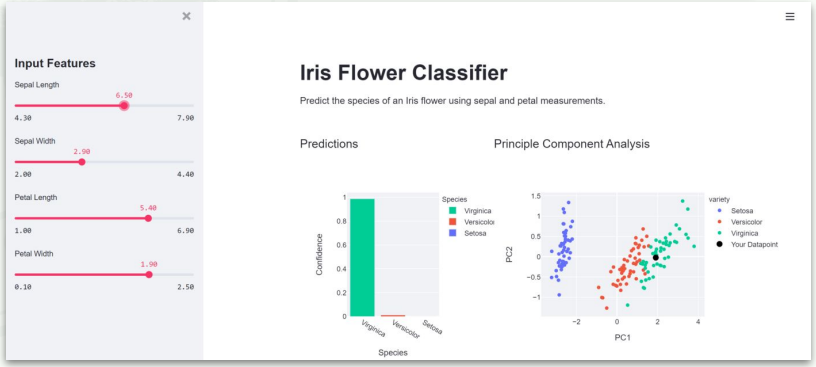
Machine Learning: Scikit-Learn, Classification and Clustering

Murillo
Computational Mathematics, Science and Engineering
Michigan State University

Final Projects

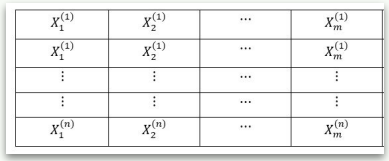
- same format as the midterm
  - make a streamlit app
  - presentations
- detailed instructions are in Slack
- must include machine learning in a very obvious way:
  - sklearn
  - more than one estimator
  - cross validation
  - score
  - hyperparameter tuning
  - feature engineering

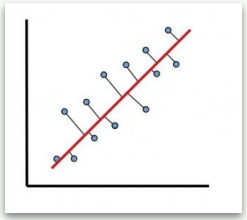


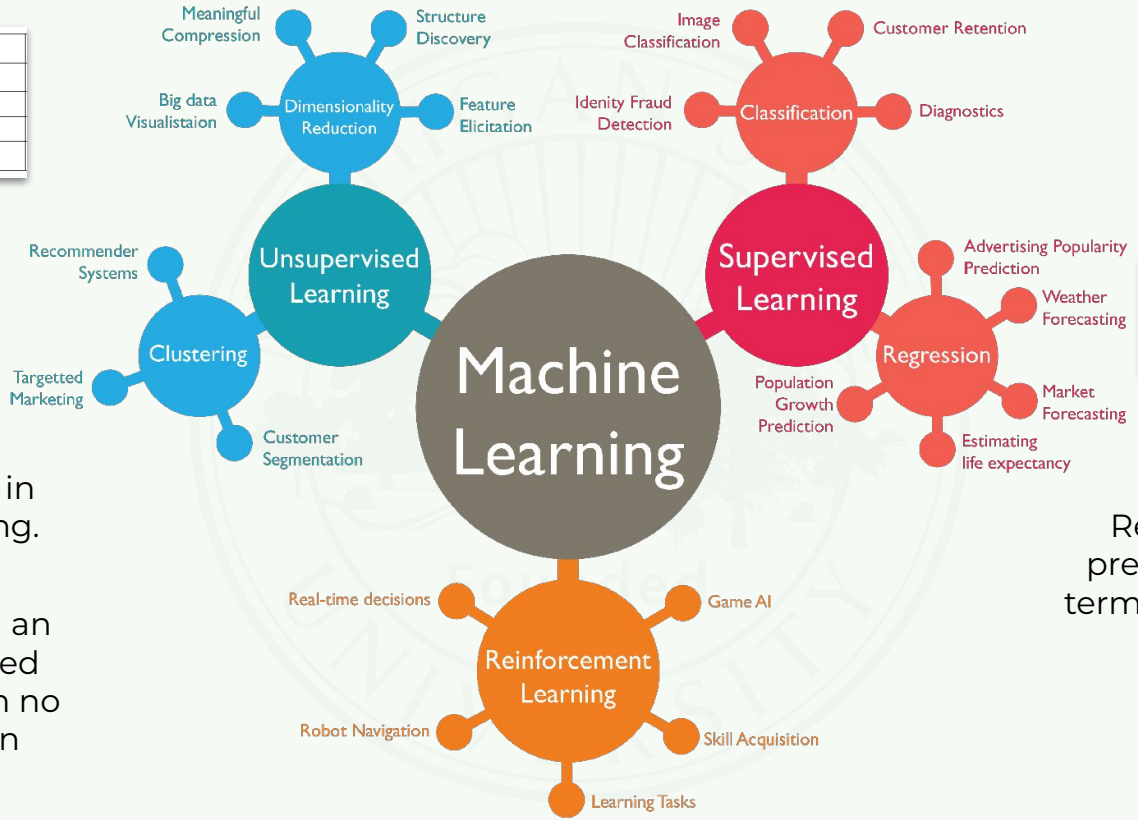
Schedule and Extra Credit

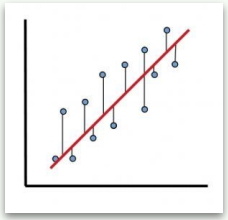
- This week (27th - 2nd)
  - mostly normal week
  - intro to ML with sklearn, for use in your project
- Next week (4th - 8th)
  - ICA on Monday
  - projects on Wednesday
- Extra Credit
  - no real HW next week: Extra Credit!
- Everything is due Friday at midnight on the 8th

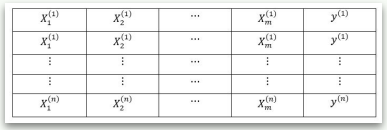
Organization of Machine Learning











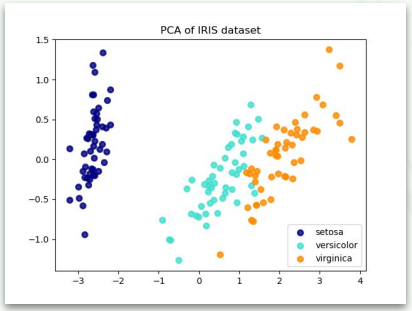
There is no known *y* in unsupervised learning.

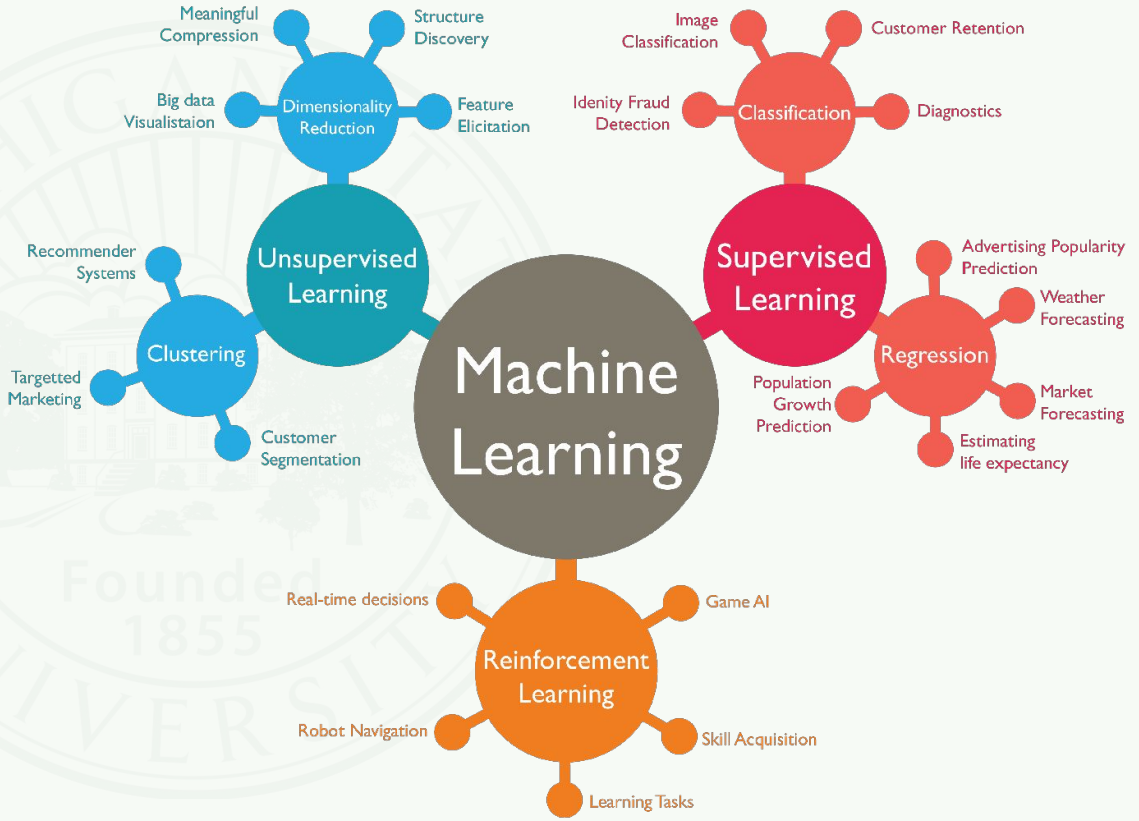
When we did SVD on an image, we simply used the data matrix *X* with no *y* to predict or use in finding weights.

Regression is used to predict a *y* (indep var) in terms of many *x* (dep vars).

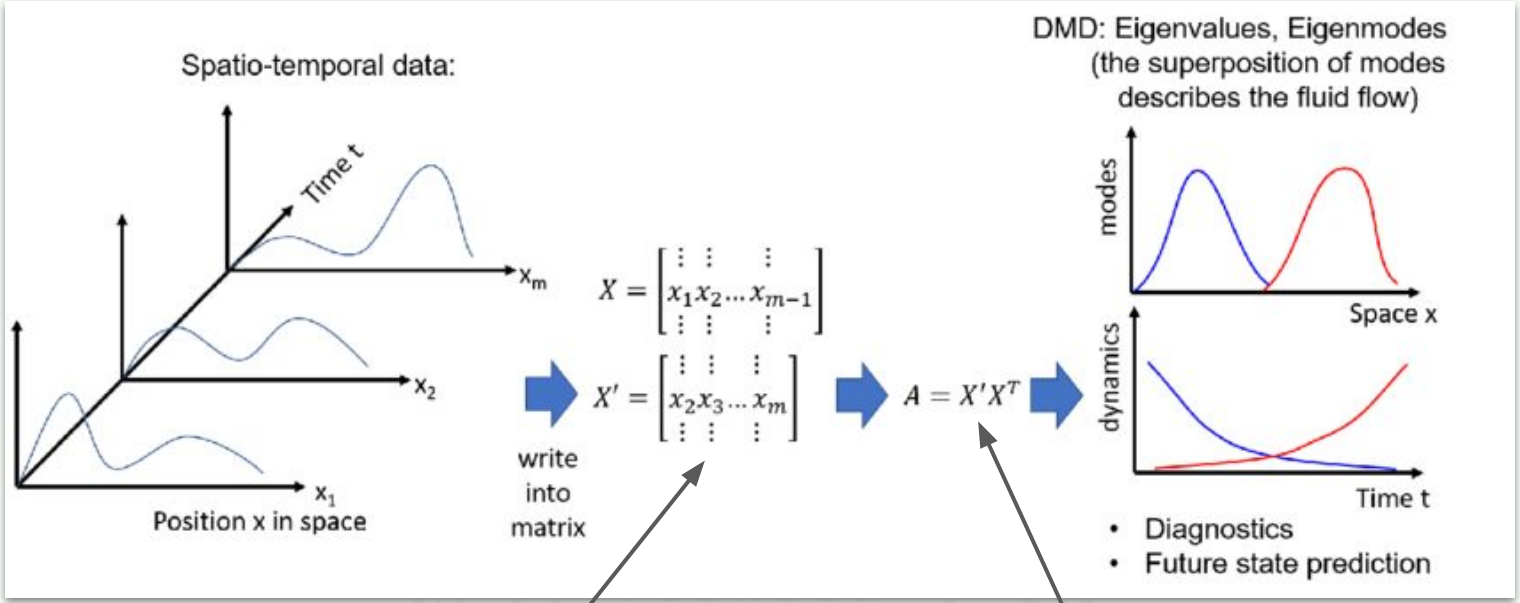
Dimensionality Reduction

dimensionality reduction: transform data to a lower-dimensional space that preserves important features of the data





Spatiotemporal Data



In this case, the values in each column are not different data samples, but values at specific spatial points. X^TX and XX^T have important physical meanings: averages over space and time.

For time dependent data we can form generalizations of the correlation matrix that includes changes over time. That correlation matrix contains interesting information about the dynamics.

Let's Compare the Two Covariance Matrices

time →

$$XX^T = \begin{bmatrix} | & | & \dots \\ n(t_1) & n(t_2) & \dots \\ | & | & \dots \end{bmatrix} \begin{bmatrix} - & n(t_1) & - \\ - & n(t_2) & - \\ & \vdots & \end{bmatrix}$$

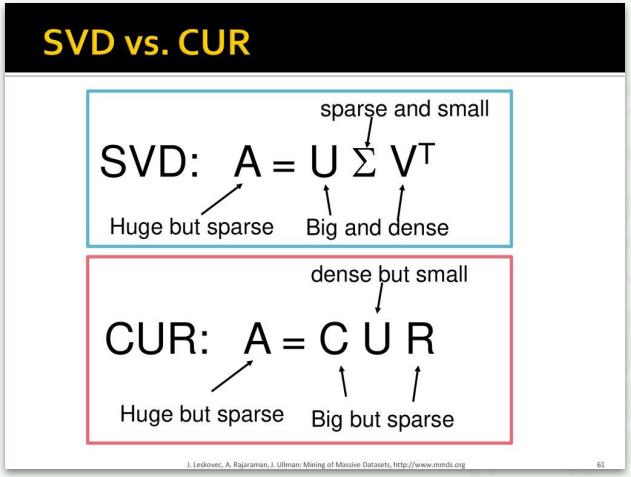
average over time

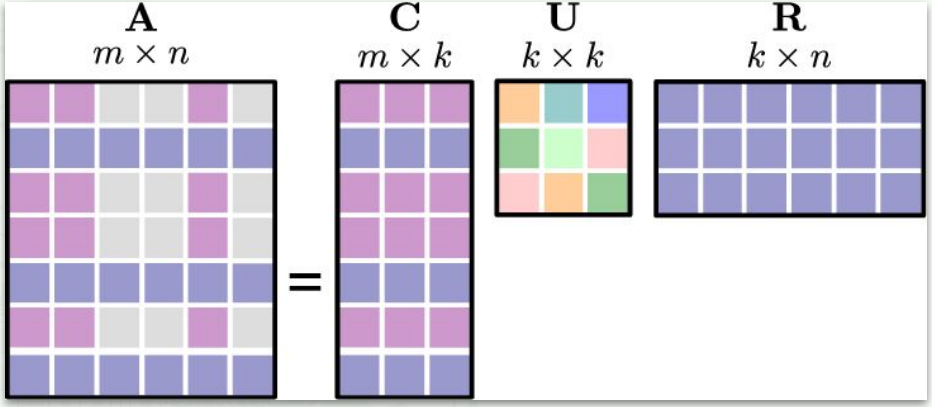
$$X^TX = \begin{bmatrix} - & n(t_1) & - \\ - & n(t_2) & - \\ & \vdots & \end{bmatrix} \begin{bmatrix} | & | & \dots \\ n(t_1) & n(t_2) & \dots \\ | & | & \dots \end{bmatrix}$$

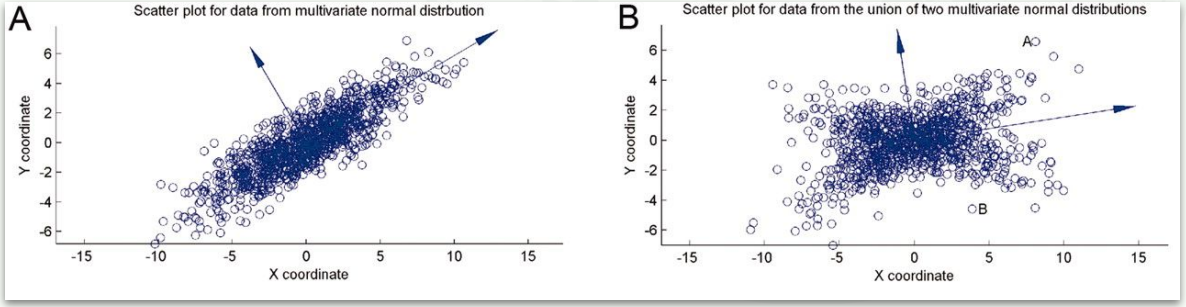
average over space

space →

SVD/PCA Are Common, But Not Always Best





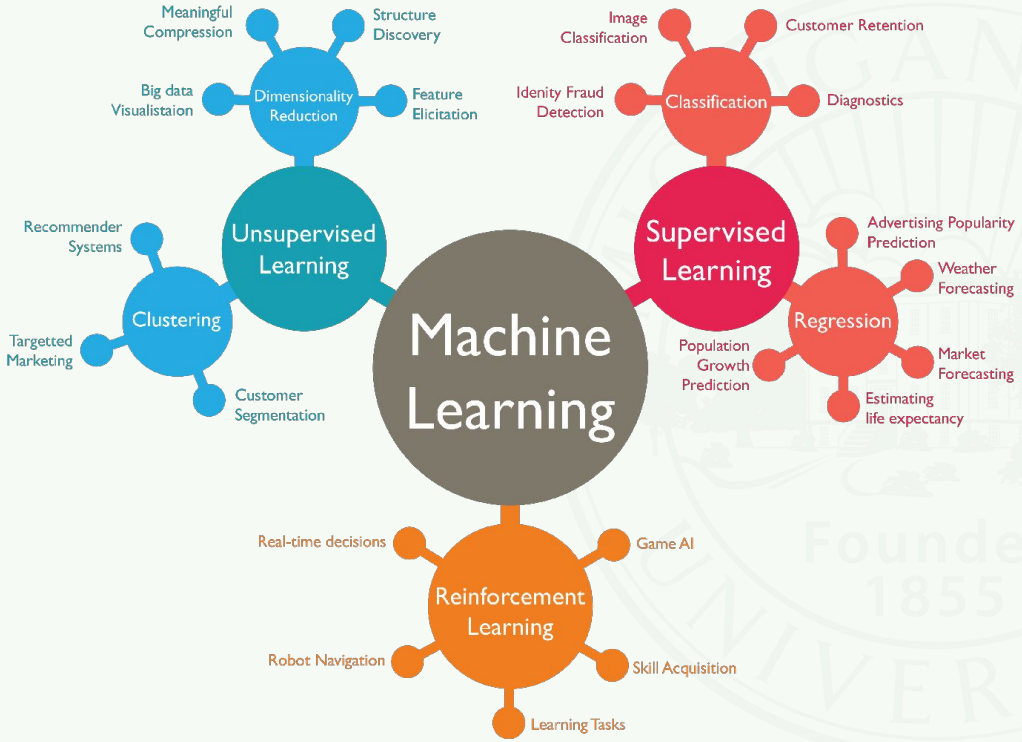


PCA is based on covariance, which may not best capture the structure in the data.

CUR Reference

This can be trivial to code, or fairly detailed. Look for Python libraries.

Learning

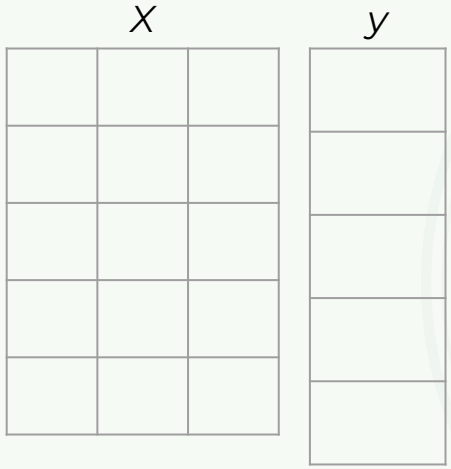


Machine Learning:

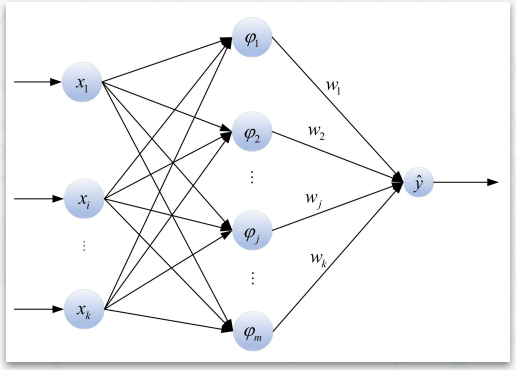
the use and development of computer systems that are able to learn and adapt without following explicit instructions, by using algorithms and statistical models to analyze and draw inferences from patterns in data

Supervised Learning

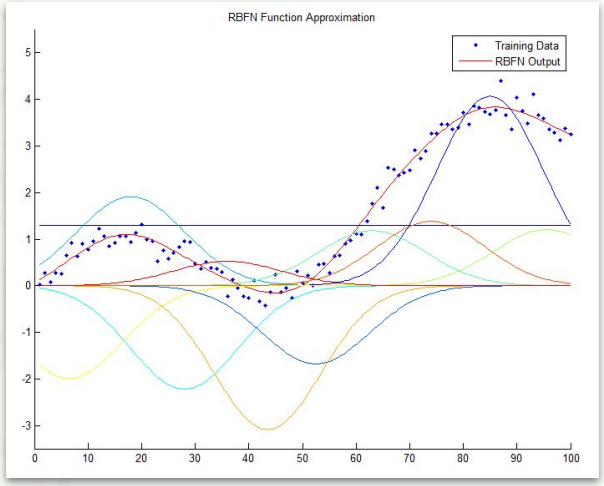
data → model → prediction



- preprocessing:
  - IDA, EDA
  - missing values
  - scaiing



- fits:
  - lines/hyperplanes
  - nonlinear functions
- methods:
  - linear algebra
  - optimization

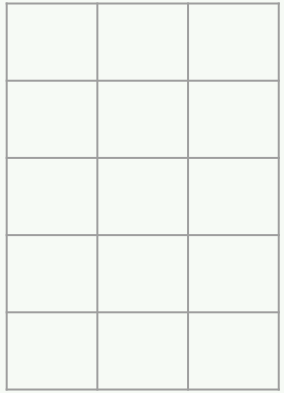


- how well did we do!?
- define a score using test data

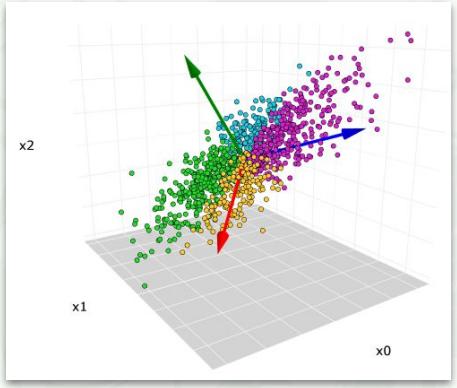
Unsupervised Learning

data → model → prediction?

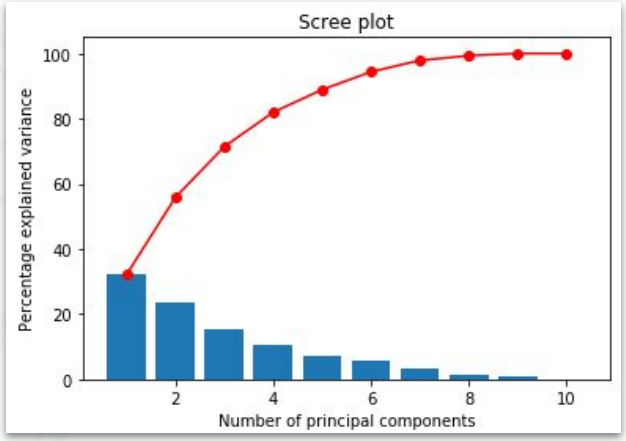
X



- preprocessing:
  - IDA, EDA
  - missing values
  - scaiing



- methods:
  - linear algebra
  - optimization

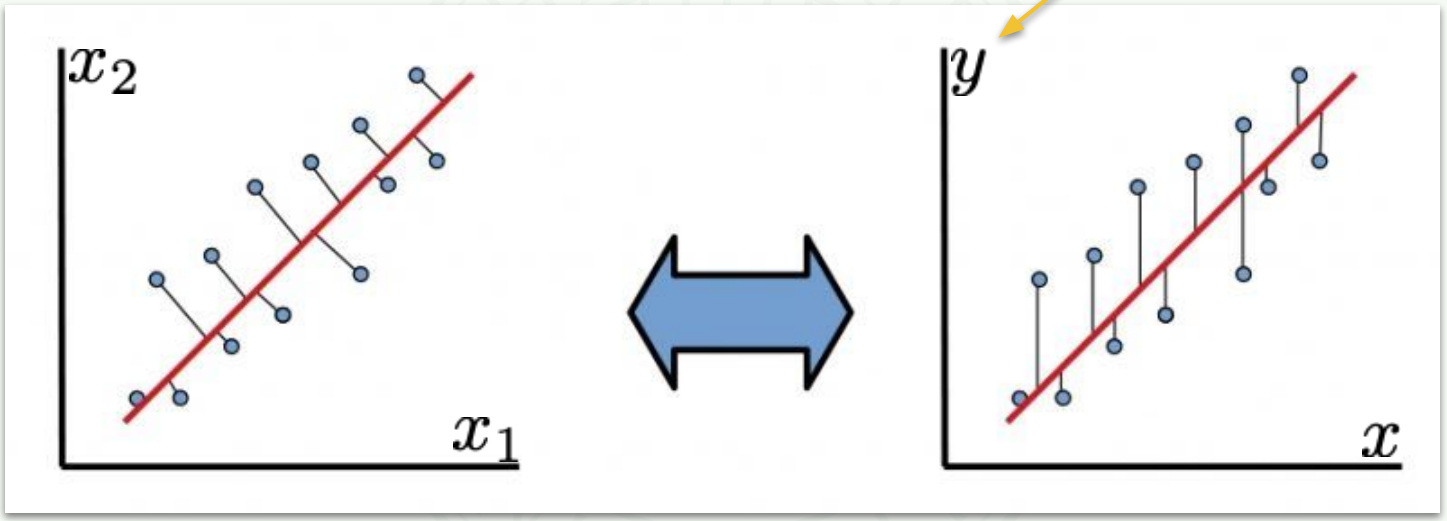


- how well did we do!?

Unsupervised and Supervised Compared

we have no labels/targets; all variables equivalent

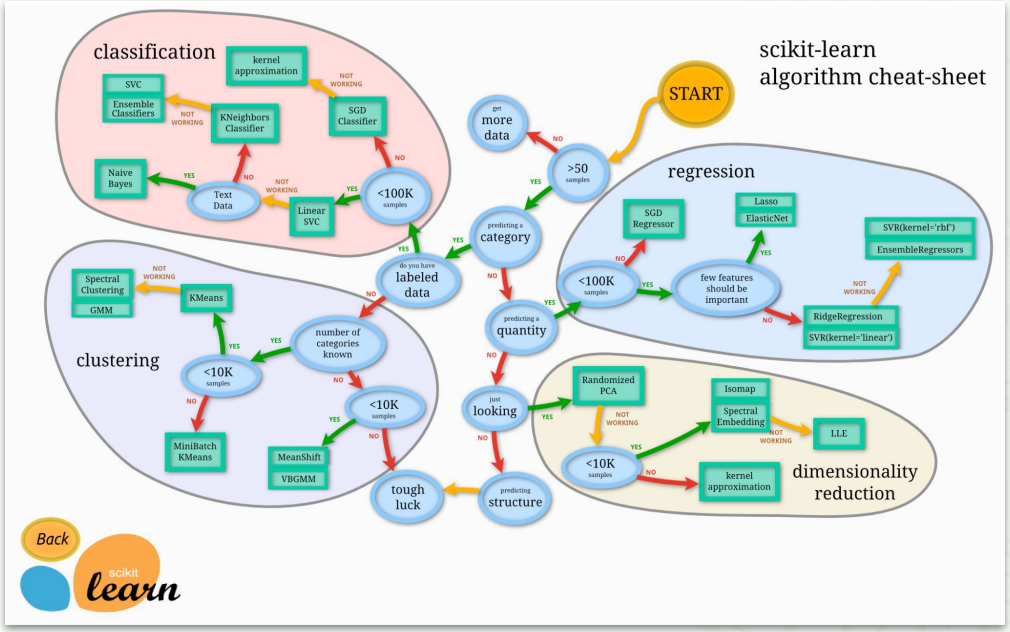
dependent variable



find structure in the data

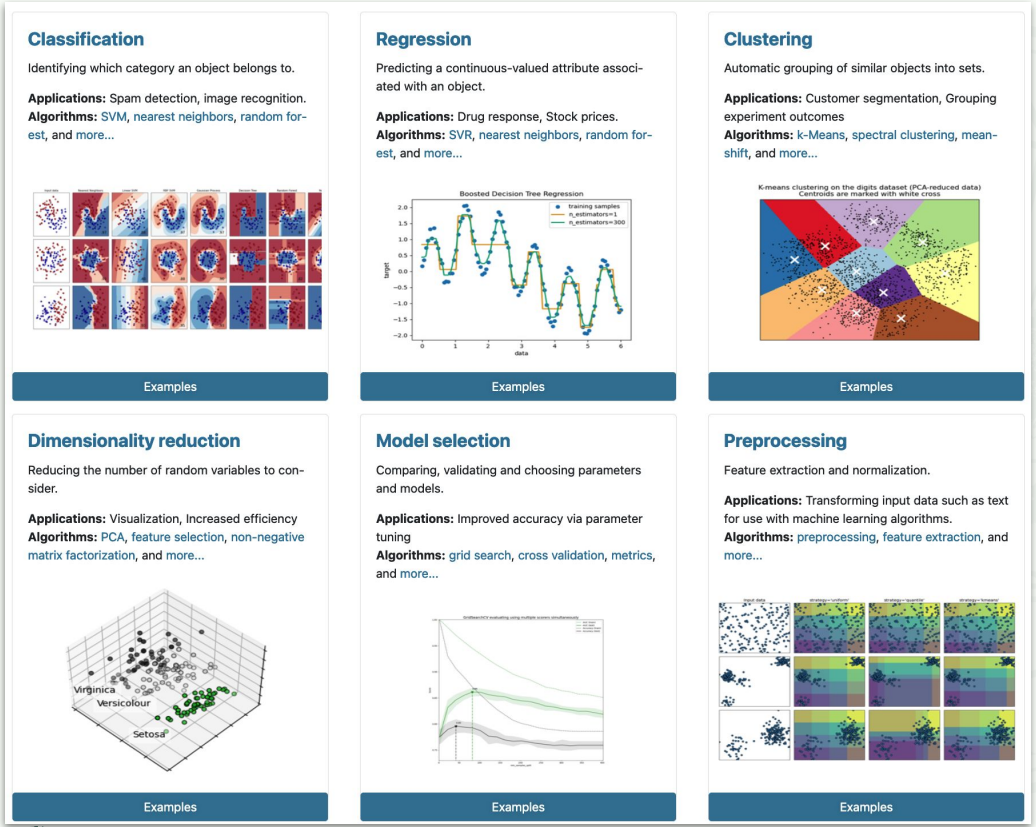
make predictions

Scikit-Learn



Scikit-Learn is the most widely used machine learning library in Python (so, probably in *anything*).

Scikit-Learn Organization



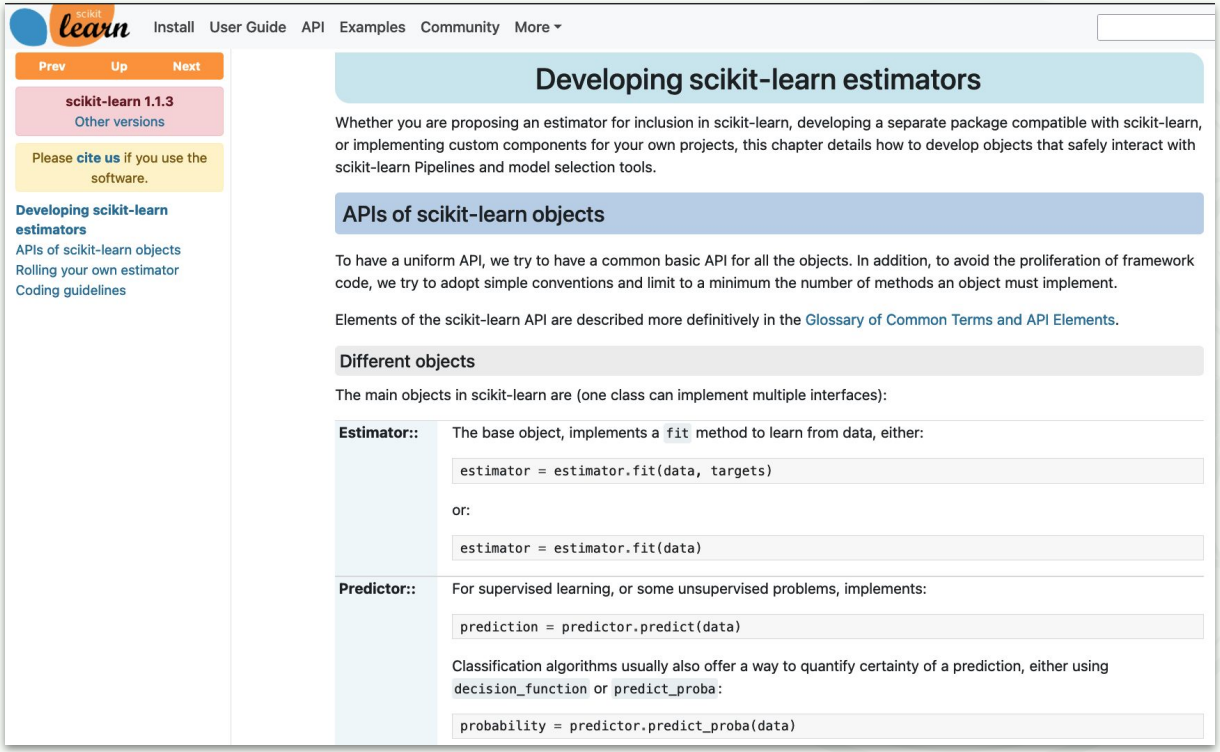
If you use some other ML library (PyTorch, TF, ...), you can combine that with sklearn:

- compare ML models in both libraries
- using the many preprocessing tools in sklearn as part of the DS steps

6.3. Preprocessing data

6.3.1. Standardization, or mean removal and variance scaling
6.3.2. Non-linear transformation
6.3.3. Normalization
6.3.4. Encoding categorical features
6.3.5. Discretization
6.3.6. Imputation of missing values
6.3.7. Generating polynomial features
6.3.8. Custom transformers

Estimators

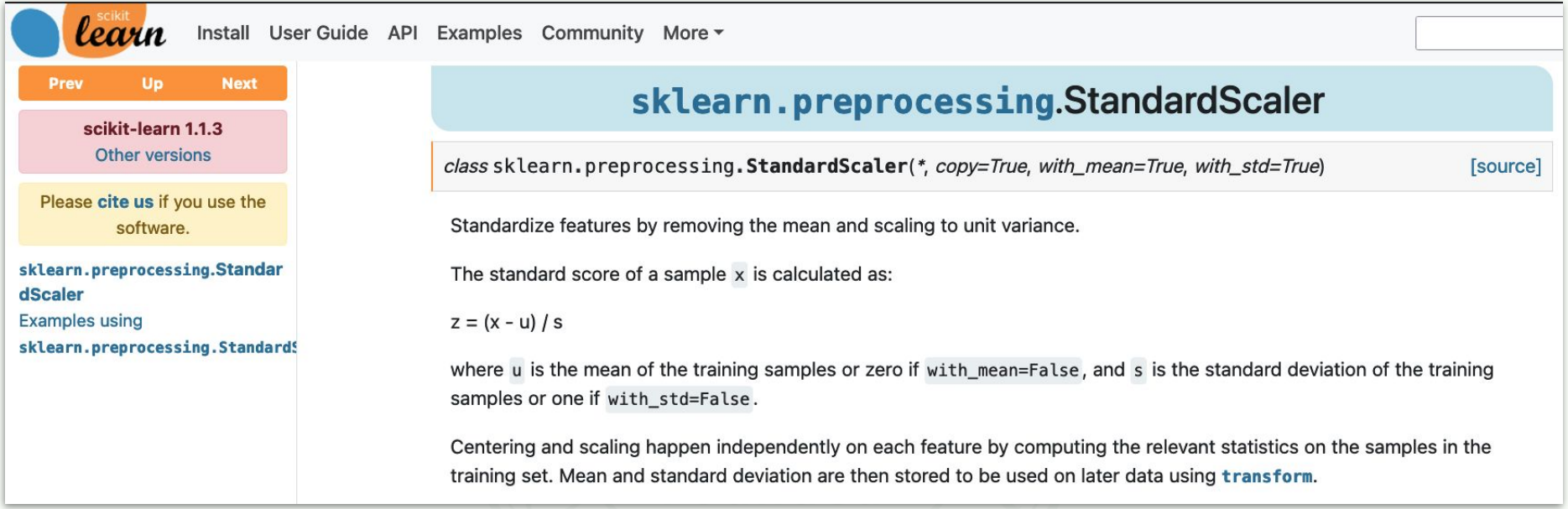


Much of the power of sklearn is its abstract workflow.

The APIs do not care which ML model you wish to use. These “black-box” models are called “estimators” in sklearn.

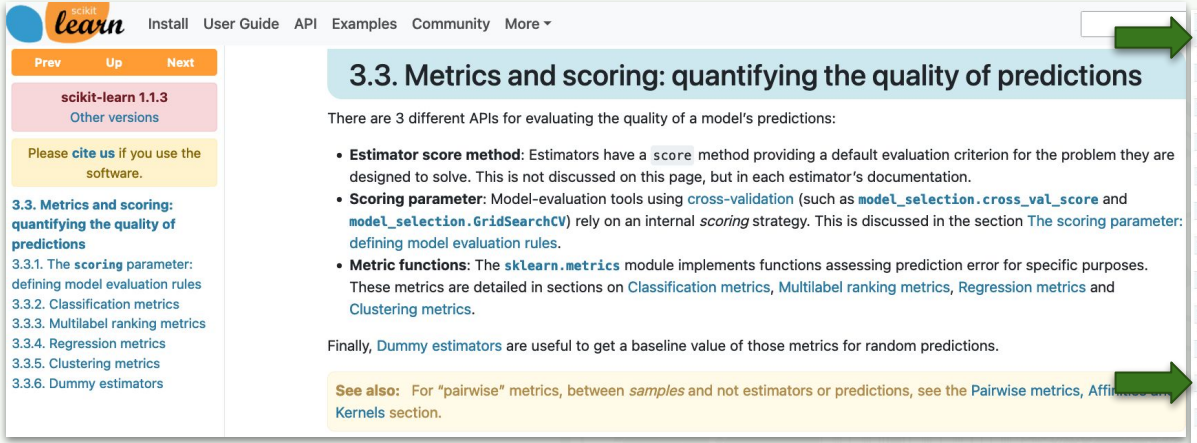
You can write your own, as long as you stick with the API.

Scaling



Of course, you can do most of this yourself with your own code. But, if you stay within sklearn, you can exploit the object oriented nature of its API.

Metrics/Scores



Scoring	Function	Comment
Classification		
'accuracy'	metrics.accuracy_score	
'balanced_accuracy'	metrics.balanced_accuracy_score	
'top_k_accuracy'	metrics.top_k_accuracy_score	
'average_precision'	metrics.average_precision_score	
'neg_brier_score'	metrics.brier_score_loss	
'f1'	metrics.f1_score	for binary targets
'f1_micro'	metrics.f1_score	micro-averaged
'f1_macro'	metrics.f1_score	macro-averaged
'f1_weighted'	metrics.f1_score	weighted average
'f1_samples'	metrics.f1_score	by multilabel sample
'neg_log_loss'	metrics.log_loss	requires predict_proba support
'precision' etc.	metrics.precision_score	suffixes apply as with 'f1'
'recall' etc.	metrics.recall_score	suffixes apply as with 'f1'
'jaccard' etc.	metrics.jaccard_score	suffixes apply as with 'f1'
'roc_auc'	metrics.roc_auc_score	
'roc_auc_ovr'	metrics.roc_auc_score	
'roc_auc_ovo'	metrics.roc_auc_score	
'roc_auc_ovr_weighted'	metrics.roc_auc_score	
'roc_auc_ovo_weighted'	metrics.roc_auc_score	
Clustering		
'adjusted_mutual_info_score'	metrics.adjusted_mutual_info_score	
'adjusted_rand_score'	metrics.adjusted_rand_score	
'completeness_score'	metrics.completeness_score	
'fowlkes_mallows_score'	metrics.fowlkes_mallows_score	
'homogeneity_score'	metrics.homogeneity_score	
'mutual_info_score'	metrics.mutual_info_score	
'normalized_mutual_info_score'	metrics.normalized_mutual_info_score	
'rand_score'	metrics.rand_score	
'v_measure_score'	metrics.v_measure_score	
Regression		
'explained_variance'	metrics.explained_variance_score	
'max_error'	metrics.max_error	
'neg_mean_absolute_error'	metrics.mean_absolute_error	
'neg_mean_squared_error'	metrics.mean_squared_error	
'neg_root_mean_squared_error'	metrics.mean_squared_error	
'neg_mean_squared_log_error'	metrics.mean_squared_log_error	
'neg_median_absolute_error'	metrics.median_absolute_error	
'r2'	metrics.r2_score	
'neg_mean_poisson_deviance'	metrics.mean_poisson_deviance	
'neg_mean_gamma_deviance'	metrics.mean_gamma_deviance	
'neg_mean_absolute_percentage_error'	metrics.mean_absolute_percentage_error	
'd2_absolute_error_score'	metrics.d2_absolute_error_score	
'd2_pinball_score'	metrics.d2_pinball_score	
'd2_tweedie_score'	metrics.d2_tweedie_score	

Metrics/scores are extremely important and their use is a large field of study.

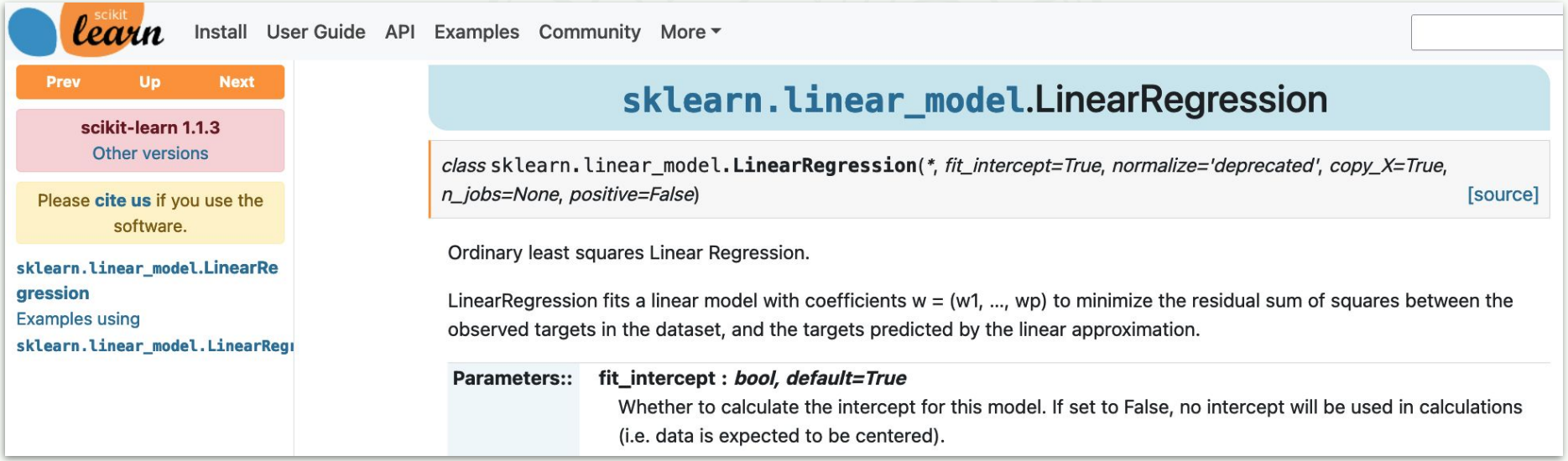
My advice: be aware of the complexity and be ready to adapt to new metrics as your problems change.

We will start this week with some simple metrics and scores.

Linear Regression in Sklearn

You have learned how to do regression “by hand”, using linear algebra. Great!

In practice, you would likely use a highly-optimized library.



Split the Data to get a Score

You will almost always need a **score**.

To get a score, you need test data, which means you will be splitting your data.

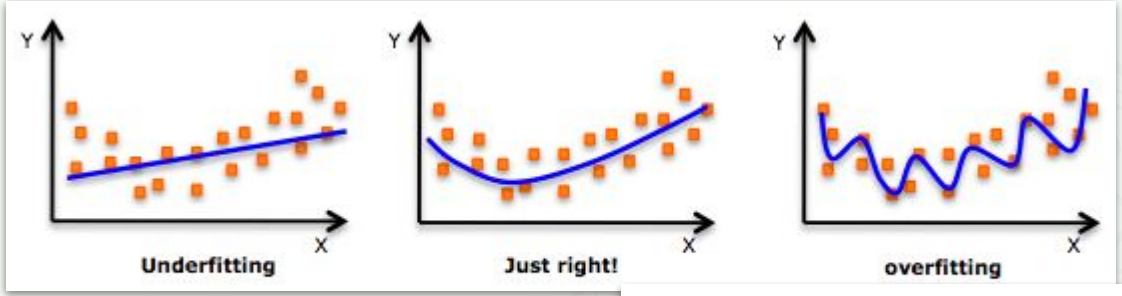
There are options:

- 50/50 split
- K-Fold CV (cross validation)
- and so on....your problem may suggest something unique

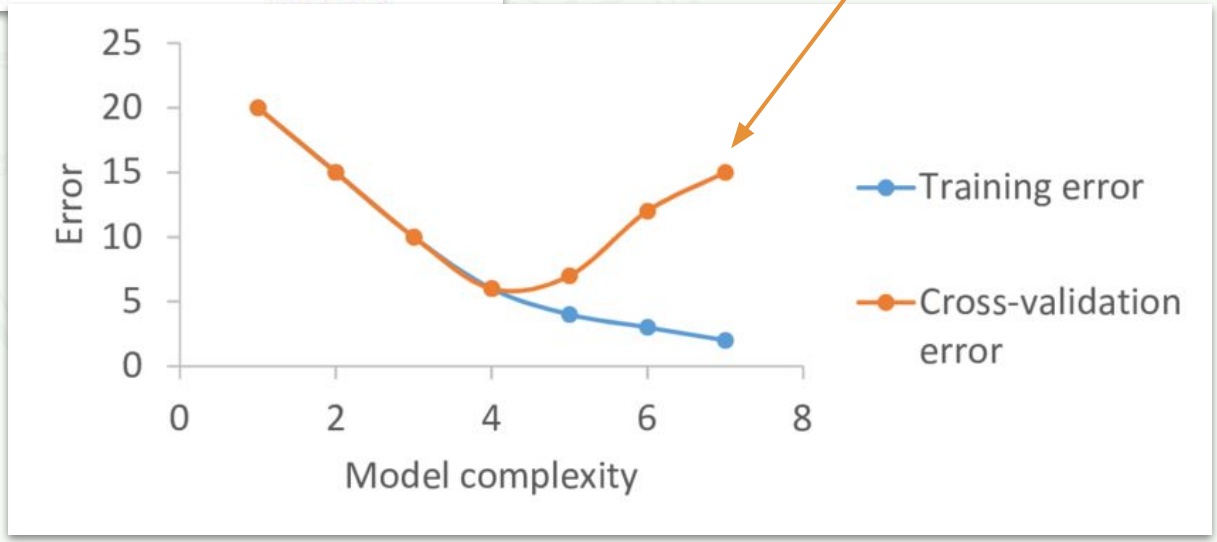
What to watch for:

- suppose you use 50/50; the classes are ordered in the dataset with the first class being the first 50% and the second class being the second 50% – big problems!!

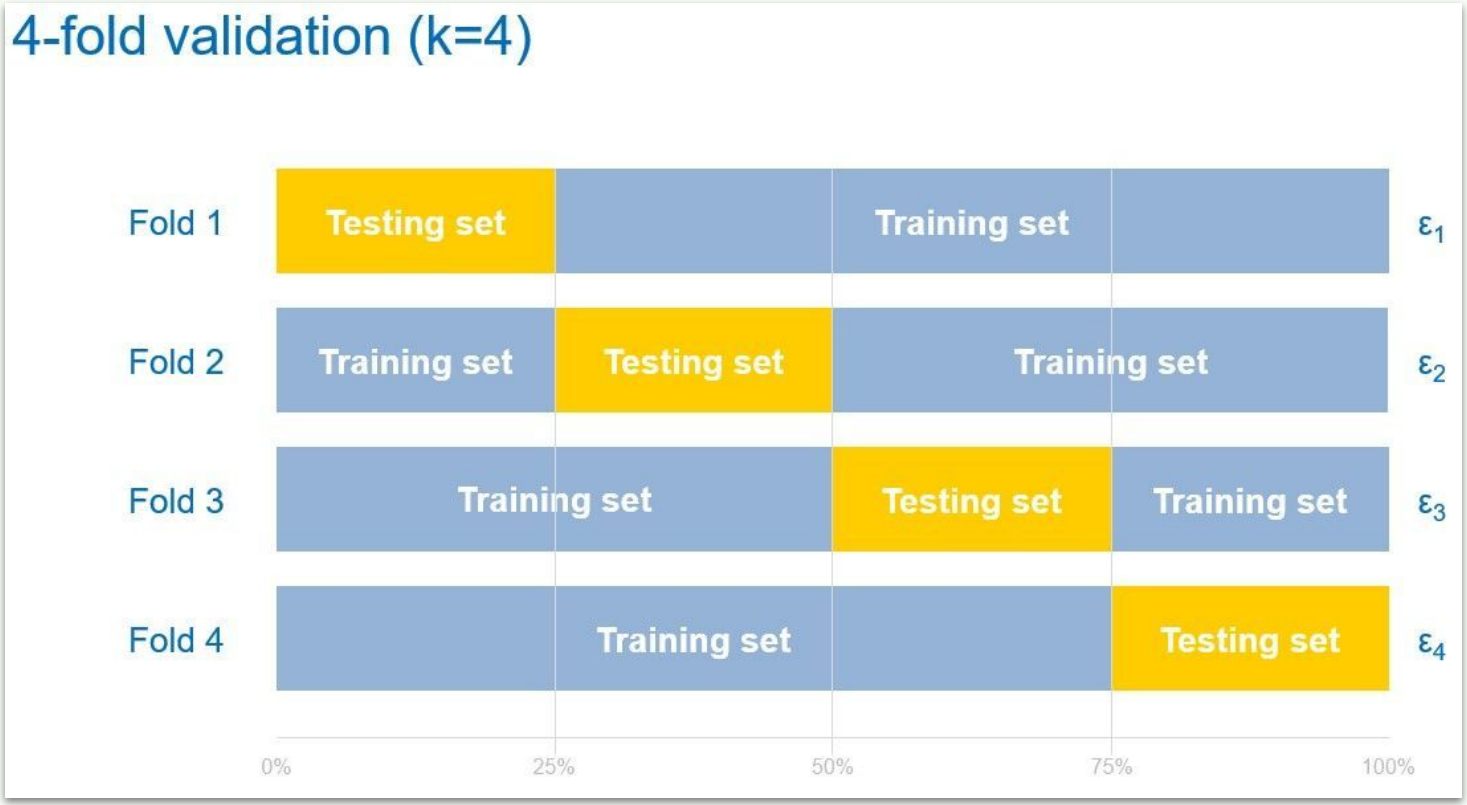
Connection to Bias-Variance Tradeoff



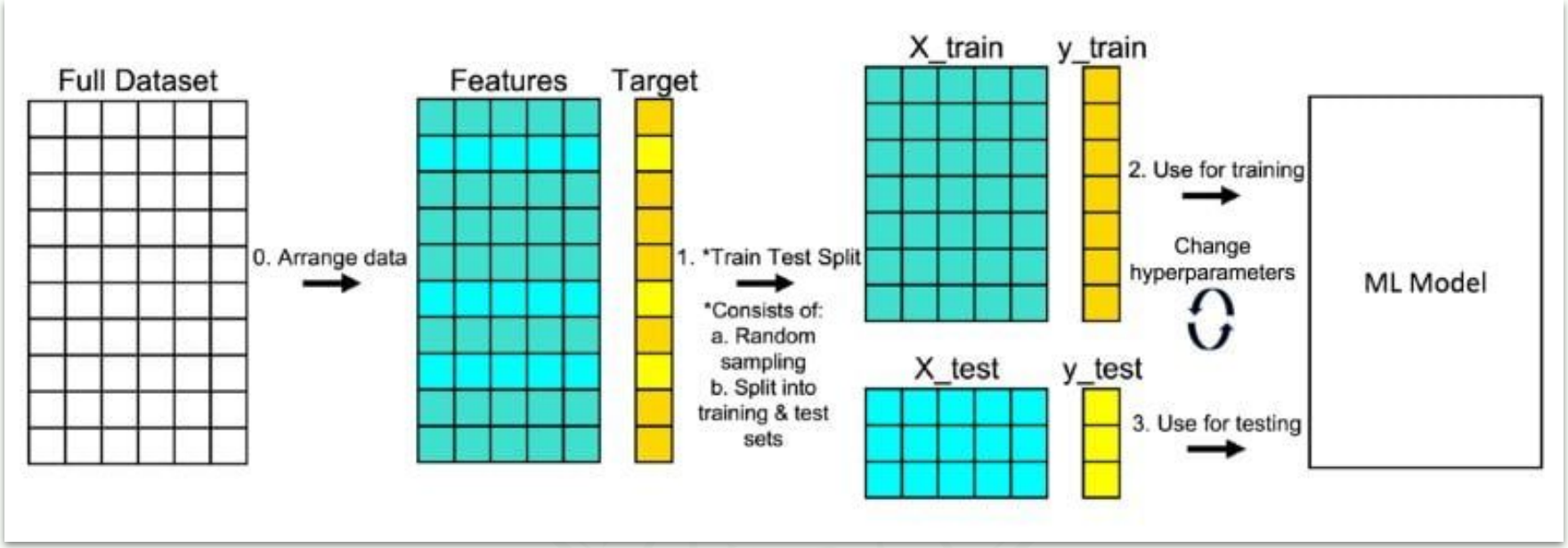
test data held back from the training step



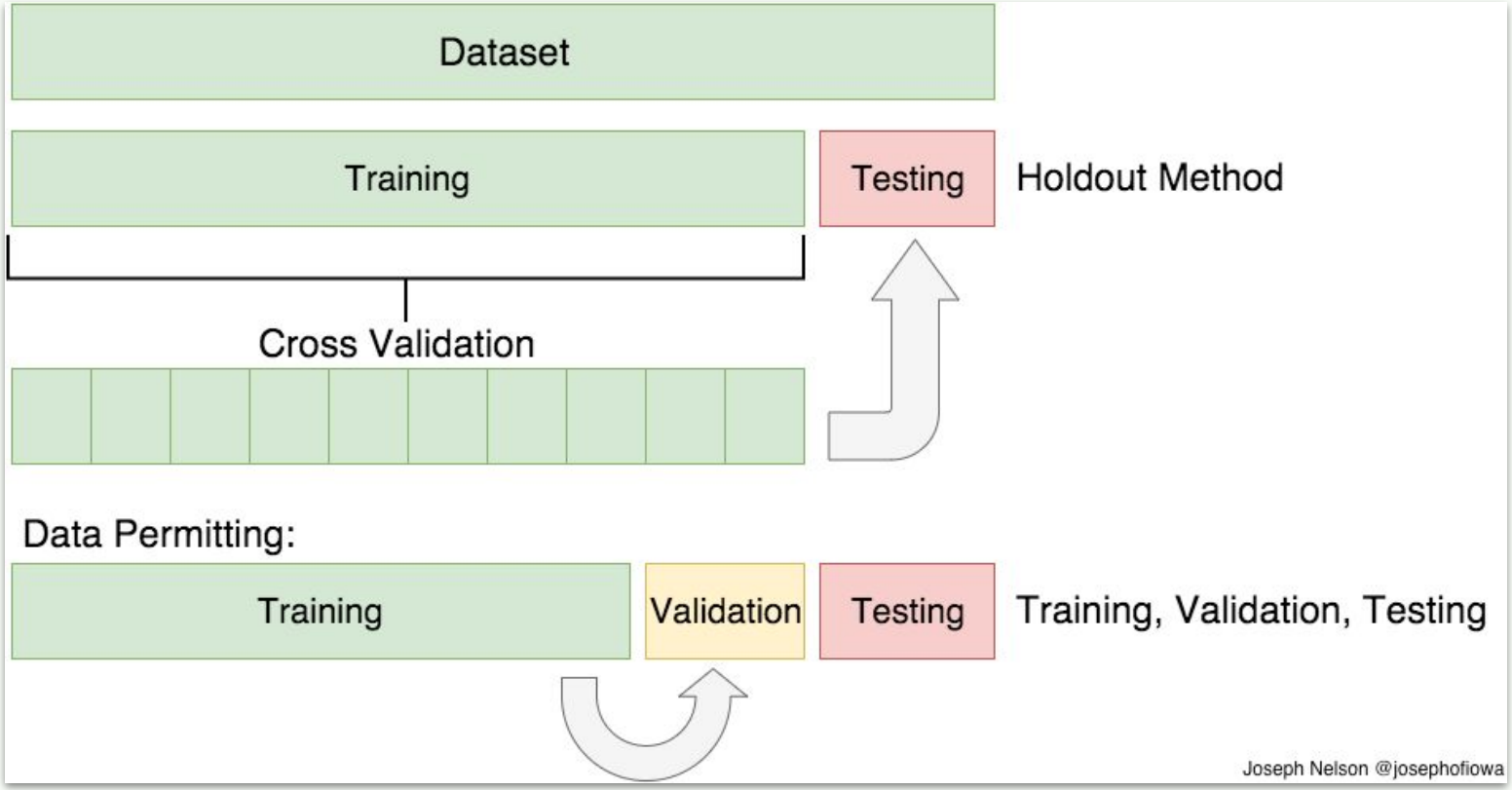
K-Fold Cross Validation



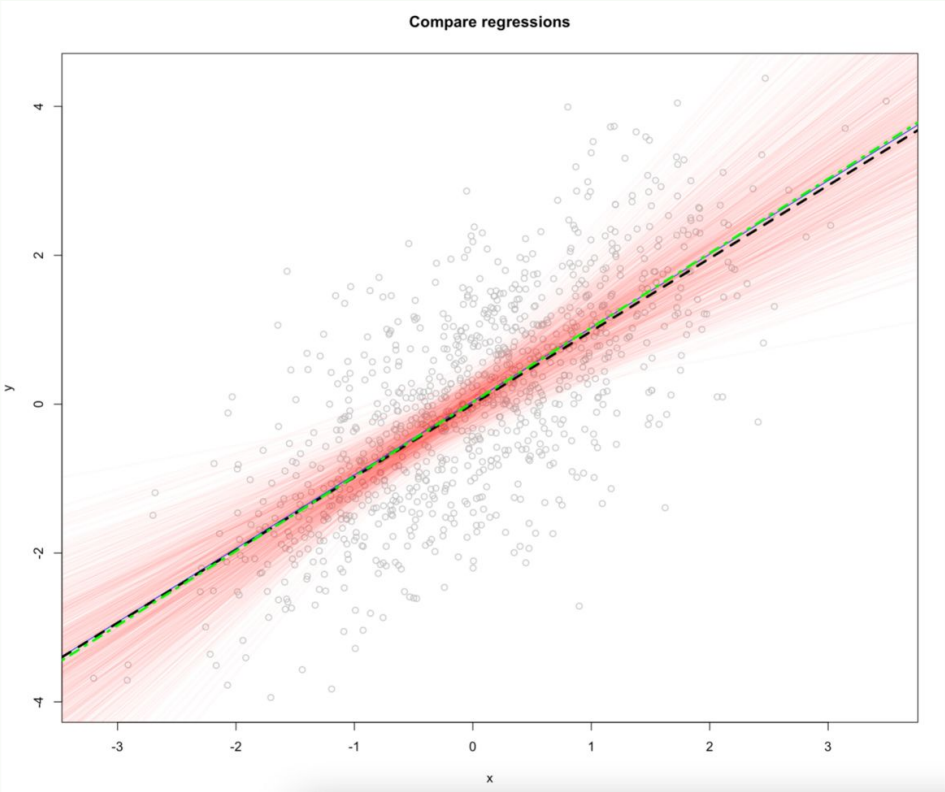
Test-Train Split



Test, Validate, Test Split



Resampling: Bootstrapping and Cross Validation



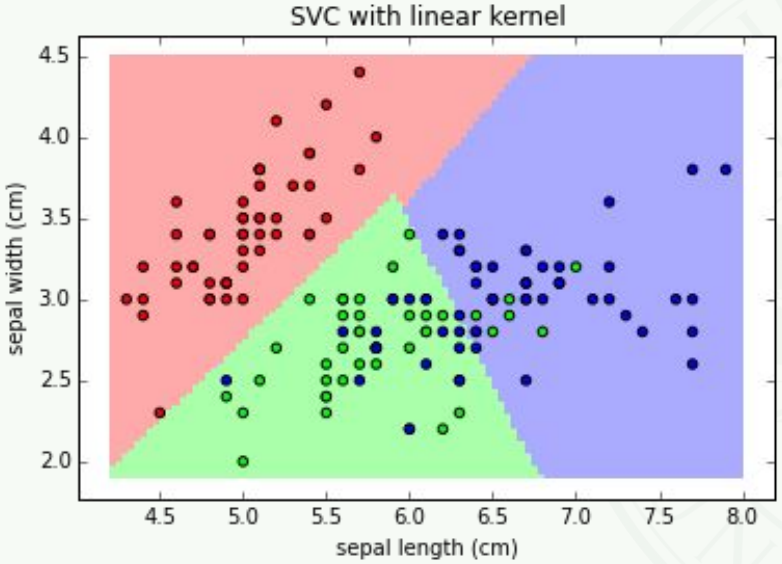
bootstrapping: sample with replacement

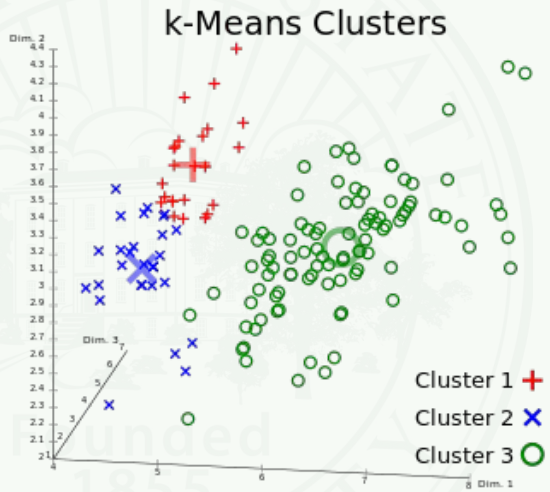
- establish distributions

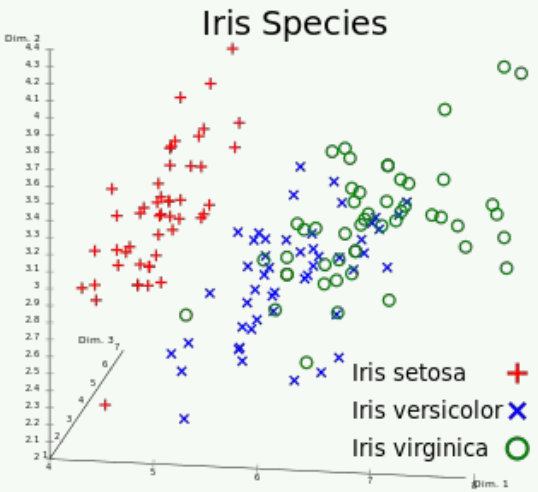
CV: sample without replacement

- measure generalization

ICA: Clustering and Classifying the Same Data







Example (The ICA for Wednesday!)

```
# get a dataset
from sklearn import datasets

# we want to split the data to test our ML model
from sklearn.model_selection import train_test_split

# ML models: one supervised, one unsupervised
from sklearn import neighbors
from sklearn.cluster import KMeans

# this is how we know how well it works (using the split)
from sklearn.metrics import accuracy_score

# compare final predictions
import matplotlib.pyplot as plt

# get the data
iris = datasets.load_iris()
```

Classification Requires an *X* and a *y*

```
1 # here, pull out the X and y parts of the data
2 X = iris["data"]
3 y = iris["target"]
```

```
1 # here, look at y so that you can see how the classes are encoded
2 y
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
```

Split Data, Fit to Training Portion

```
1 # KNN: supervised learning (uses y)
2
3 test_fraction = 0.2
4
5 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_fraction)
6
7 my_classifier = neighbors.KNeighborsClassifier()
8
9 my_classifier.fit(X_train, y_train)
```

ML steps:

1. define the ML model
2. use that model to fit to the data

It is thus very easy to use many different estimators!

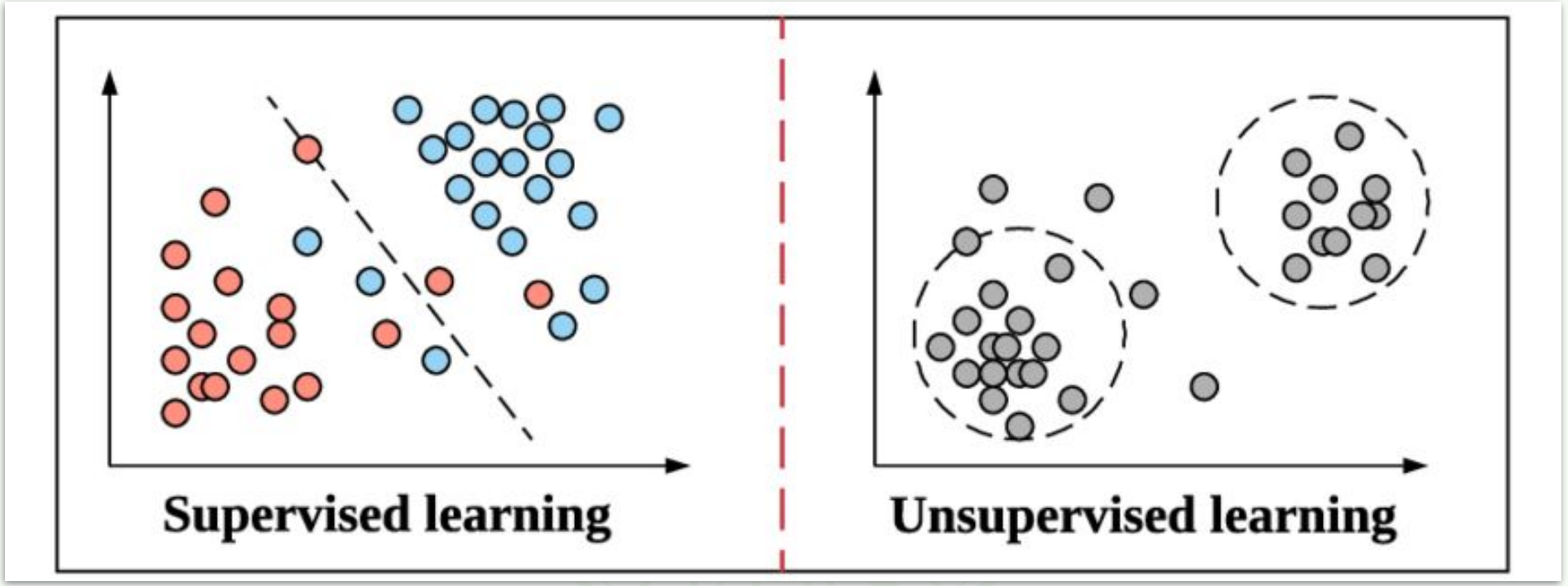
Make Predictions: Use Test Data for Score

```
[39]  1 my_predictions = my_classifier.predict(X_test)
      2
      3 print(accuracy_score(y_test, my_predictions))
      4
```

```
0.9333333333333333
```

Note that this code is independent of the estimator.

ICA Goal: Scikit-Learn and Two Learning Approaches



colored

not colored

HW: Confusion Matrix

Classification has its own metrics.