

Machine Learning: Scikit-Learn, Classification and Clustering

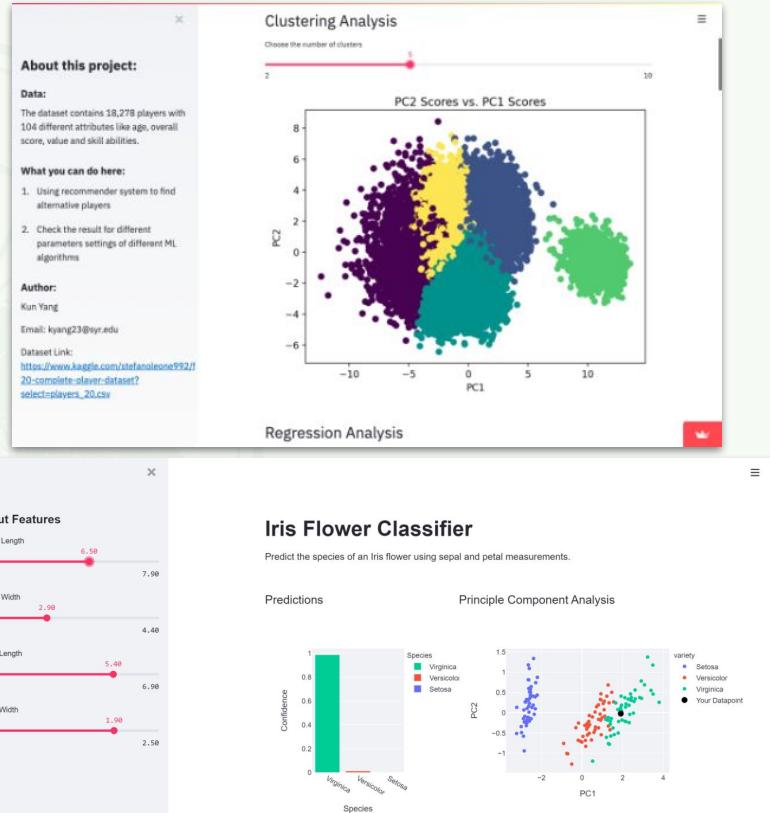
Murillo

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



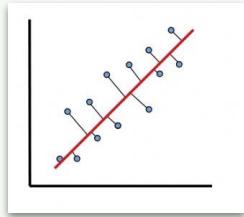
Schedule and Extra Credit

- 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



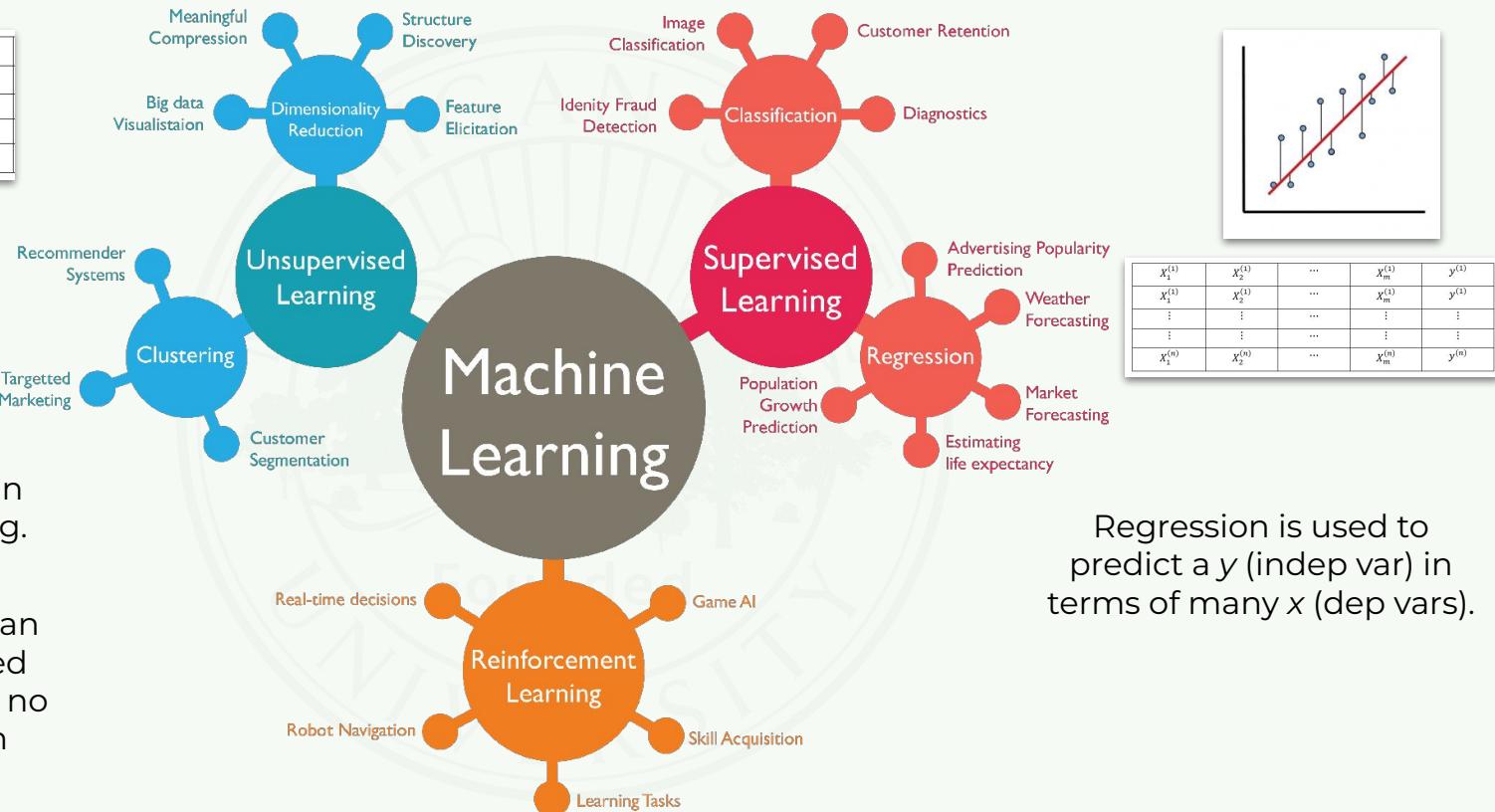
Organization of Machine Learning

$X_1^{(1)}$	$X_2^{(1)}$...	$X_m^{(1)}$
$X_1^{(1)}$	$X_2^{(1)}$...	$X_m^{(1)}$
:	:	...	:
:	:	...	:



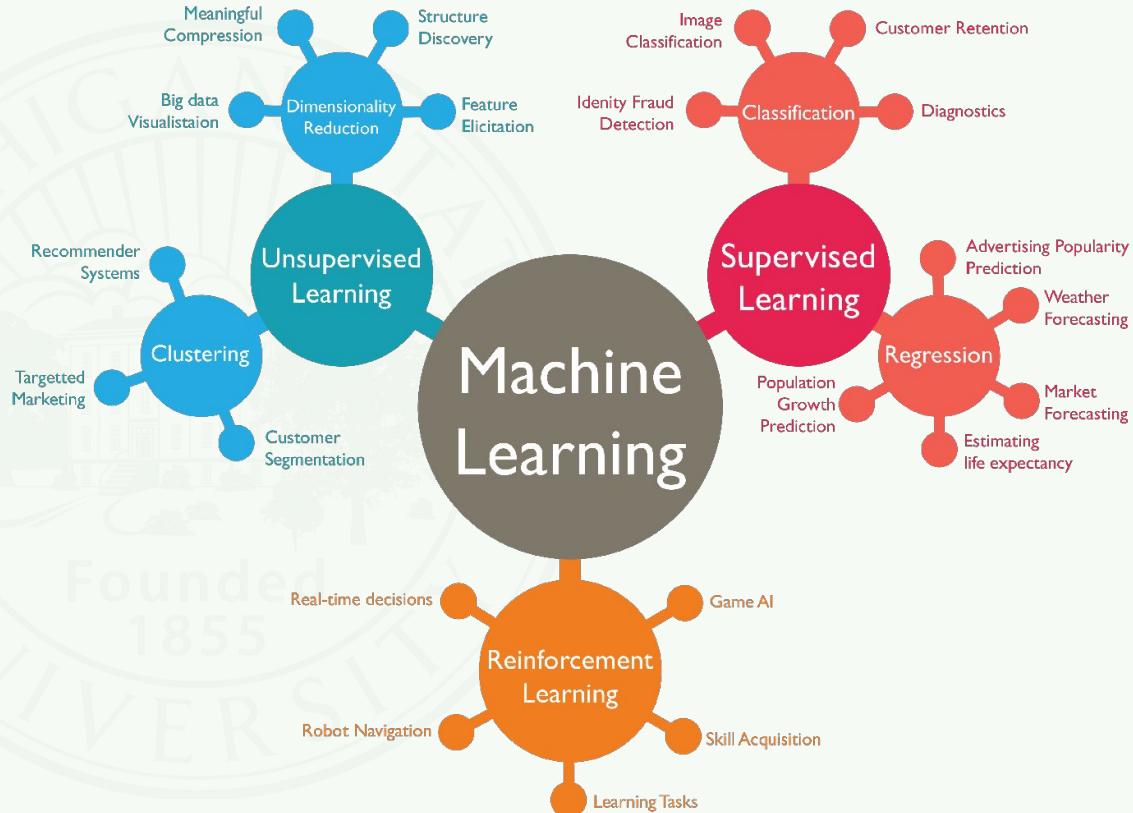
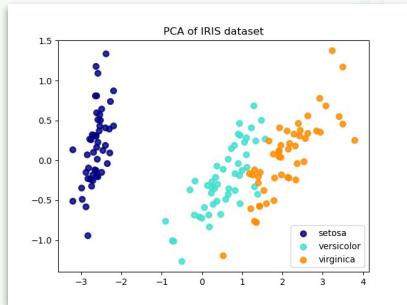
There is no known y in unsupervised learning.

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

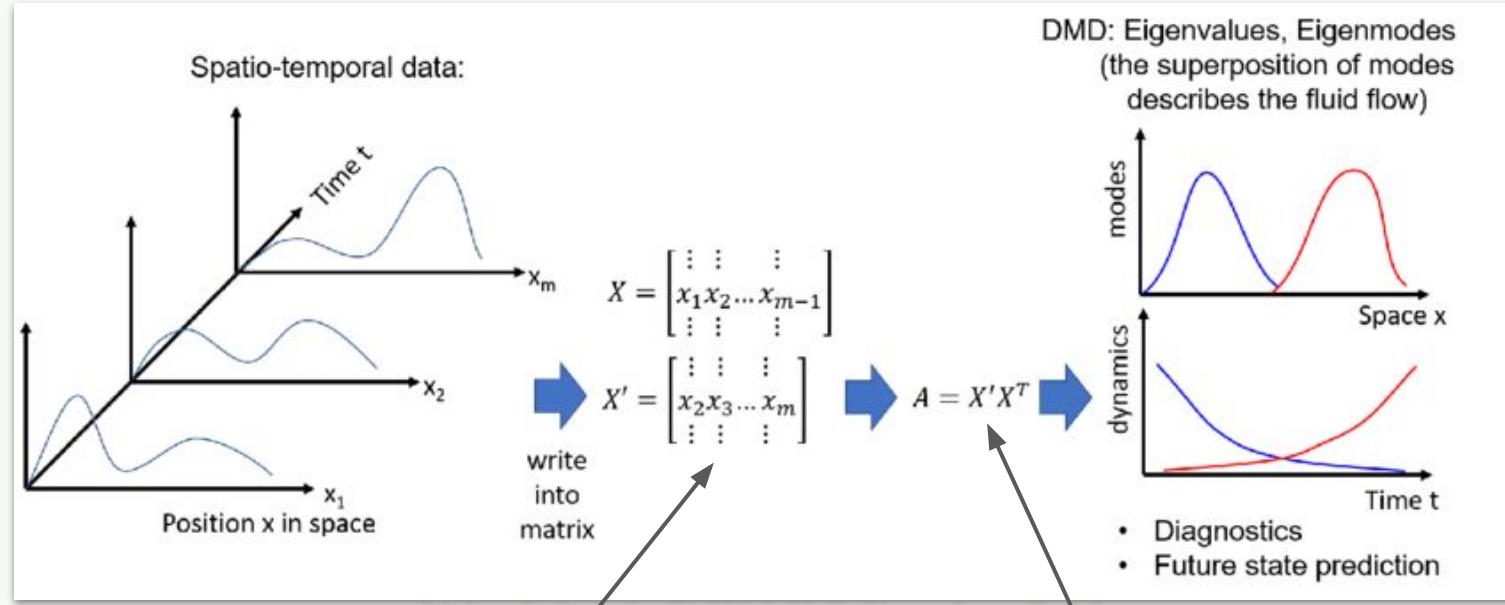


Dimensionality Reduction

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 →

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

average over time

space →

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

average over space



SVD/PCA Are Common, But Not Always Best

SVD vs. CUR

$$\text{SVD: } A = U \Sigma V^T$$

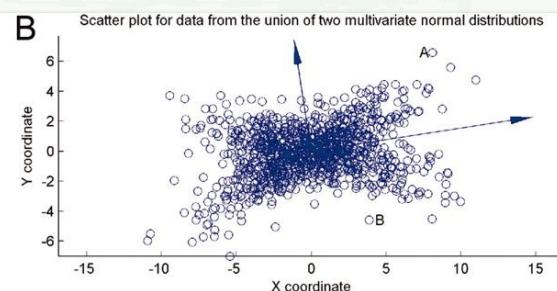
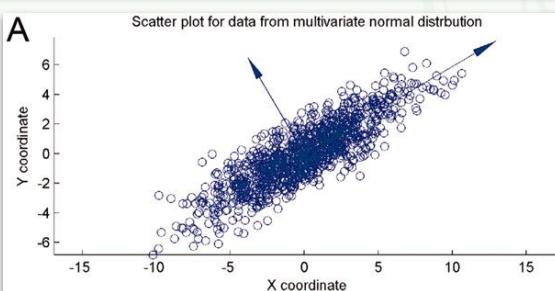
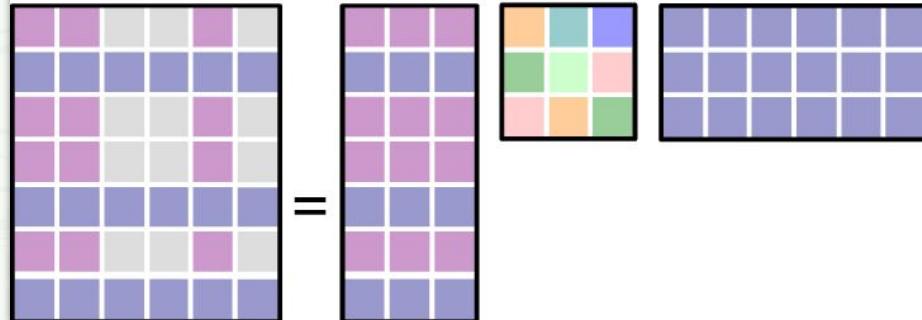
Huge but sparse Big and dense
sparse and small

$$\text{CUR: } A = C U R$$

Huge but sparse Big but sparse
dense but small

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, <http://www.mmds.org>

A $m \times n$ **C** $m \times k$ **U** $k \times k$ **R** $k \times n$



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



CUR Reference

PNAS

ARTICLES ▾

FRONT MATTER

AUTHORS ▾

TOPICS +

RESEARCH ARTICLE | COMPUTER SCIENCES | ✓



CUR matrix decompositions for improved data analysis

Michael W. Mahoney  and Petros Drineas [Authors Info & Affiliations](#)

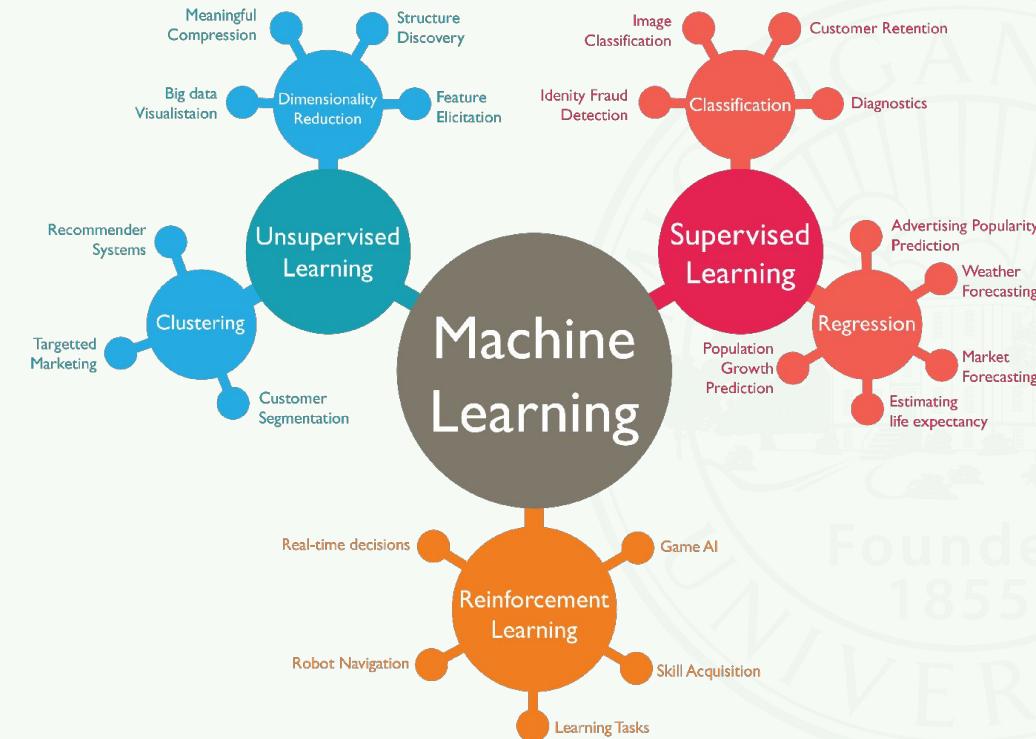
Edited by Jon Kleinberg, Cornell University, Ithaca, NY, and accepted by the Editorial Board November 11, 2008

January 20, 2009 | 106 (3) 697-702 | <https://doi.org/10.1073/pnas.0803205106>

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



Learning



Machine Learning:

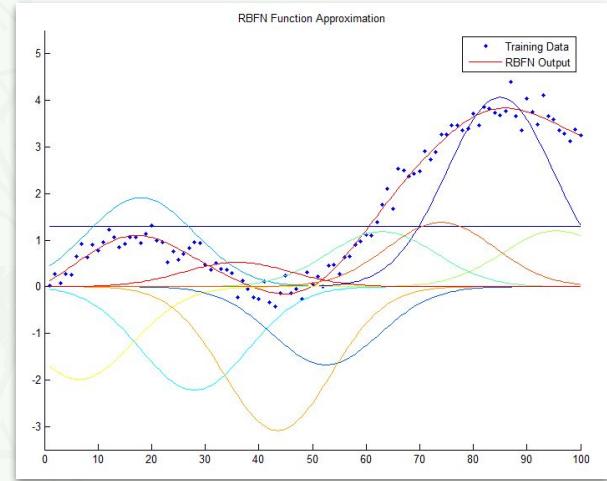
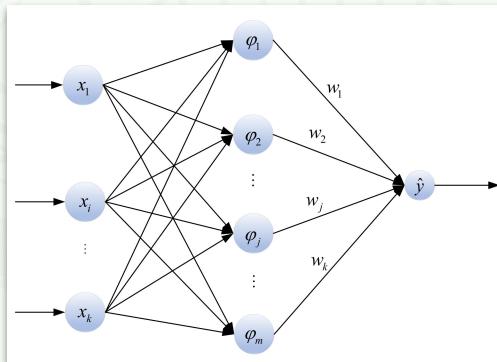
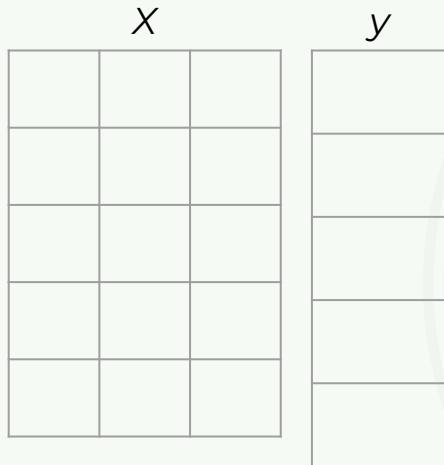
the use and development of computer systems that are able to learn and adapt without following explicit instructions, by using algorithms and statistical models to analyze and draw inferences from patterns in data

Supervised Learning

data

model

prediction



- preprocessing:
 - IDA, EDA
 - missing values
 - scaling
- fits:
 - lines/hyperplanes
 - nonlinear functions
- methods:
 - linear algebra
 - optimization
- how well did we do!?
- define a score using test data

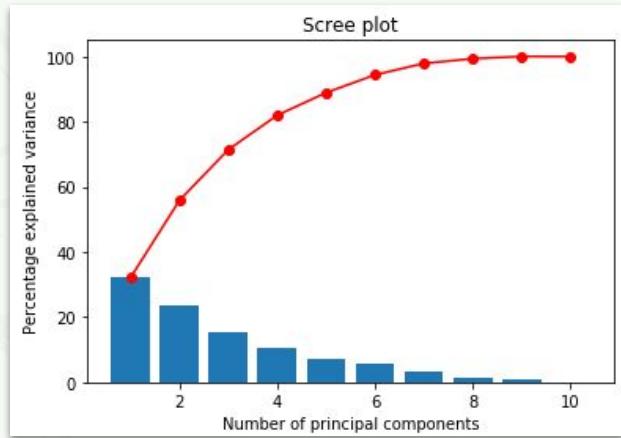
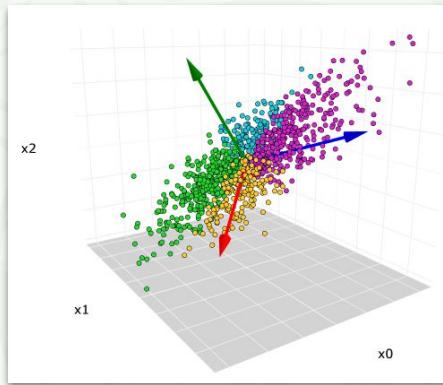
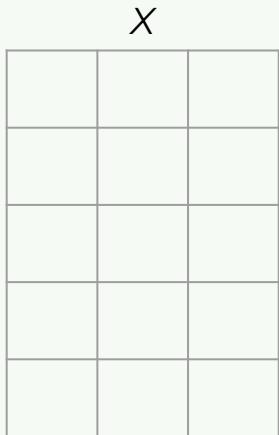


Unsupervised Learning

data

model

prediction?

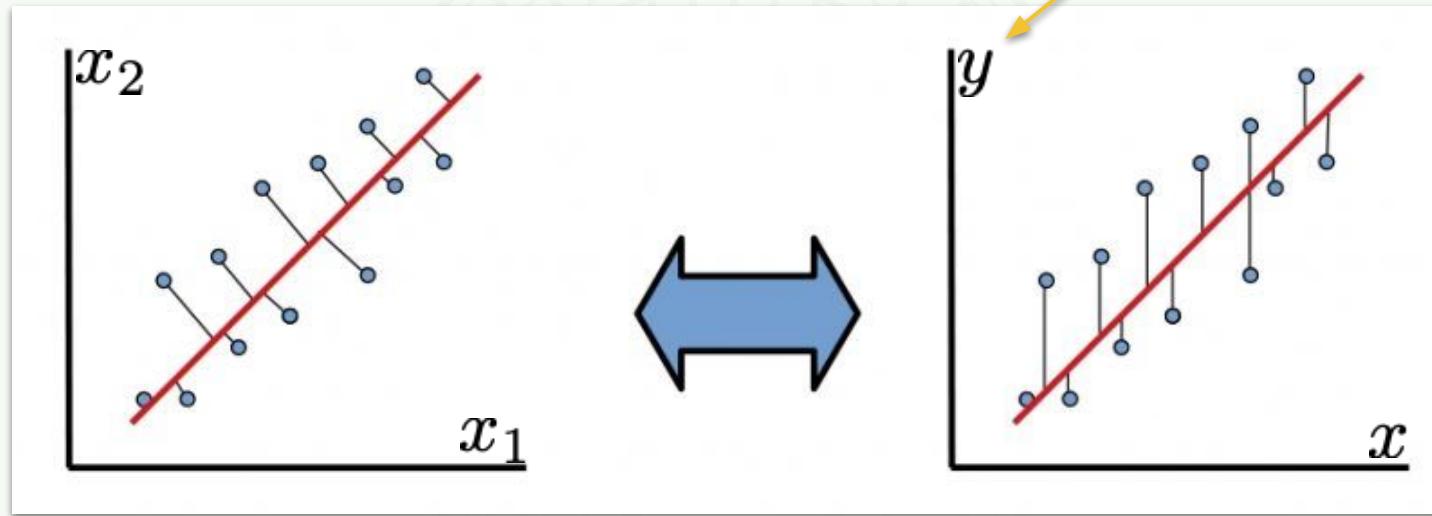


- preprocessing:
 - IDA, EDA
 - missing values
 - scaling
- methods:
 - linear algebra
 - optimization
- how well did we do!?



Unsupervised and Supervised Compared

we have no labels/targets; all variables equivalent

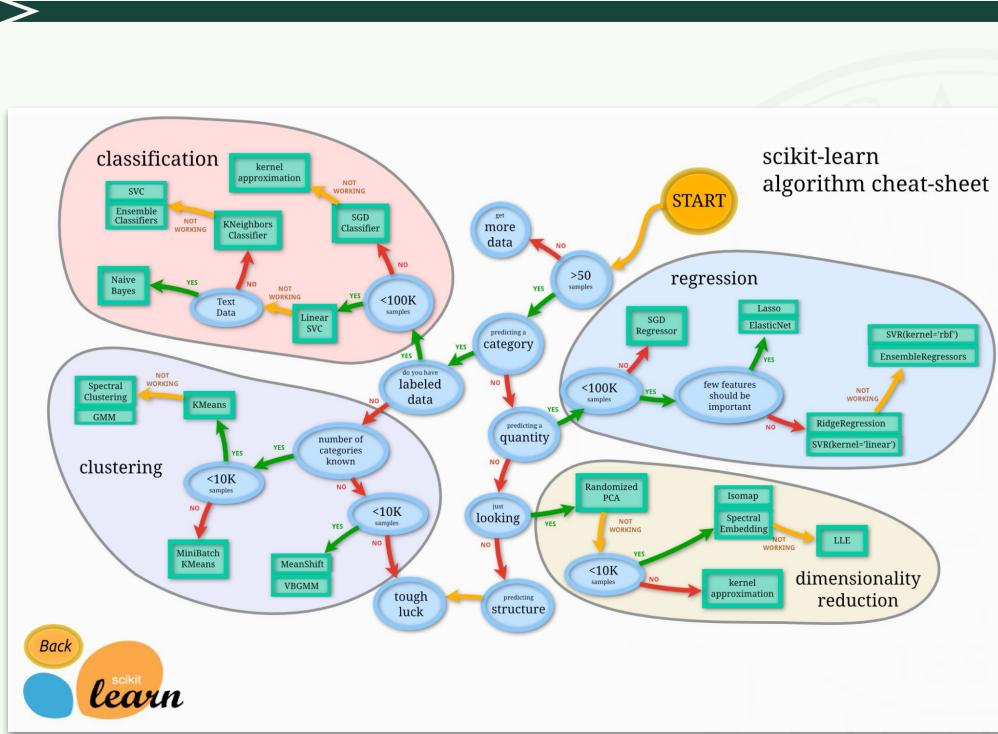


find structure in the data

make predictions



Scikit-Learn



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

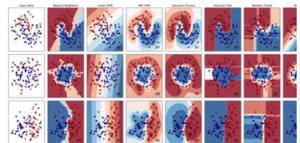


Scikit-Learn Organization

Classification

Identifying which category an object belongs to.

Applications: Spam detection, image recognition.
Algorithms: SVM, nearest neighbors, random forest, and more...

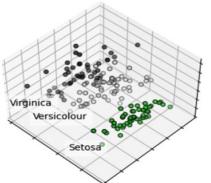


Examples

Dimensionality reduction

Reducing the number of random variables to consider.

Applications: Visualization, Increased efficiency
Algorithms: PCA, feature selection, non-negative matrix factorization, and more...

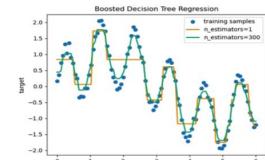


Examples

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.
Algorithms: SVR, nearest neighbors, random forest, and more...



Examples

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: k-Means, spectral clustering, mean-shift, and more...

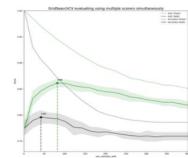


Examples

Model selection

Comparing, validating and choosing parameters and models.

Applications: Improved accuracy via parameter tuning
Algorithms: grid search, cross validation, metrics, and more...

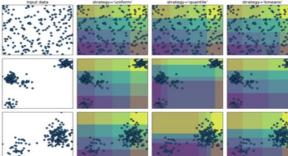


Examples

Preprocessing

Feature extraction and normalization.

Applications: Transforming input data such as text for use with machine learning algorithms.
Algorithms: preprocessing, feature extraction, and more...



Examples

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

The screenshot shows a section of the scikit-learn documentation titled "Developing scikit-learn estimators". The page includes a sidebar with links for "Install", "User Guide", "API", "Examples", "Community", and "More". A note at the top says "Please cite us if you use the software." Below the title, there's a paragraph about developing estimators. A sub-section "APIs of scikit-learn objects" is shown, with a note that all objects have a common basic API. It defines "Different objects" and lists "Estimator" and "Predictor" types. Examples of code snippets for each are provided.

Developing scikit-learn estimators

Whether you are proposing an estimator for inclusion in scikit-learn, developing a separate package compatible with scikit-learn, or implementing custom components for your own projects, this chapter details how to develop objects that safely interact with scikit-learn Pipelines and model selection tools.

APIs of scikit-learn objects

To have a uniform API, we try to have a common basic API for all the objects. In addition, to avoid the proliferation of framework code, we try to adopt simple conventions and limit to a minimum the number of methods an object must implement.

Elements of the scikit-learn API are described more definitively in the [Glossary of Common Terms and API Elements](#).

Different objects

The main objects in scikit-learn are (one class can implement multiple interfaces):

Estimator: The base object, implements a `fit` method to learn from data, either:

```
estimator = estimator.fit(data, targets)
```

or:

```
estimator = estimator.fit(data)
```

Predictor: For supervised learning, or some unsupervised problems, implements:

```
prediction = predictor.predict(data)
```

Classification algorithms usually also offer a way to quantify certainty of a prediction, either using `decision_function` or `predict_proba`:

```
probability = predictor.predict_proba(data)
```

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

The screenshot shows the scikit-learn documentation page for the `StandardScaler` class. The top navigation bar includes links for Install, User Guide, API, Examples, Community, and More. On the left, there's a sidebar with links for `sklearn.preprocessing.StandardScaler`, Examples using `StandardScaler`, and a note to cite the software. The main content area has a title `sklearn.preprocessing.StandardScaler` and a code snippet for its definition. It describes the class as standardizing features by removing the mean and scaling to unit variance. It also provides the formula for calculating the standard score: $z = (x - u) / s$, where u is the mean and s is the standard deviation. A note states that centering and scaling happen independently on each feature.

scikit-learn 1.1.3
Other versions

Please cite us if you use the software.

`sklearn.preprocessing.StandardScaler`
Examples using `sklearn.preprocessing.StandardScaler`

sklearn.preprocessing.StandardScaler

```
class sklearn.preprocessing.StandardScaler(*, copy=True, with_mean=True, with_std=True)
```

[source]

Standardize features by removing the mean and scaling to unit variance.

The standard score of a sample x is calculated as:

$$z = (x - u) / s$$

where u is the mean of the training samples or zero if `with_mean=False`, and s is the standard deviation of the training samples or one if `with_std=False`.

Centering and scaling happen independently on each feature by computing the relevant statistics on the samples in the training set. Mean and standard deviation are then stored to be used on later data using `transform`.

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

Prev Up Next

scikit-learn 1.1.3
Other versions

Please cite us if you use the software.

3.3. Metrics and scoring:
quantifying the quality of predictions

3.3.1. The scoring parameter:
defining model evaluation rules
3.3.2. Classification metrics
3.3.3. Multilabel ranking metrics
3.3.4. Regression metrics
3.3.5. Clustering metrics
3.3.6. Dummy estimators

3.3. Metrics and scoring: quantifying the quality of predictions

There are 3 different APIs for evaluating the quality of a model's predictions:

- **Estimator score method:** Estimators have a `.score` method providing a default evaluation criterion for the problem they are designed to solve. This is not discussed on this page, but in each estimator's documentation.
- **Scoring parameter:** Model-evaluation tools using `cross-validation` (such as `model_selection.cross_val_score` and `model_selection.GridSearchCV`) rely on an internal `scoring` strategy. This is discussed in the section [The scoring parameter](#): defining model evaluation rules.
- **Metric functions:** The `sklearn.metrics` module implements functions assessing prediction error for specific purposes. These metrics are detailed in sections on [Classification metrics](#), [Multilabel ranking metrics](#), [Regression metrics](#) and [Clustering metrics](#).

Finally, [Dummy estimators](#) are useful to get a baseline value of those metrics for random predictions.

See also: For "pairwise" metrics, between `samples` and not estimators or predictions, see the [Pairwise metrics, Affine Kernels](#) section.

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.

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



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.

The screenshot shows the official scikit-learn documentation for the `LinearRegression` class. The top navigation bar includes links for Install, User Guide, API, Examples, Community, and More. A sidebar on the left provides links for previous versions (1.1.3 and others), a citation notice, and examples for the `LinearRegression` class. The main content area features a large title `sklearn.linear_model.LinearRegression`. Below it is the class definition code:

```
class sklearn.linear_model.LinearRegression(*, fit_intercept=True, normalize='deprecated', copy_X=True, n_jobs=None, positive=False)
```

A link to the source code is provided at the end of this block. The description below the code states: "Ordinary least squares Linear Regression." It explains that the `LinearRegression` class fits a linear model to minimize the residual sum of squares between observed targets and predicted targets. A detailed description of the `fit_intercept` parameter follows, explaining its default value of `True` and its behavior when set to `False`.



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:

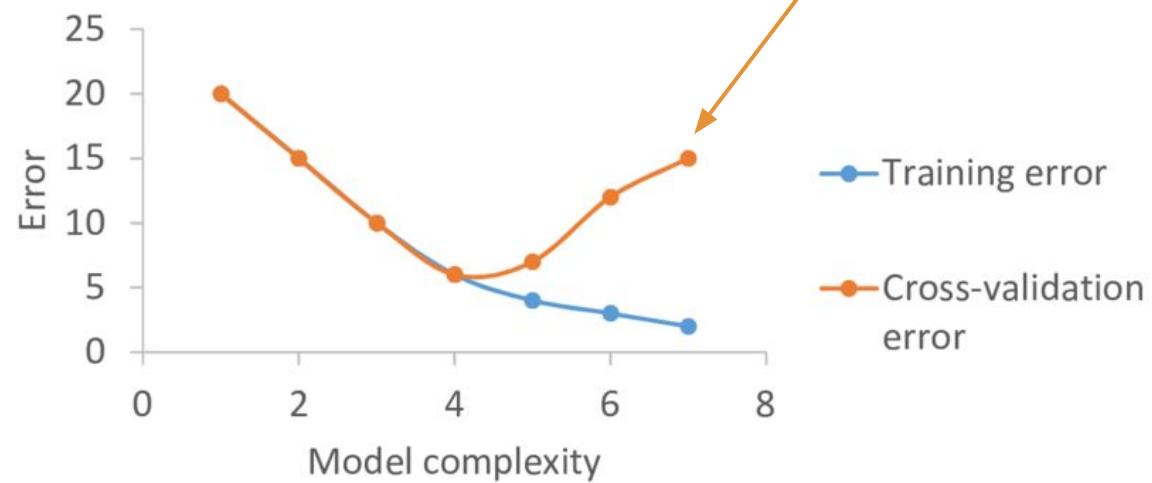
- suppose you use 50/50; the classes are ordered in the dataset with the first class being the first 50% and the second class being the second 50% – big problems!!



Connection to Bias-Variance Tradeoff



test data held back
from the training step

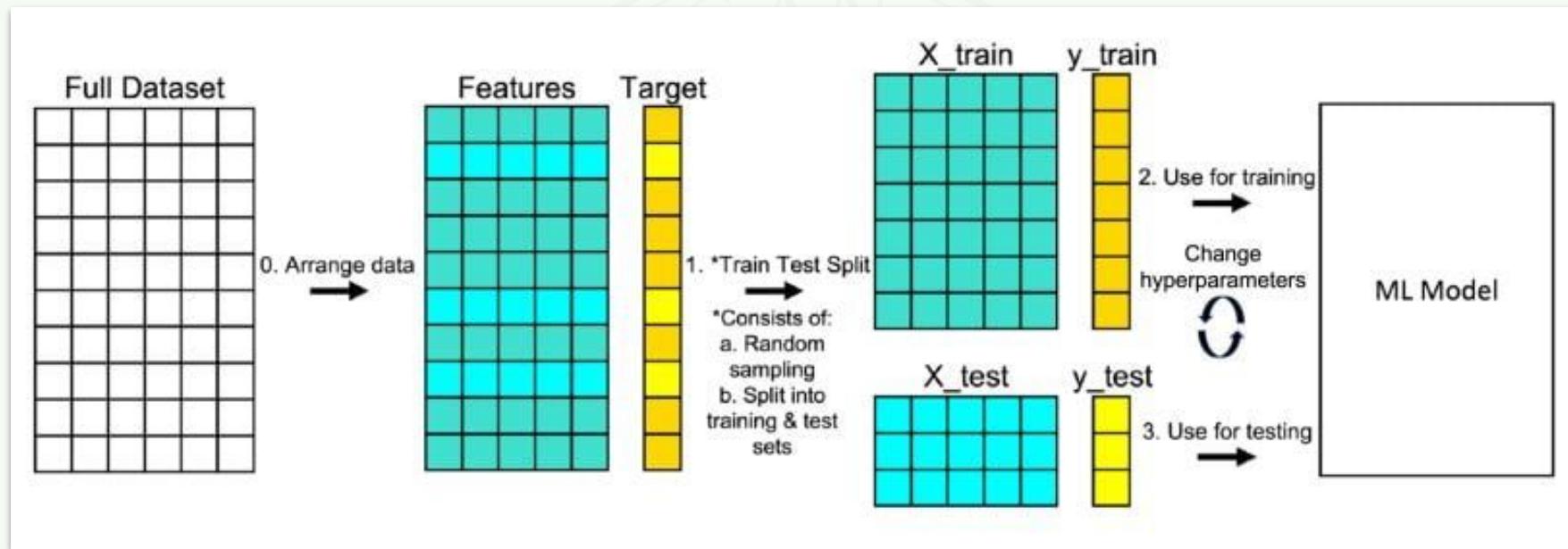


K-Fold Cross Validation

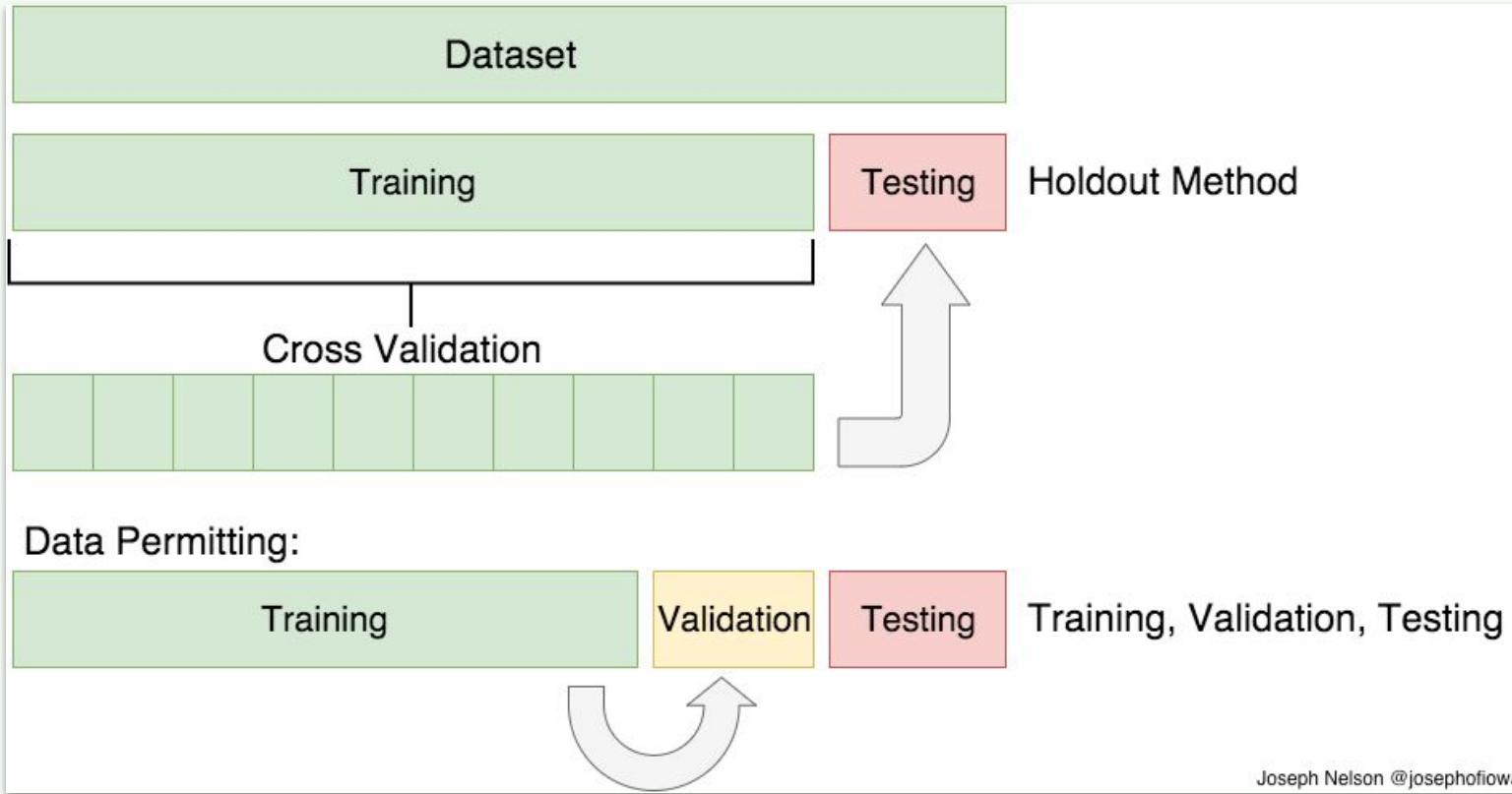
4-fold validation (k=4)



Test-Train Split



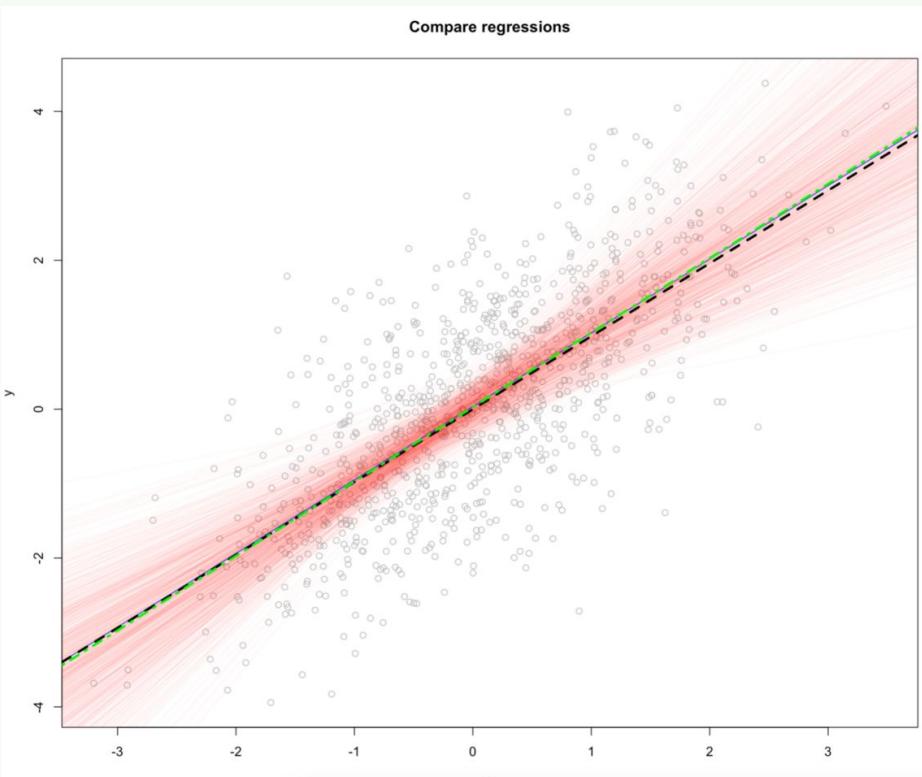
Test, Validate, Test Split



Joseph Nelson @josephofiowa



Resampling: Bootstrapping and Cross Validation



bootstrapping: sample with replacement

- establish distributions

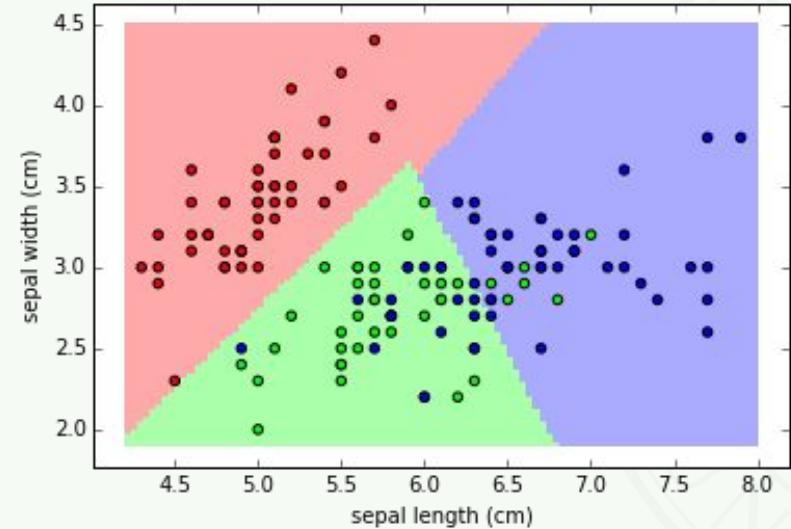
CV: sample without replacement

- measure generalization

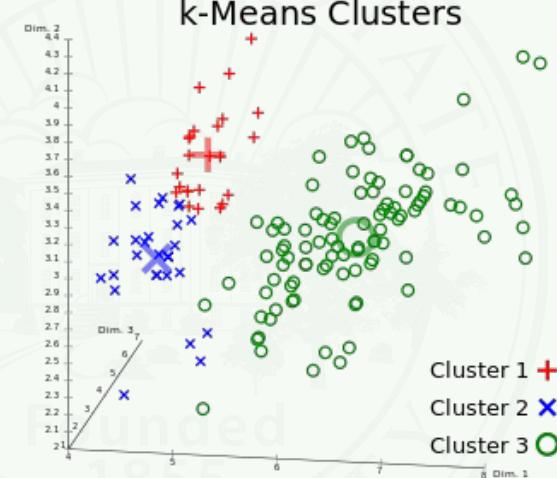


ICA: Clustering and Classifying the Same Data

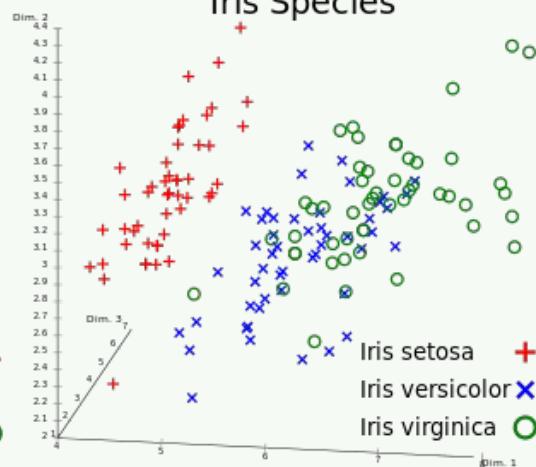
SVC with linear kernel



k-Means Clusters



Iris Species



Example (The ICA for Wednesday!)

```
> 1 # get a dataset
2 from sklearn import datasets
3
4 # we want to split the data to test our ML model
5 from sklearn.model_selection import train_test_split
6
7 # ML models: one supervised, one unsupervised
8 from sklearn import neighbors
9 from sklearn.cluster import KMeans
10
11 # this is how we know how well it works (using the split)
12 from sklearn.metrics import accuracy_score
13
14 # compare final predictions
15 import matplotlib.pyplot as plt
16
17 # get the data
18 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, 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

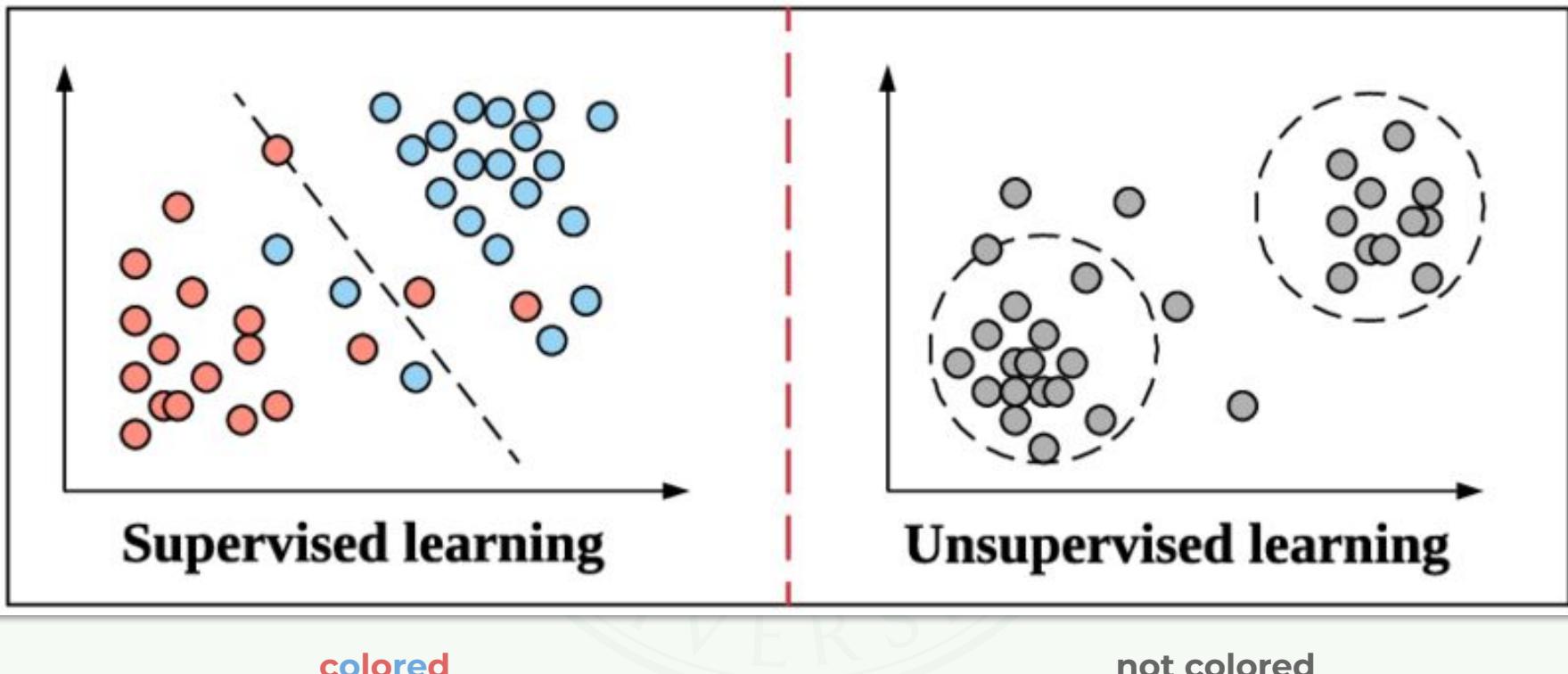
```
[39] 1 my_predictions = my_classifier.predict(X_test)
2
3 print(accuracy_score(y_test, my_predictions))
4
```

```
0.9333333333333333
```

Note that this code is independent of the estimator.



ICA Goal: Scikit-Learn and Two Learning Approaches



HW: Confusion Matrix

Classification has its own metrics.

		True Classes
Predicted Classes	True Positive	False Positive
	True Positive Case: Cyber Attack Model: Alarm activated Result: You saved your server	False Positive Case: NO Cyber Attack Model: Alarm activated Result: You didn't lose anything but got tensed
	False Negative Case: Cyber Attack Model: Alarm NOT activated Result: You lost your data	True Negative Case: NO Cyber Attack Model: Alarm NOT activated Result: All good

