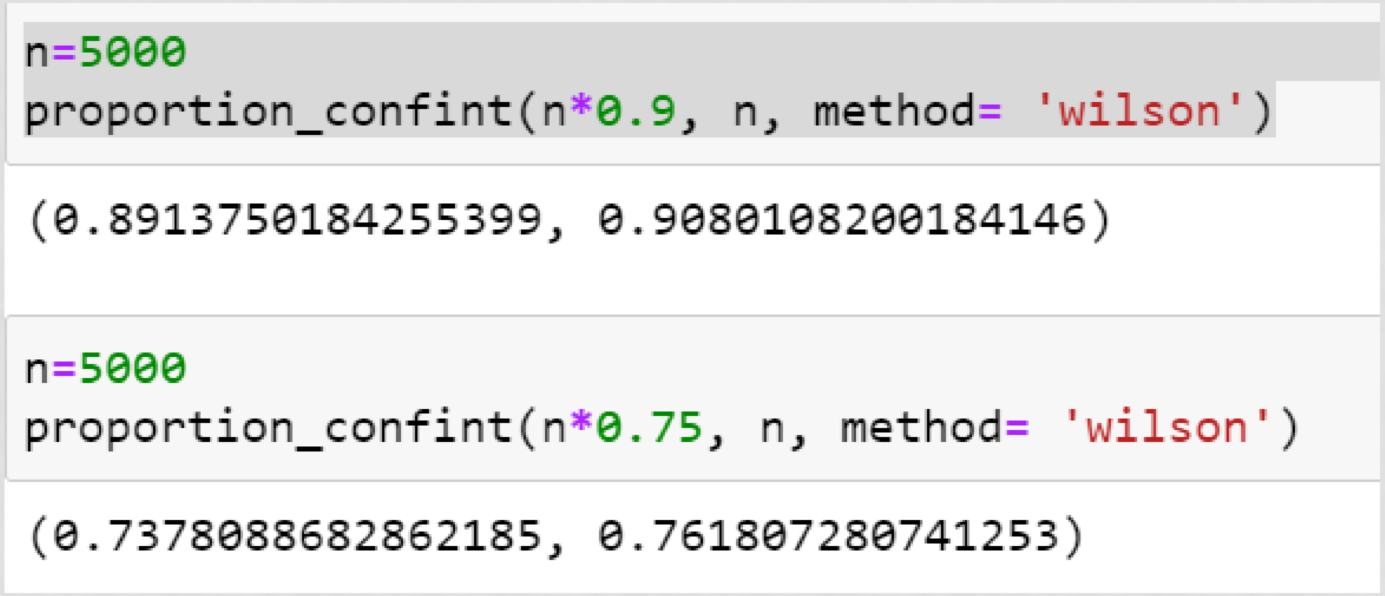
```
(0.8913750184255399, 0.9080108200184146)
```

```
n=10000
proportion_confint(n*0.9, n, method= 'wilson')
```

```
(0.8939656314740893, 0.9057271698293695)
```

RELATION BETWEEN THE WIDTH OF THE CONFIDENCE INTERVAL AND THE SIZE OF THE TEST PARTITION

The size of the confidence interval also depends on $\hat{f}$ (the larger $\hat{f}$, the narrower the interval)



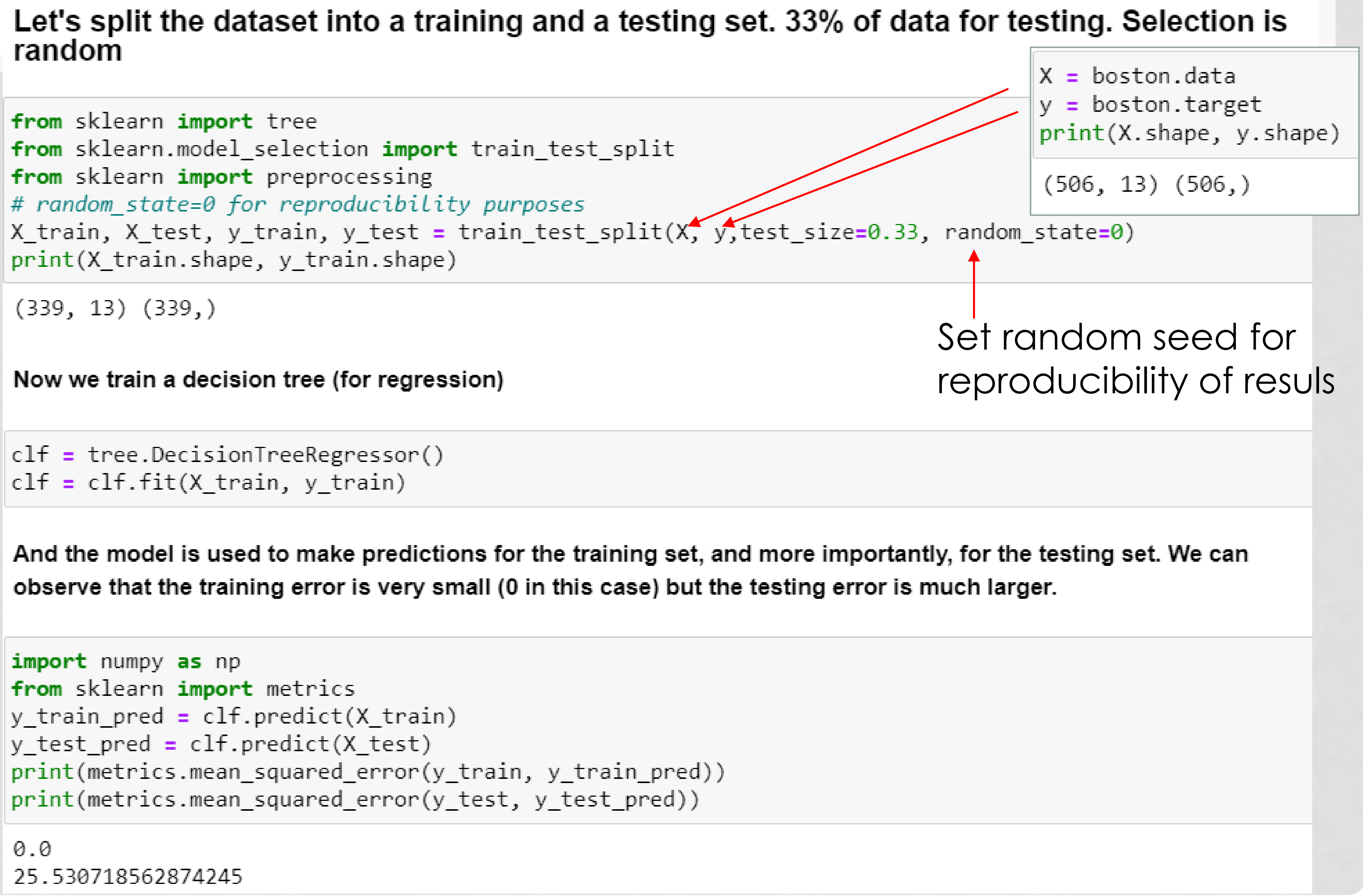
```
n=5000
proportion_confint(n*0.9, n, method= 'wilson')
```

```
(0.8913750184255399, 0.9080108200184146)
```

```
n=5000
proportion_confint(n*0.75, n, method= 'wilson')
```

```
(0.7378088682862185, 0.7618072807412539)
```

TRAIN-TEST IN SCIKITLEARN



Let's split the dataset into a training and a testing set. 33% of data for testing. Selection is random

```
X = boston.data
y = boston.target
print(X.shape, y.shape)
```

```
(506, 13) (506,)
```

```
from sklearn import tree
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
# random_state=0 for reproducibility purposes
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.33, random_state=0)
print(X_train.shape, y_train.shape)
```

```
(339, 13) (339,)
```

Set random seed for reproducibility of resuls

Now we train a decision tree (for regression)

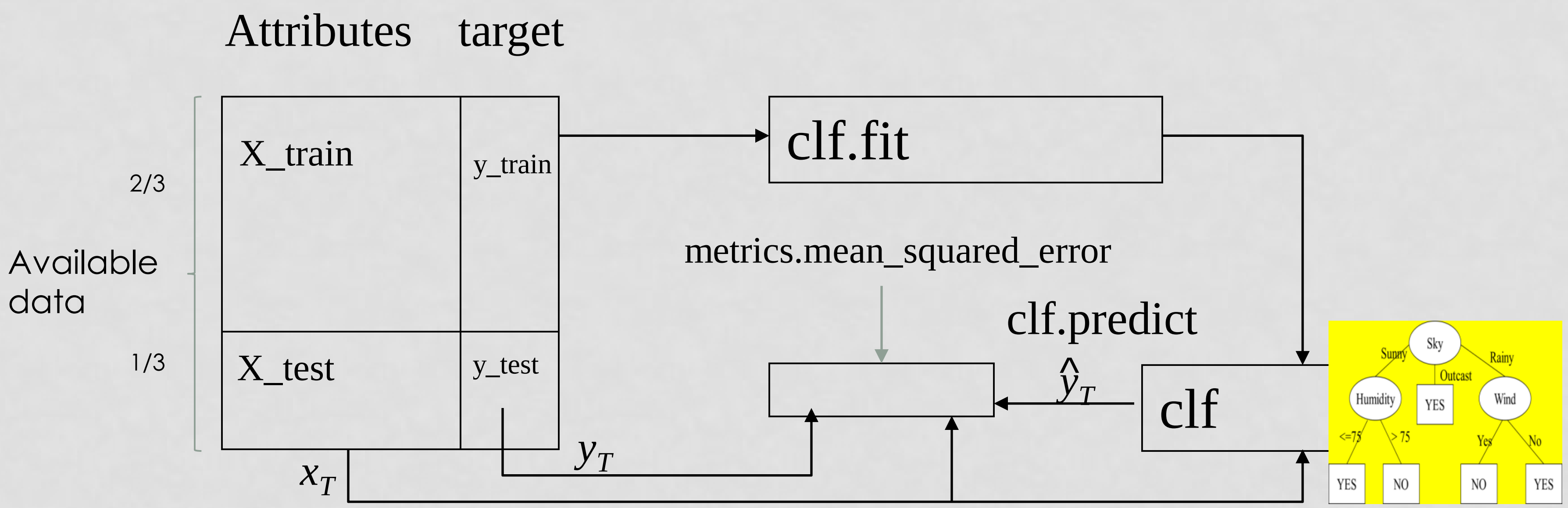
```
clf = tree.DecisionTreeRegressor()
clf = clf.fit(X_train, y_train)
```

And the model is used to make predictions for the training set, and more importantly, for the testing set. We can observe that the training error is very small (0 in this case) but the testing error is much larger.

```
import numpy as np
from sklearn import metrics
y_train_pred = clf.predict(X_train)
y_test_pred = clf.predict(X_test)
print(metrics.mean_squared_error(y_train, y_train_pred))
print(metrics.mean_squared_error(y_test, y_test_pred))
```

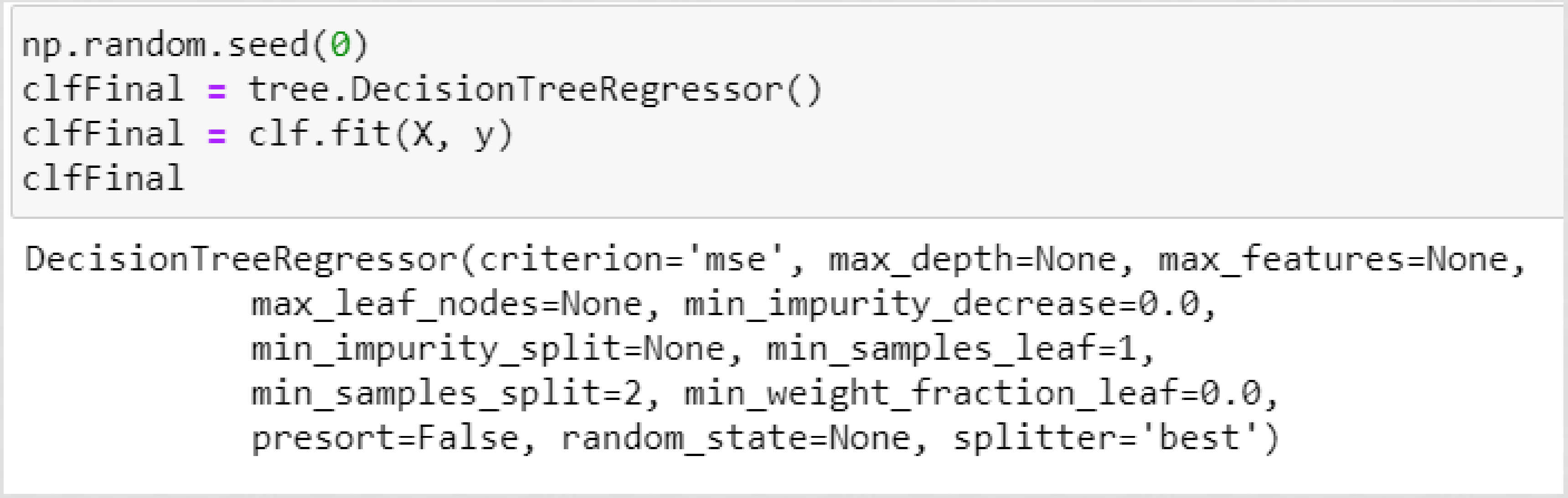
```
0.0
25.530718562874245
```

TRAIN / TEST EVALUATION METHOD (A.K.A. HOLDOUT METHOD)



TRAIN-TEST IN SCIKITLEARN

Training the final model with the complete dataset

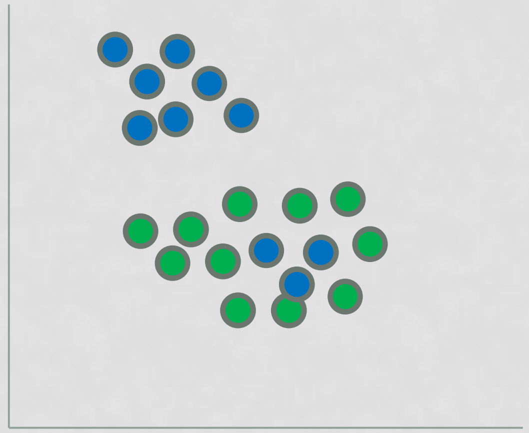


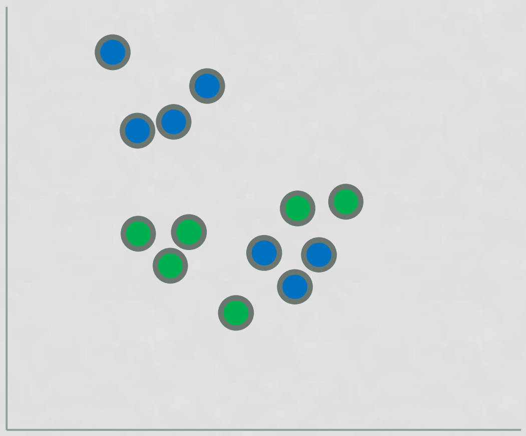
```
np.random.seed(0)
clfFinal = tree.DecisionTreeRegressor()
clfFinal = clf.fit(X, y)
clfFinal
```

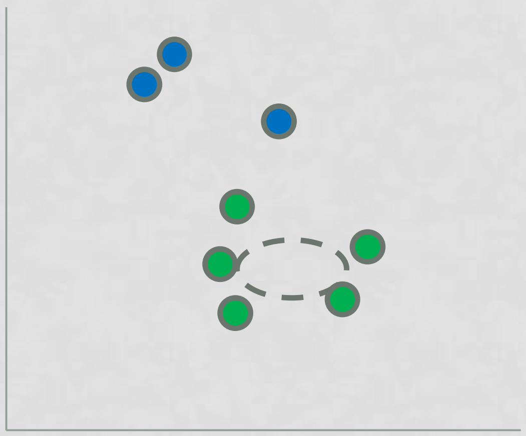
```
DecisionTreeRegressor(criterion='mse', max_depth=None, max_features=None,
           max_leaf_nodes=None, min_impurity_decrease=0.0,
           min_impurity_split=None, min_samples_leaf=1,
           min_samples_split=2, min_weight_fraction_leaf=0.0,
           presort=False, random_state=None, splitter='best')
```

train/test (holdout) drawback

- It is possible that the test partition does not represent well the problem (by chance), mainly **if dataset is small**.


Available data

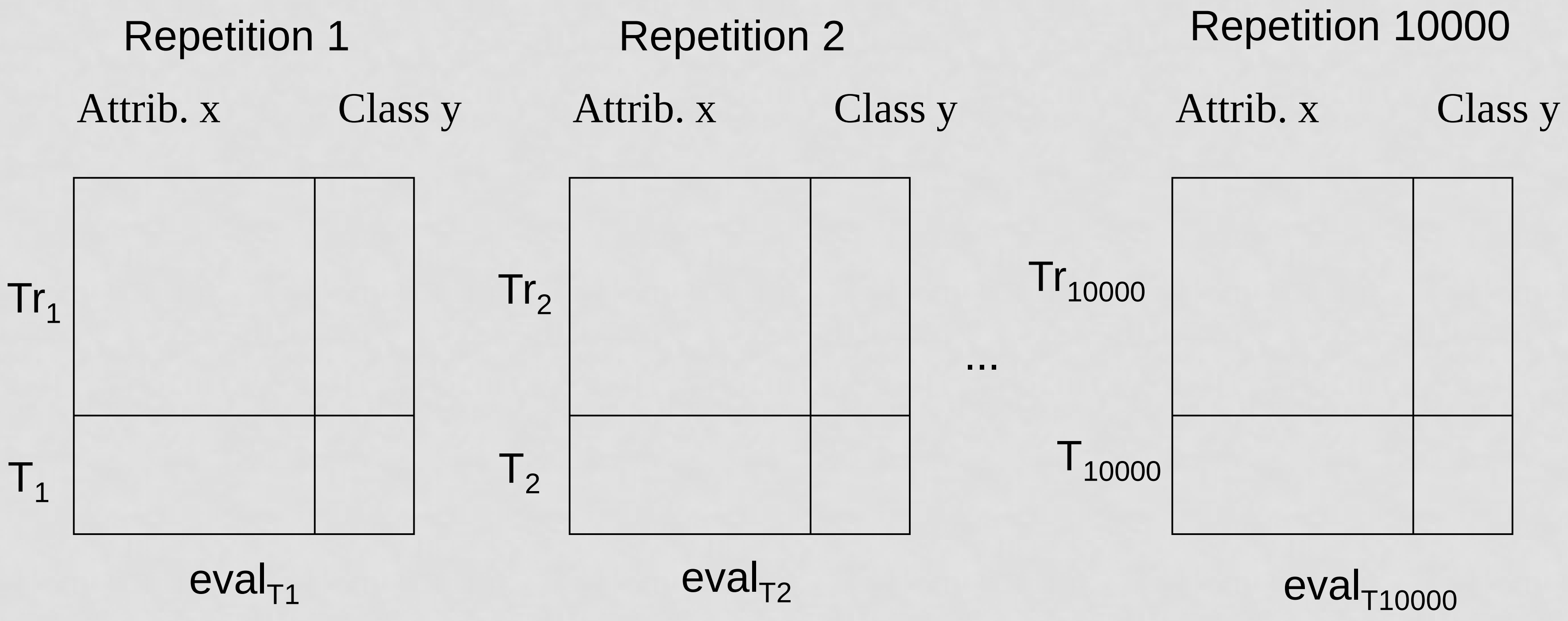

Train partition


Biased test partition

REPEATED TRAIN / TEST

- Repeat train / test many times
- Given that biases happen randomly, biases that appear in some of the train / test partition will not appear in other train / test partitions. In general random biases will cancel each other after averaging.
- **Method:**
  - **Repeat many times:**
    1. **Sort available data randomly (randomly shuffle dataset)**
    2. **Take the first 2/3 of instances for training and train the model**
    3. **Take the last 1/3 for testing and estimate model accuracy**
  - **Average all test success rates**

REPEATED TRAIN / TEST



Let's suppose we have very (very!) bad luck: $T_1 = T_2 = T_{10000}$
Does it make sense to compute the average $(eval_{T1} + eval_{T2} + ... + eval_{T10000})/10000$?

REPEATED TRAIN / TEST

- Main problem: the different test partitions might overlap: they are not independent
- In an extreme (and unlikely) case, if all test partitions were exactly the same, computing the average of all of them would be useless
- This extreme case never happens, but there is always some overlap between test partitions

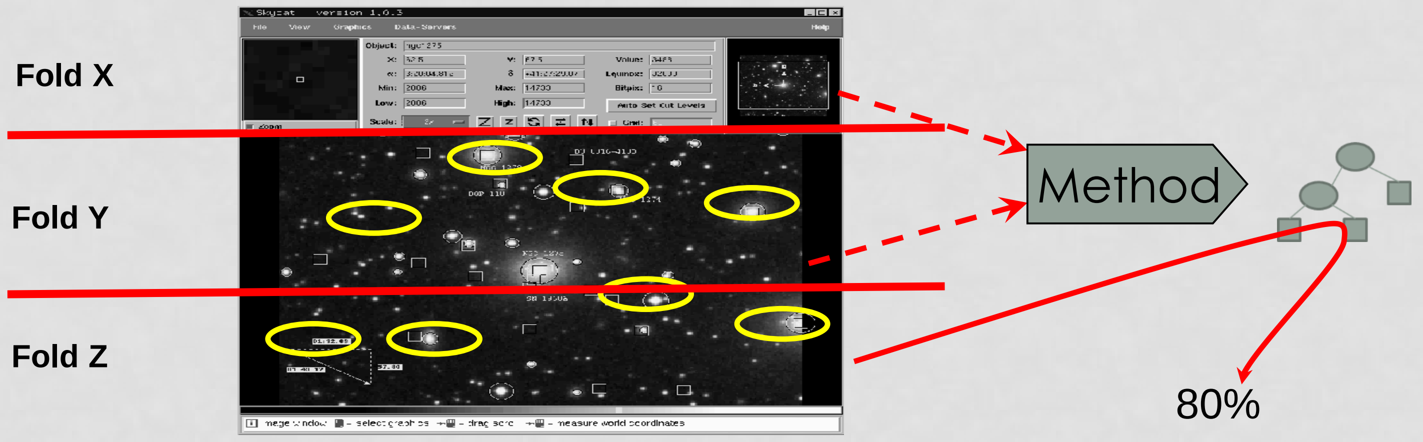
CROSSVALIDATION

- The available data is divided into k folds (k partitions).
- With k=3, three folds X, Y, and Z.
- The process has k steps (3 in this case):
  - Train model with X, Y, and test it with Z (T1 = success rate on Z)
  - Train model with X, Z, and test it with Y (T2 = success rate on Y)
  - Train model with Y, Z and test it with X (T3 = success rate on X)
  - Accuracy TX = (T1+T2+T3)/3
- k=10 is recommended. K between 5 and 10 can also be used.

3-fold cross-validation evaluation

Train with X and Y, evaluate with Z

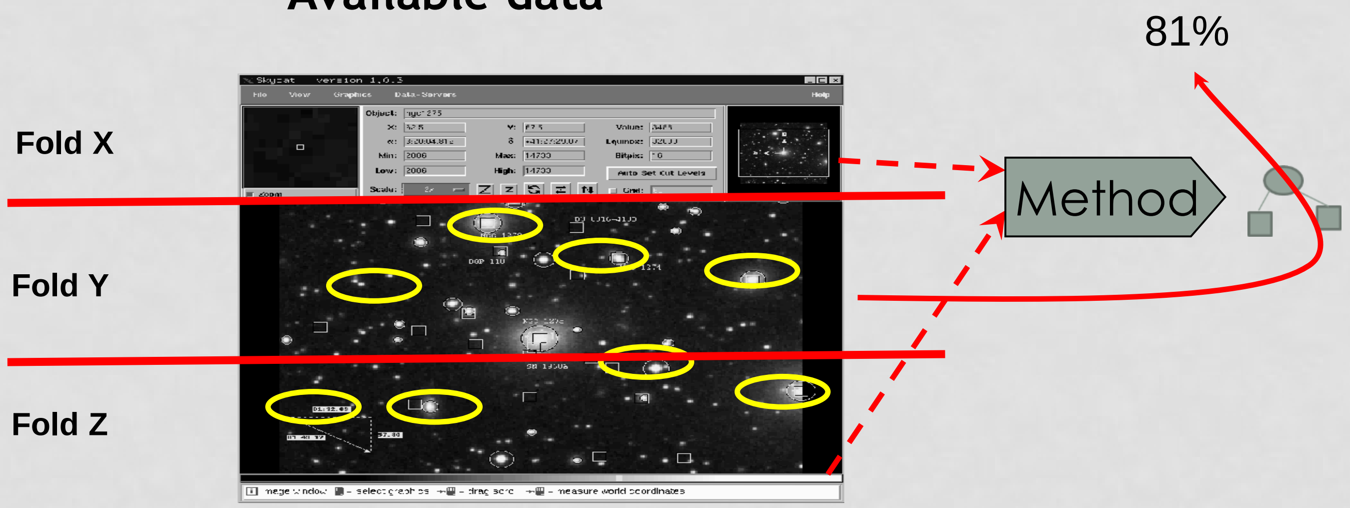
Available data



3-fold cross-validation evaluation

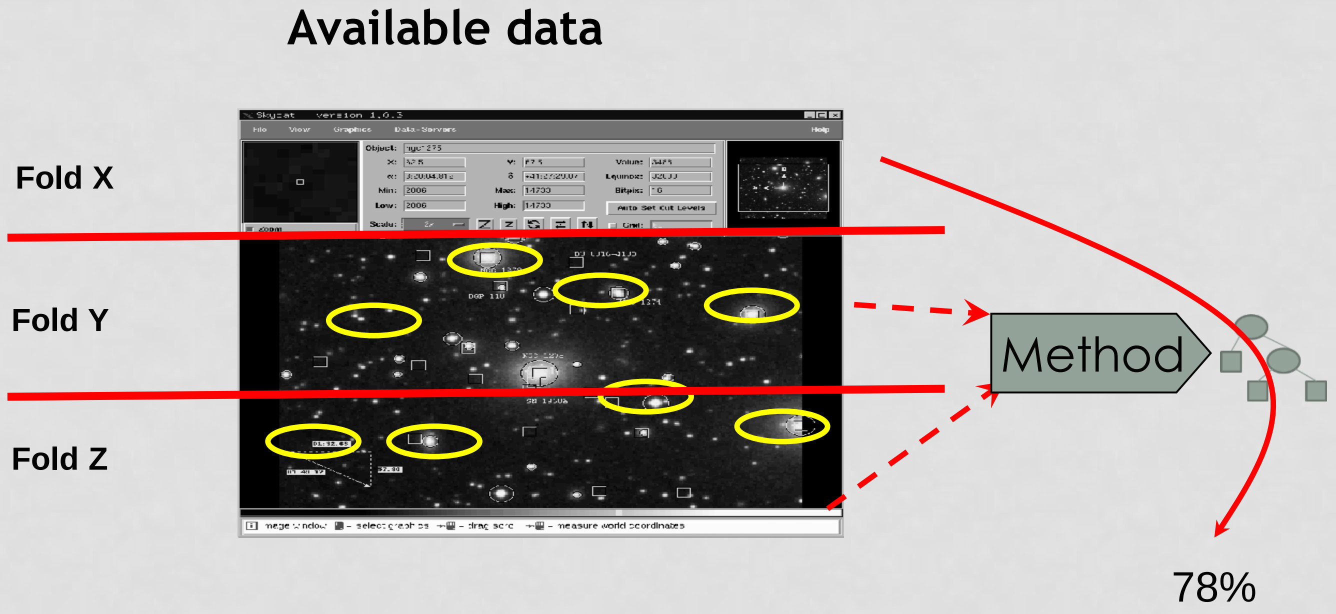
Train with X, Z; evaluate with Y

Available data



3-fold cross-validation evaluation

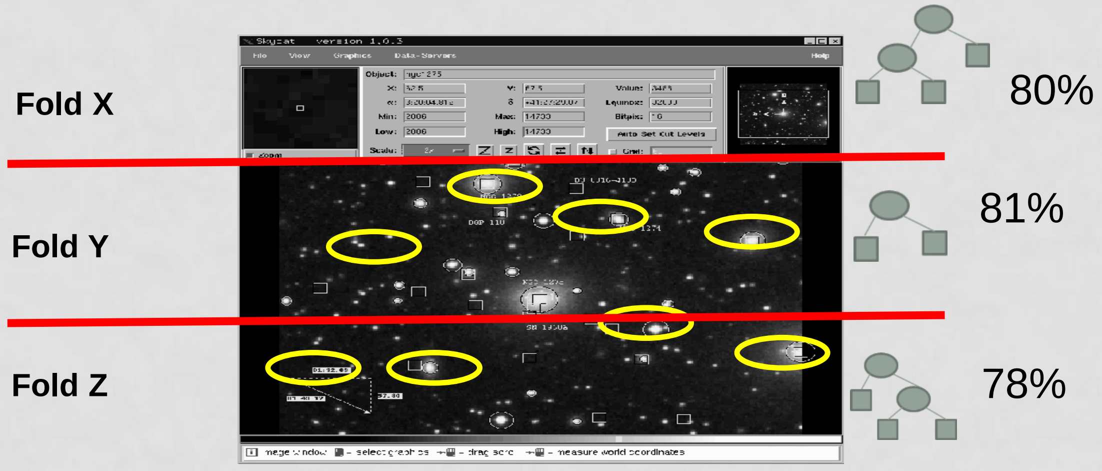
Train with Y, Z; evaluate with X



3-fold cross-validation evaluation

<u>The estimation of future performance T is the average of the three folds.</u>

Available data



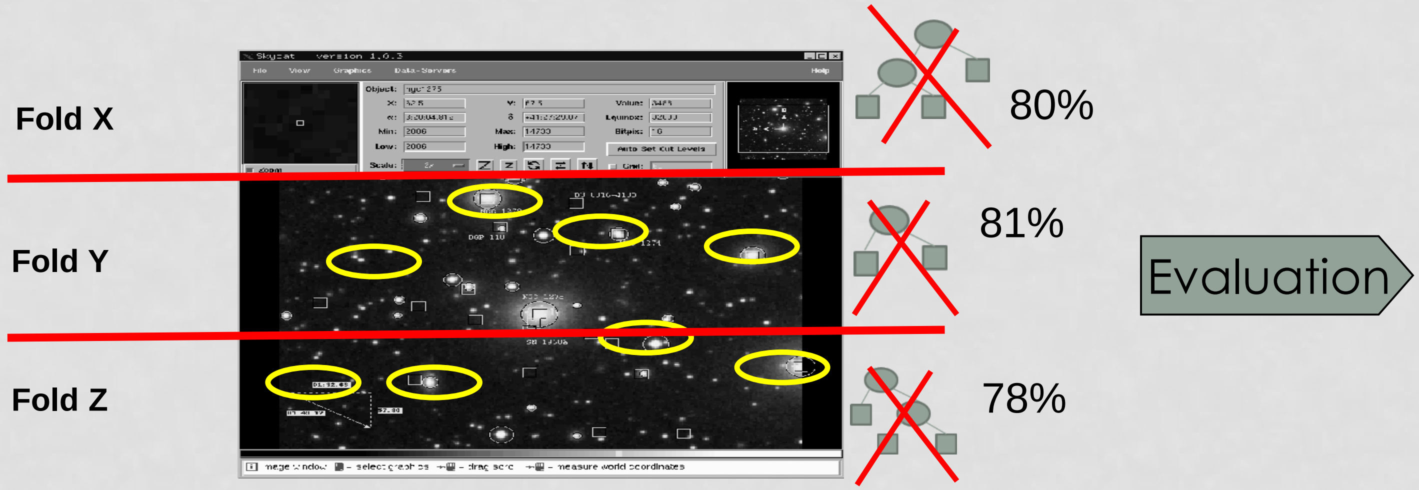
Evaluation

T=
(80%+81%+78%)/3
= 79.7%

3-fold cross-validation evaluation

Once T has been computed, the three models used to compute it are discarded and …

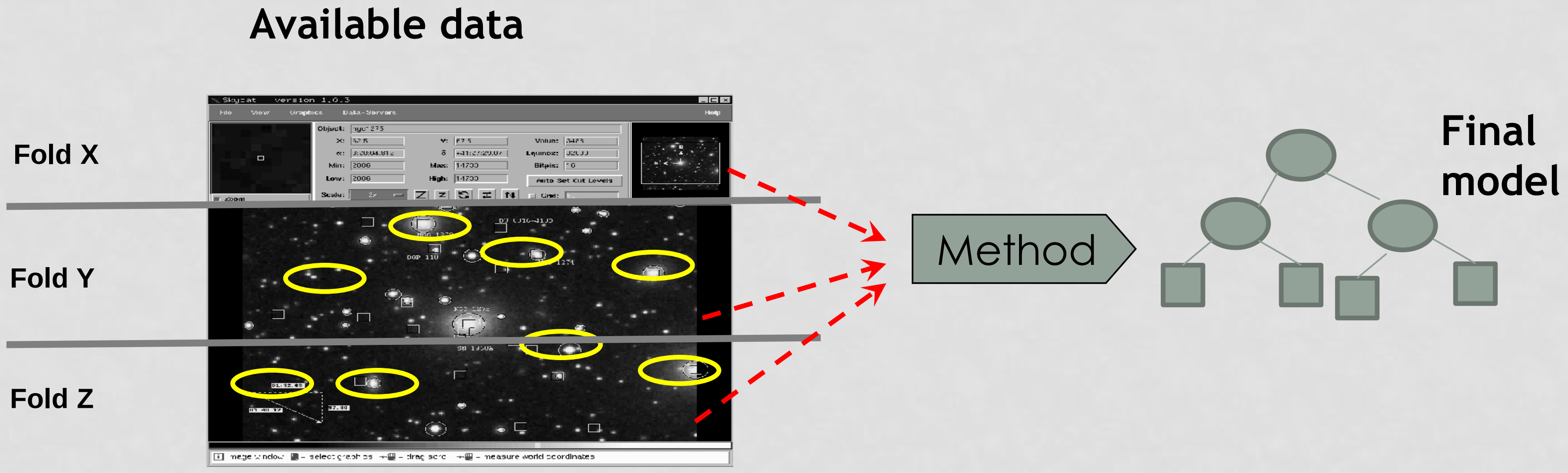
Available data



T= (80%+81%+78%)/3 = 79.7%

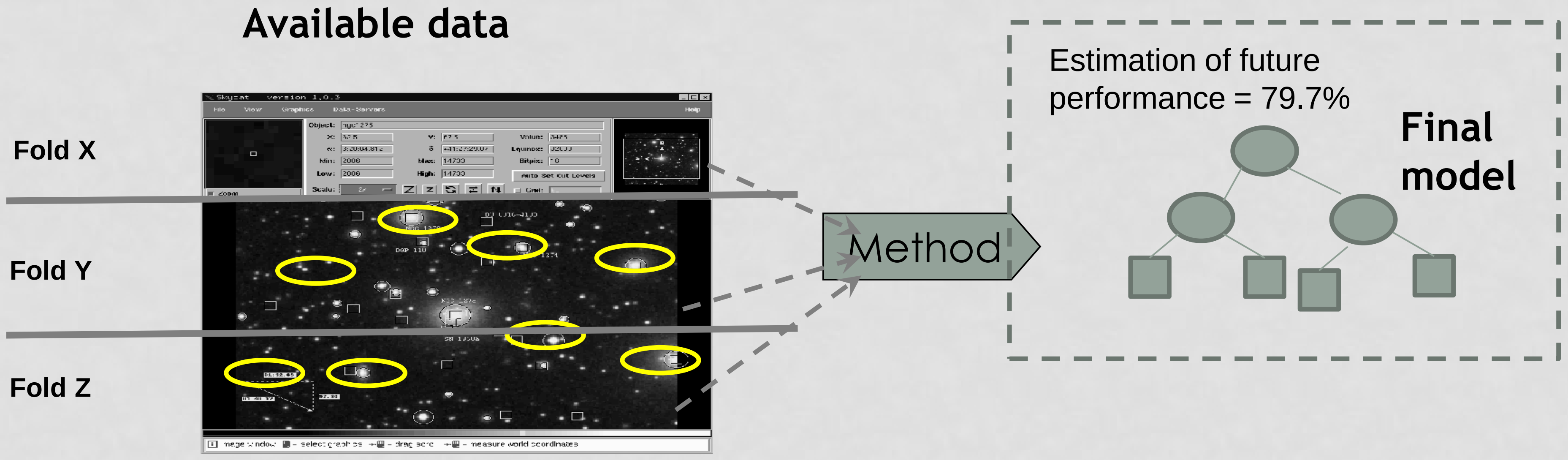
3-fold cross-validation evaluation

- <u>A final model is trained with the complete dataset</u>



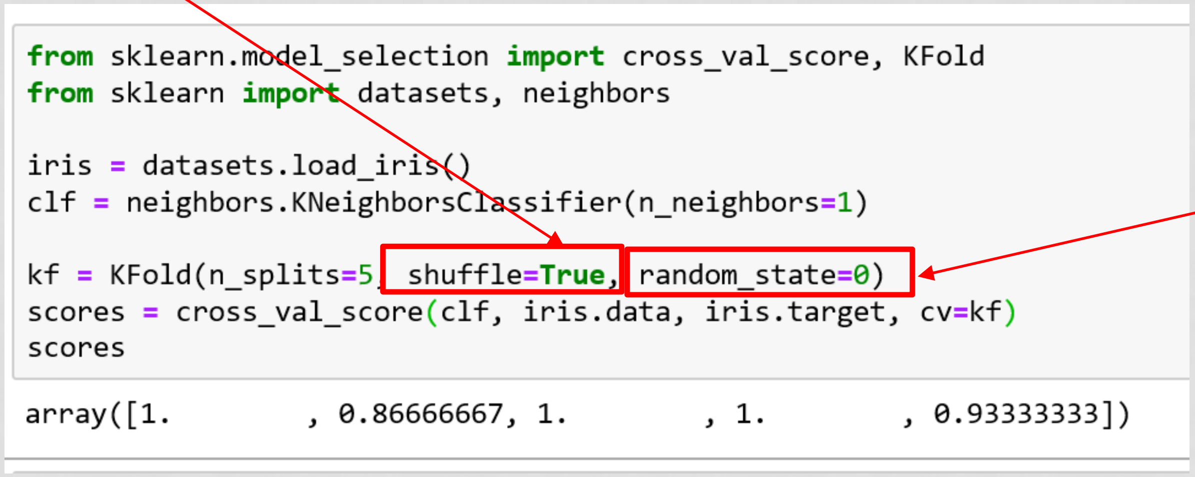
3-fold cross-validation evaluation

- A final model is trained with the complete dataset
- The estimation of future performance computed previously is kept (79.7%)
- Again, this is considered a **pesimistic estimation**, because the data partitions used to compute it were smaller (2/3) than the dataset used to train the final model.



CROSSVALIDATION IN SCIKIT-LEARN

- shuffle = True means, randomly reorder the available data.



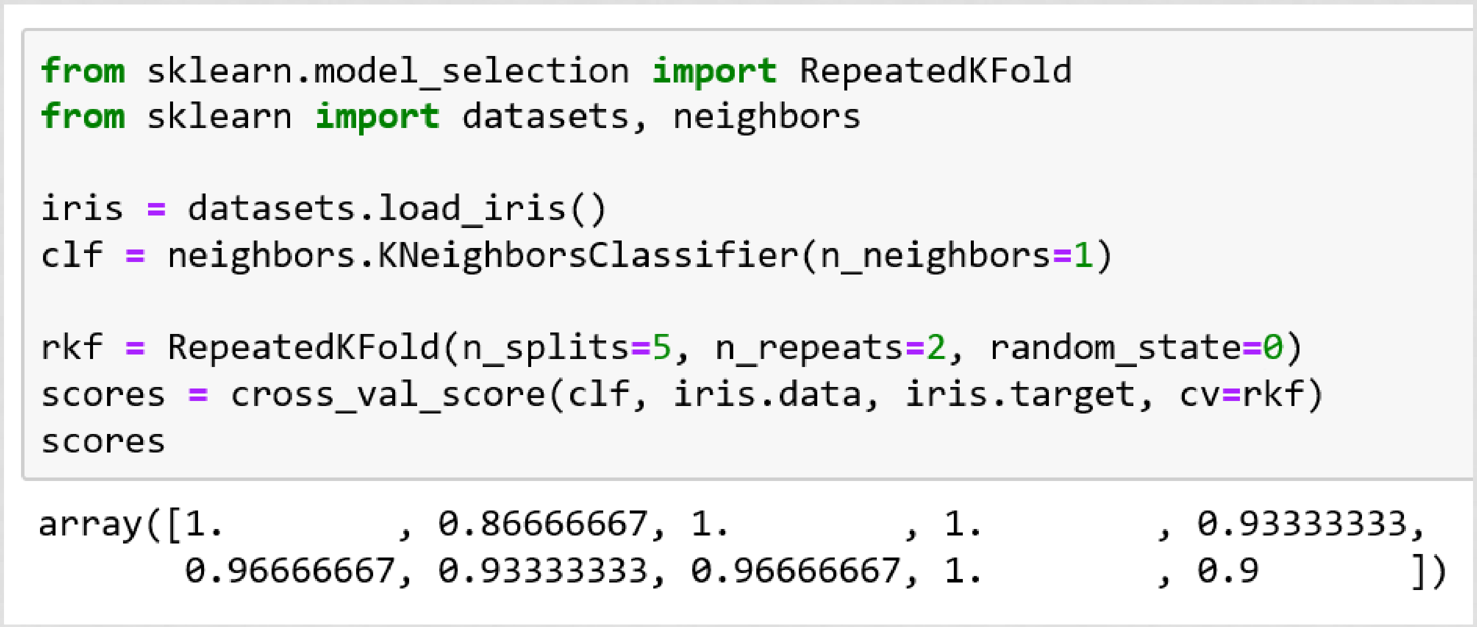
```
from sklearn.model_selection import cross_val_score, KFold
from sklearn import datasets, neighbors

iris = datasets.load_iris()
clf = neighbors.KNeighborsClassifier(n_neighbors=1)

kf = KFold(n_splits=5, shuffle=True, random_state=0)
scores = cross_val_score(clf, iris.data, iris.target, cv=kf)
scores
```

```
array([1.        , 0.86666667, 1.        , 1.        , 0.93333333])
```

For reproducibility of results

CROSSVALIDATION IN SCIKIT-LEARN

- Different random states will obtain different data shuffling which in turn will return different crossvalidation scores
- If stability of crossvalidation results is required, do repeated crossvalidation



```python
from sklearn.model_selection import RepeatedKFold
from sklearn import datasets, neighbors

iris = datasets.load_iris()
clf = neighbors.KNeighborsClassifier(n_neighbors=1)

rkf = RepeatedKFold(n_splits=5, n_repeats=2, random_state=0)
scores = cross_val_score(clf, iris.data, iris.target, cv=rkf)
scores
```

```
array([1.        , 0.86666667, 1.        , 1.        , 0.93333333,
       0.96666667, 0.93333333, 0.96666667, 1.        , 0.9       ])
```

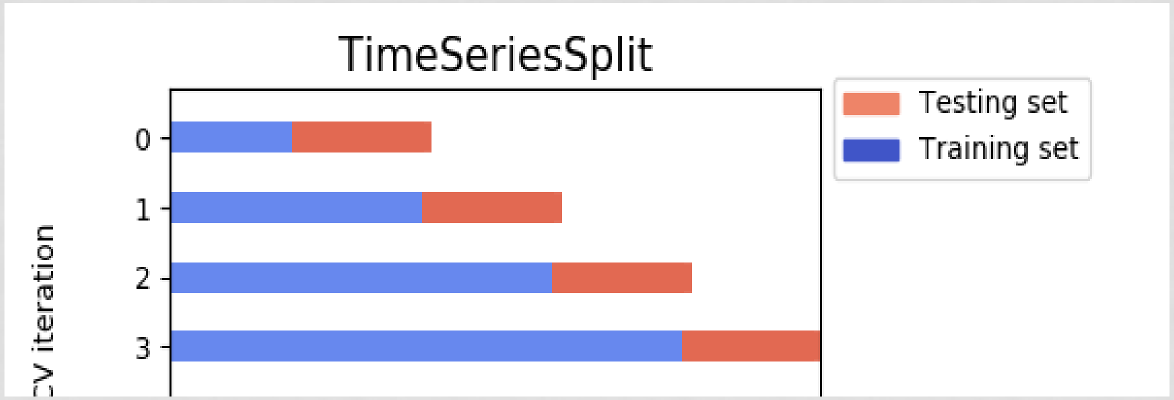
CROSSVALIDATION IN SCIKIT-LEARN

Training the final model with the complete dataset

```
X = iris.data
y = iris.target
clfFinal = neighbors.KNeighborsClassifier(n_neighbors=1)
clfFinal.fit(X,y)
```

OTHER FORMS OF CROSSVALIDATION

- Standard k-fold assumes instances are independent
- Group k-fold crossvalidation:
    - But sometimes, they may come in groups (e.g. instances (records) about patients)
    - Same patient instances should go to either train or test (but not both)
- Time series crossvalidation:
    - Instances near in time are auto-correlated
    - Typically, the past is chosen for training and the future for testing



BASIC CRITERIA FOR EVALUATING CLASSIFICATION MODELS

- The standard performance measure for classification is accuracy = average number of correctly classified instances
  - Or equivalently the missclassification error = 1-accuracy
- In order to know whether a model is "good enough", compare it with a trivial / naive model.
- E.g. with 2 classes, our model must be better than chance (tossing a head/tails coin): accuracy > 0.5
- For m classes, accuracy > 1/m
- However …

BASIC CRITERIA FOR EVALUATING CLASSIFICATION MODELS

- Imbalanced datasets contain much more data for one of the classes (the majority class) than the other. E.g. most people do not have cancer.
- Example of imbalanced dataset:
  - 990 negative instances (99%)
  - 10 positive instances (1%)
- What is the minimum success rate that our model should have in order to be considered useful?

BASIC CRITERIA FOR EVALUATING CLASSIFICATION MODELS

- Let be a problem with an imbalanced dataset:
  - 990 negative instances (99%)
  - 10 positive instances (1%)
- What is the minimum success rate that our model should have in order to be considered useful?

- A **trivial classifier** that always says “Negative” would already obtain 99% accuracy!!
- In imbalanced classification problems, a successful model has to obtain an accuracy larger than that of the majority class classifier.

EVALUATING IMBALANCED CLASSIFICATION PROBLEMS

- Accuracy is not very meaningful in imbalanced problems
- Other performance measures are used:
  - Confusion matrix: True Positive Rate, True Negative Rate, …
  - Balanced accuracy
  - Area under the ROC curve
  - F1
- Also, **stratified** partitions must be used (for both holdout and cross-validation)
  - X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42, stratify=y_train)
  - cv = StratifiedKFold(n_splits=5, random_state=42, shuffle=True)

PERFORMANCE MEASURES FOR REGRESSION

- Actual values of the response variable (ground truth): $\{y_1, \ldots, y_n\}$
- Model predictions: $\{p_1, \ldots, p_n\}$

MSE = Mean Squared Error
RMSE = Root-Mean Squared Error

$$MSE: \frac{(p_1 - y_1)^2 + \ldots + (p_n - y_n)^2}{n}; \quad RMSE = \sqrt{MSE}$$

- RMSE is very widely used
- But instances with large errors have too much weight on the average.

PERFORMANCE MEASURES FOR REGRESSION

MAE = Mean Absolute Error

$$MAE: \frac{|p_1 - y_1| + \ldots + |p_n - y_n|}{n}$$

- With MAE, instances with large errors do not have so much weight on the average (compared to RMSE).
- There is also the Median Absolute Error, which is more robust to outliers.

PERFORMANCE MEASURES FOR REGRESSION

MAE = Mean Absolute Error

$$MAE : \frac{|p_1 - y_1| + \ldots + |p_n - y_n|}{n}$$

- With MAE, instances with large errors do not have so much weight on the average (compared to RMSE).
- But both RMSE and MAE have the problem that their scale is relative to the scale of the output variable (y_i)
  - E.g.: if the response variable unit is meters, the RMSE and MAE will be 1000 larger than if the unit is km. That does not mean that the model is 1000 times worse. Just the scale is different.
- Therefore, in order to know whether the error of my model is “too large”, compare it with the error of a trivial / naive model.

PERFORMANCE MEASURES FOR REGRESSION

- An example of trivial model for regression: predict with a constant, disregarding the input attributes.
- What is the constant *c* that minimizes MSE?
- $MSE_c = \frac{1}{n}\sum_{i=1}^{i=n}(c - y_i)^2$
- $0 = \frac{dMSE}{dc} = \frac{1}{n}\sum_{i=1}^{i=n} 2(c - y_i) \quad = 2\left(c - \frac{1}{n}\sum_{i=1}^{i=n} y_i\right) = 2(c - \bar{y}) = 0$
  - $c = \bar{y}$
- Therefore, in the case of MSE, the MSE of the trivial model is:
- $\mathrm{MSE}_{\bar{y}} = \frac{1}{n}\sum_{i=1}^{i=n}(\bar{y} - y_i)^2$
- Which happens to be the variance of the response variable
- MSE_{model} should be smaller than $\mathrm{MSE}_{\bar{y}}$

PERFORMANCE MEASURES FOR REGRESSION

- R^2, also called the coefficient of determination, widely used to evaluate linear models, is defined in general as:

$$R^2 = 1 - \frac{\frac{1}{n}\sum_{i=1}^{i=n}(p_i - y_i)^2}{\frac{1}{n}\sum_{i=1}^{i=n}(\bar{y} - y_i)^2} = 1 - \frac{MSE_{model}}{Var(y\quad)} = 1 - \frac{MSE_{model}}{MSE_{\bar{y}}}$$

- It can be interpreted as a relative performance measure of the model versus the trivial mean-model.
- Upper bound of R^2 is obviously 1
- A trivial model that always predicts $\bar{y}$, disregarding the input features, would get $R^2 = 0$.
- Can R^2 be negative?

PERFORMANCE MEASURES FOR REGRESSION

- R^2, also called the coefficient of determination, widely used to evaluate linear models, is defined in general as:

$$R^2 = 1 - \frac{\frac{1}{n}\sum_{i=1}^{i=n}(p_i - y_i)^2}{\frac{1}{n}\sum_{i=1}^{i=n}(\bar{y} - y_i)^2} = 1 - \frac{MSE_{model}}{Var(y\quad)} = 1 - \frac{MSE_{model}}{MSE_{\bar{y}}}$$

- It can be interpreted as a relative performance measure of the model versus the mean.
- Upper bound of R^2 is obviously 1
- Can R^2 be negative?
  - Yes, our model can be even worse than predicting with the mean.
  - Note: we should expect R^2 to be bounded between zero and one only if a linear regression model is fit, and it is evaluated on the same data it is fitted on.
- Typically $0 \leq R^2 \leq 1$

PERFORMANCE MEASURES FOR REGRESSION

- An example of trivial model for regression: predict with a constant.
- What is the constant *c* that minimizes MAE?
- $MAE_c = \frac{1}{n}\sum_{i=1}^{i=n} |c - y_i| \quad = \frac{1}{n}\sum_{i=1}^{i=a} (c - y_i) \quad + \frac{1}{n}\sum_{i=1}^{i=b} (y_i - c)$

  $a+b=n$
- $0 = \frac{dMAE}{dc} = \frac{1}{n}\sum_{i=1}^{i=a}(+1) \quad + \frac{1}{n}\sum_{i=1}^{i=b} (-1) \quad = 0$ when a=b
  - $c = median(y)$
- Therefore, in the case of MAE, the MAE of the trivial model is:
- $MAE_{median(y)} = \frac{1}{n}\sum_{i=1}^{i=n} |median(y) - y_i|$
- MAE_{model} should be smaller than $MAE_{median(y)}$

PERFORMANCE MEASURES FOR REGRESSION

- An example of trivial model for regression: predict with a constant.
- What is the constant *c* that minimizes MAE?
- $MAE_c = \frac{1}{n}\sum_{i=1}^{i=n} |c - y_i| \quad = \frac{1}{n}\sum_{i=1}^{i=a} (c - y_i) \quad + \frac{1}{n}\sum_{i=1}^{i=b} (y_i - c)$

  $a+b=n$
- $0 = \frac{dMAE}{dc} = \frac{1}{n}\sum_{i=1}^{i=a}(+1) \quad + \frac{1}{n}\sum_{i=1}^{i=b} (-1) \quad = 0$ when a=b
  - $c = median(y)$
- Therefore, in the case of MAE, the MAE of the trivial model is:
- $MAE_{median(y)} = \frac{1}{n}\sum_{i=1}^{i=n} |median(y) - y_i|$
- MAE_{model} should be smaller than $MAE_{median(y)}$
- In other words, $\frac{\mathrm{MAE}_{model}}{MAE_{median(y)}} \leq 1$

TRIVIAL / NAIVE MODELS IN SCIKITLEARN: DUMMY ESTIMATORS

- DummyClassifier:
  - Stratified: generates random predictions by respecting the training set class distribution.
  - Most_frequent always predicts the most frequent class in the training set (the majority class).
  - uniform generates predictions uniformly at random.
  - constant always predicts a constant label that is provided by the user.

- DummyRegressor:
  - Mean: always predicts the mean of the training targets.
  - Median: always predicts the median of the training targets.
  - Quantile: always predicts a user provided quantile of the training targets.
  - Constant: always predicts a constant value that is provided by the user.

TRIVIAL / NAIVE MODELS IN SCIKITLEARN: DUMMY ESTIMATORS

- DummyClassifier:

```
from sklearn.dummy import DummyClassifier
clf = DummyClassifier(strategy='most_frequent', random_state=42)
clf.fit(X_train, y_train)
print(metrics.accuracy_score(y_test, clf.predict(X_test)))
```