EM 538-001: PRACTICAL MACHINE LEARNING FOR ENGINEERING ANALYTICS

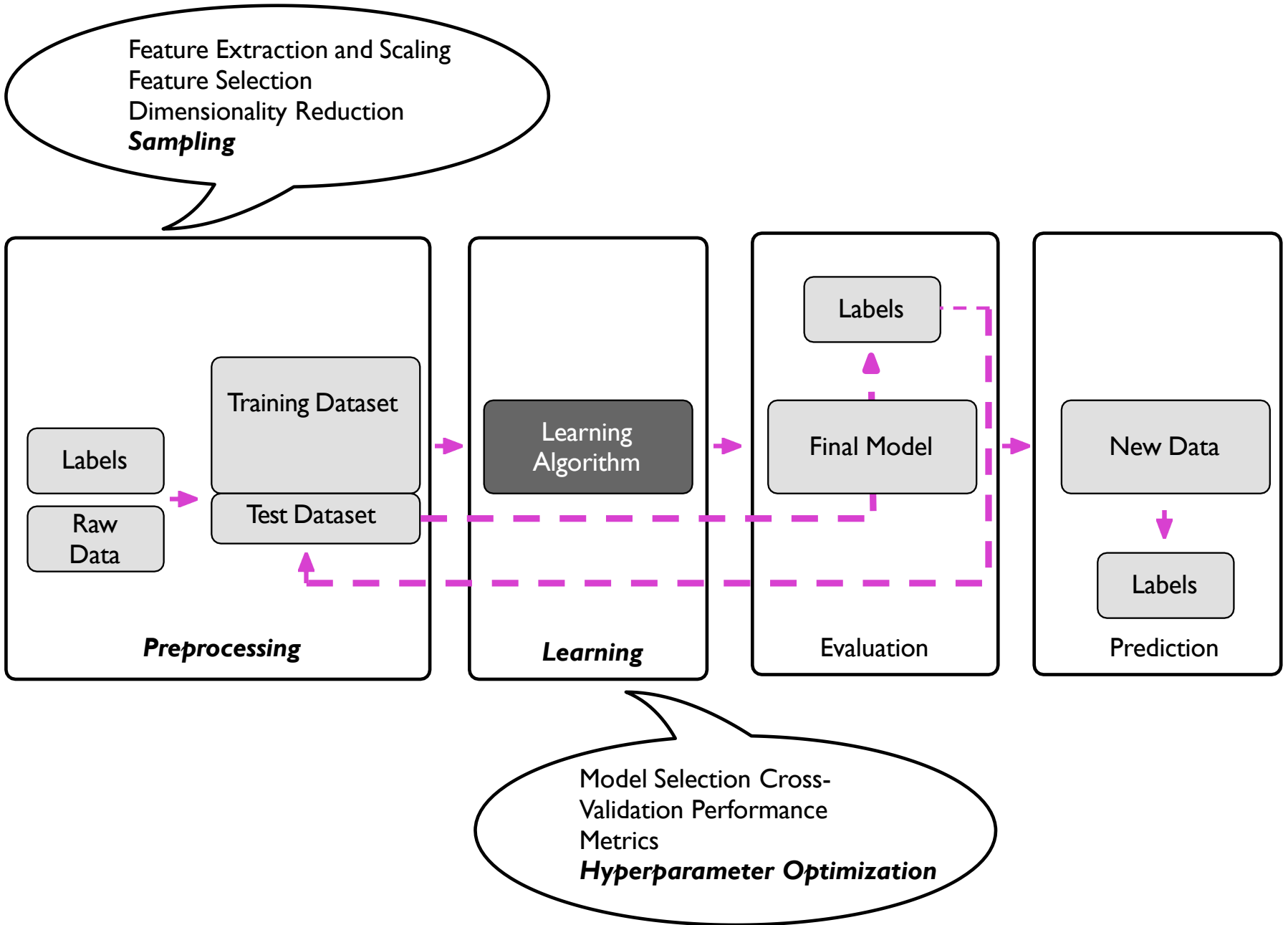
LECTURE 007

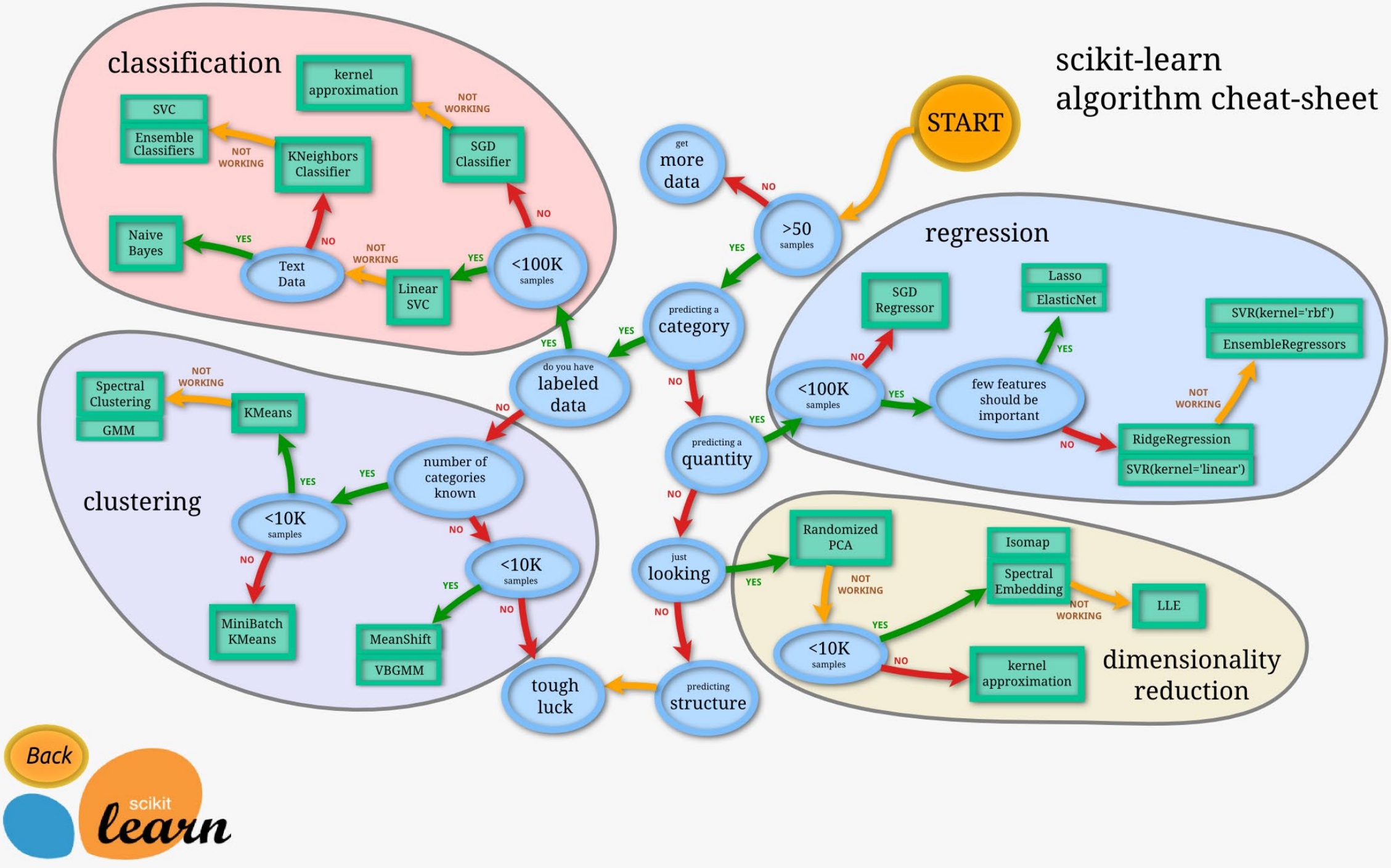
Fred Livingston, Ph.D.

NEAREST NEIGHBOOR MACHINE LEARNING MODELS

- Model Performance
- Introduction to Unsupervised Learning

MACHINE LEARNING WORKFLOW

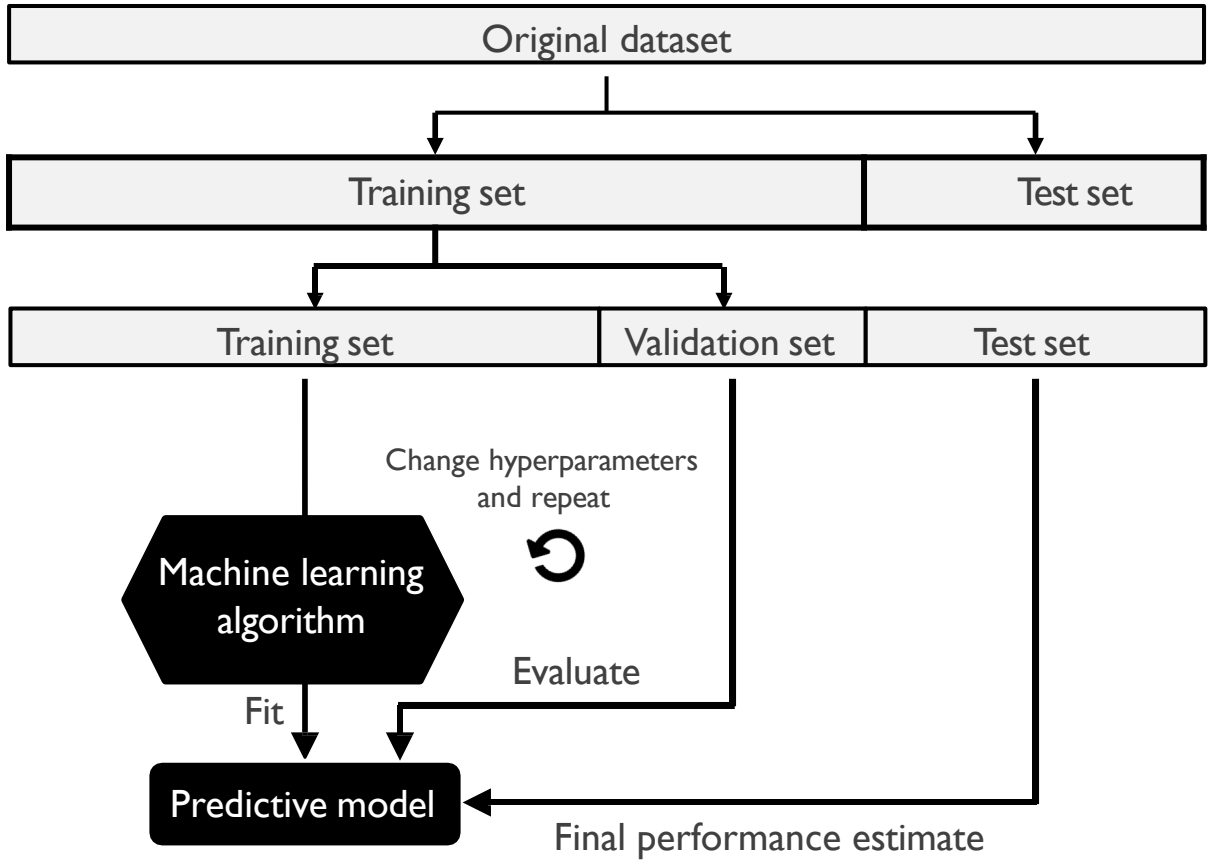



scikit-learn
algorithm cheat-sheet
START
classification
kernel approximation
SVC
Ensemble Classifiers
KNeighbors Classifier
SGD Classifier
Naive Bayes
Text Data
Linear SVC
<100K samples
NOT WORKING
NO
YES
get more data
>50 samples
predicting a category
do you have labeled data
regression
SGD Regressor
Lasso
ElasticNet
SVR(kernel='rbf')
EnsembleRegressors
<100K samples
few features should be important
RidgeRegression
SVR(kernel='linear')
predicting a quantity
clustering
Spectral Clustering
GMM
KMeans
number of categories known
<10K samples
MiniBatch KMeans
MeanShift
VBGMM
just looking
Randomized PCA
Isomap
Spectral Embedding
LLE
kernel approximation
dimensionality reduction
predicting structure
tough luck
Back
scikit learn

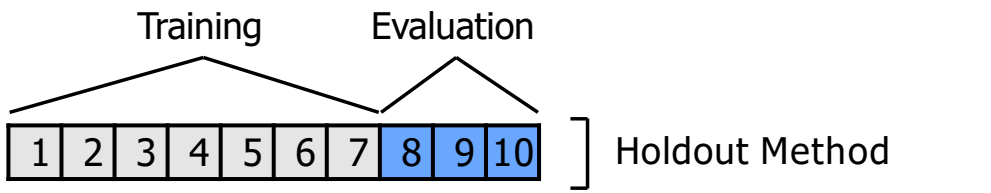
HYPERPARAMETERS

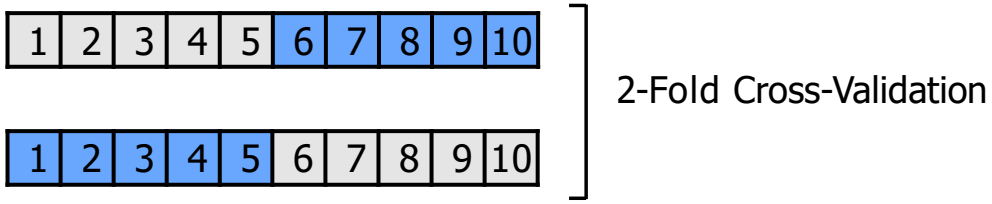
- Value of *k*
- Scaling of the feature axes
- Distance measure
- Weighting of the distance measure

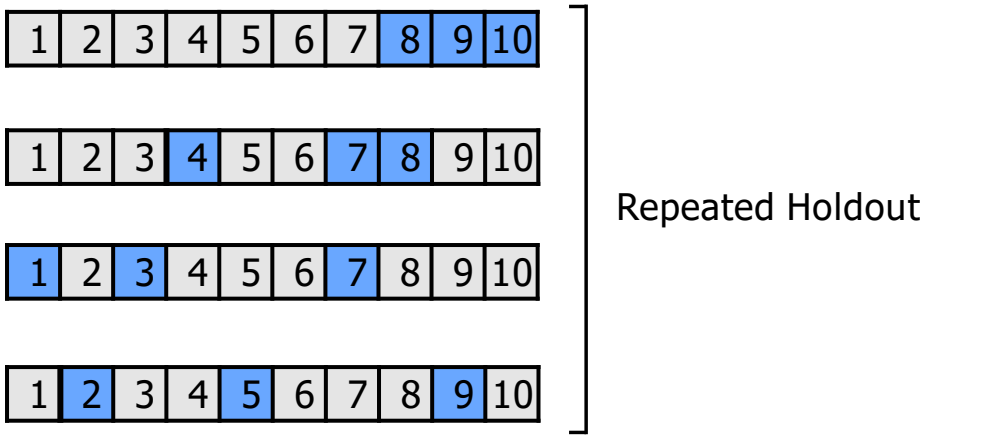
3-WAY HOLDOUT METHOD



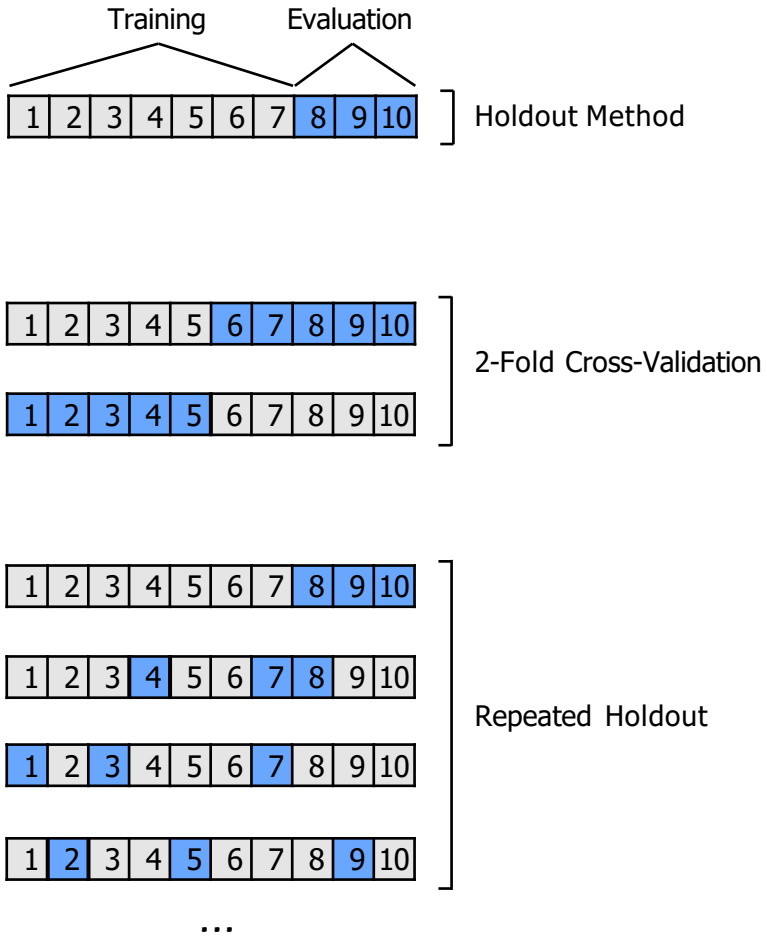
K-FOLD CV SPECIAL CASES: K=2 & K=N





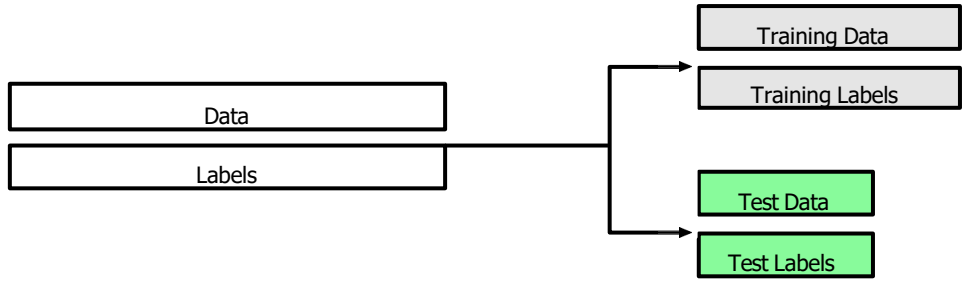


K-FOLD CV SPECIAL CASES: K=2 & K=N

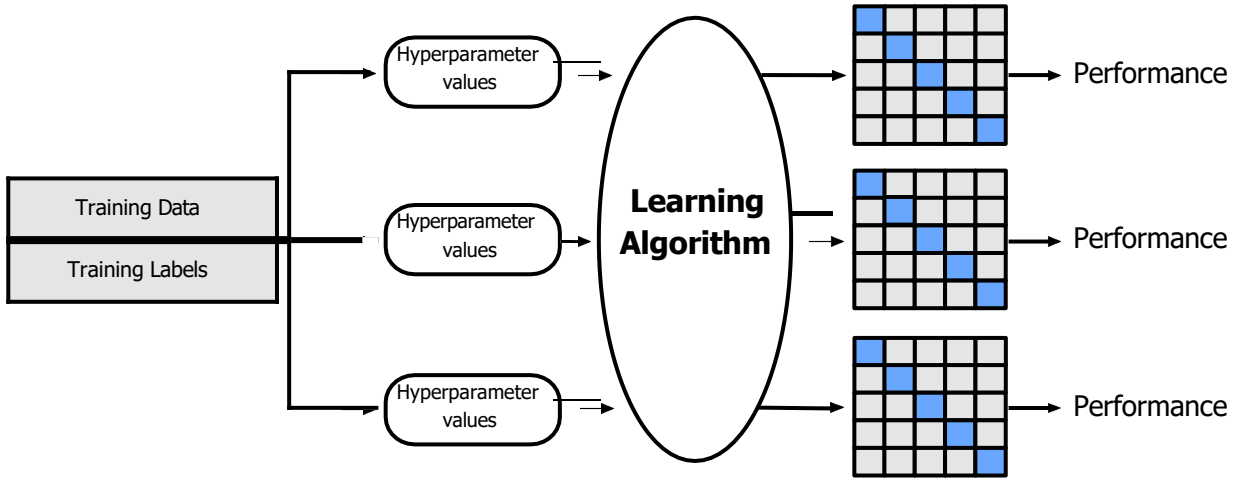


LOOCV

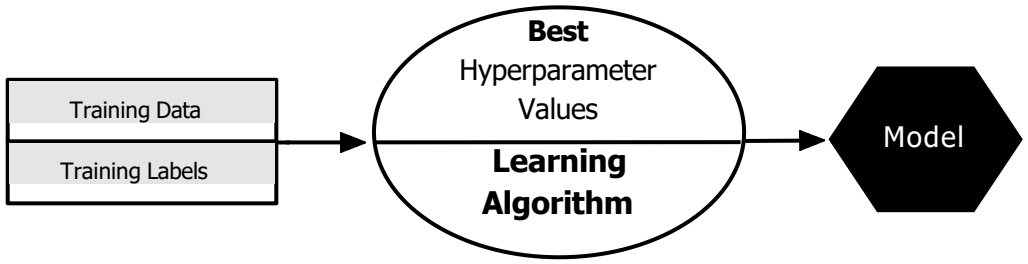
1


Data
Labels
Training Data
Training Labels
Test Data
Test Labels

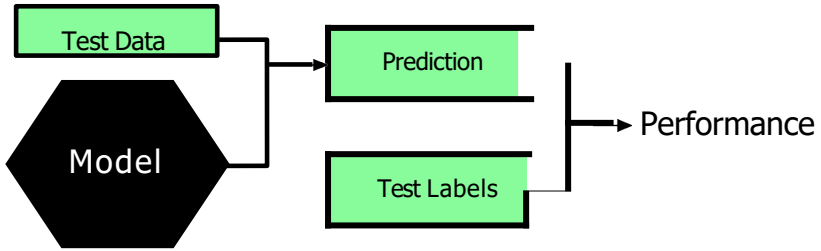
2


Training Data
Training Labels
Hyperparameter values
Hyperparameter values
Hyperparameter values
Learning Algorithm
Performance
Performance
Performance

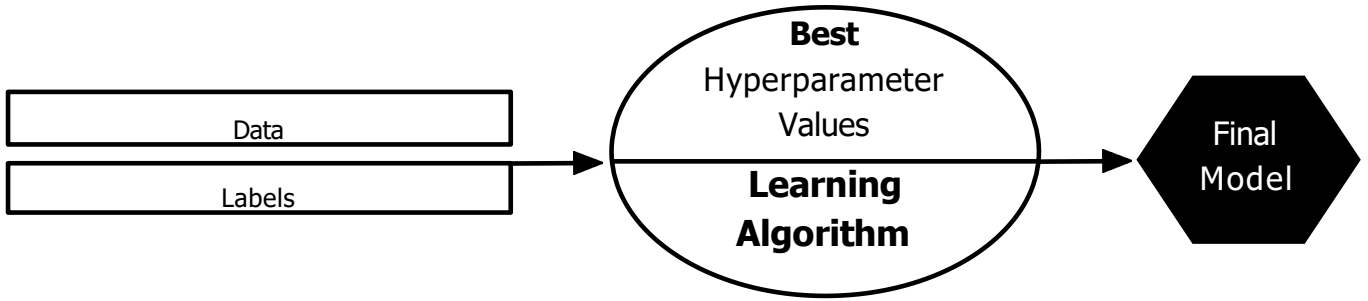
3


Training Data
Training Labels
Best
Hyperparameter
Values
Learning
Algorithm
Model

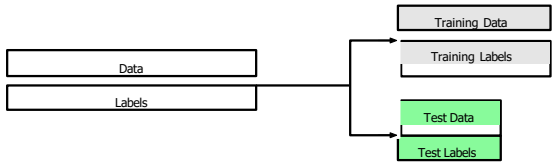
4


Test Data
Prediction
Model
Performance
Test Labels

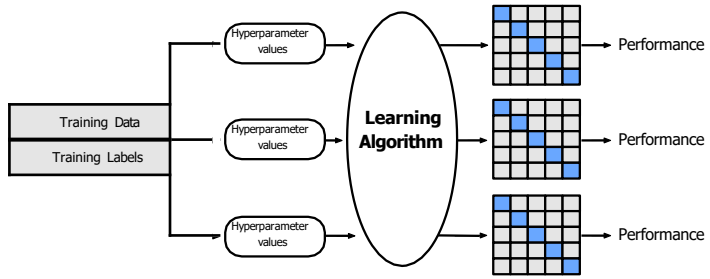
5


Data
Labels
Best
Hyperparameter
Values
Learning
Algorithm
Final
Model

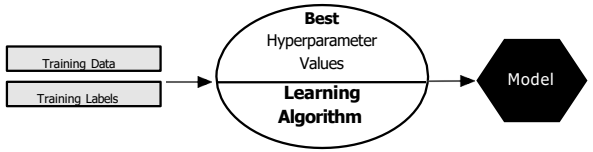
1


Data
Labels
Training Data
Training Labels
Test Data
Test Labels

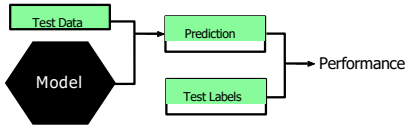
2


Training Data
Training Labels
Hyperparameter values
Hyperparameter values
Hyperparameter values
Learning Algorithm
Performance
Performance
Performance

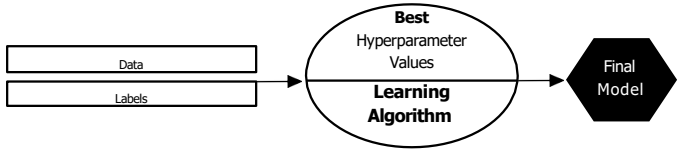
3


Training Data
Training Labels
Best Hyperparameter Values
Learning Algorithm
Model

4


Test Data
Model
Prediction
Test Labels
Performance

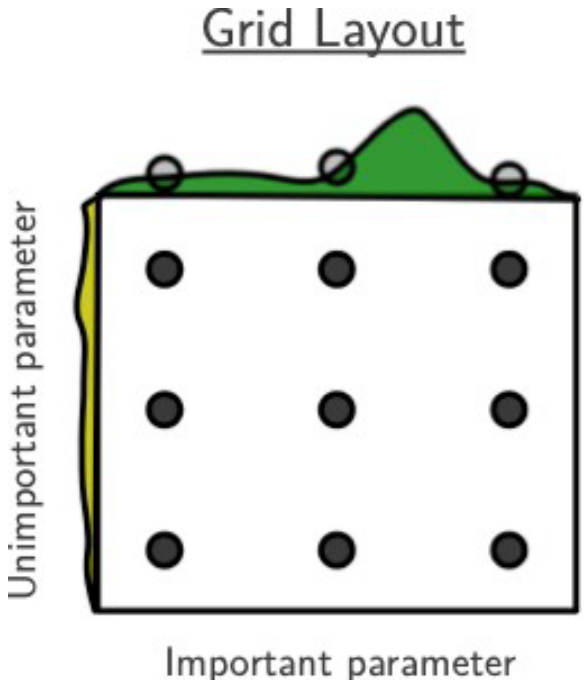
5


Data
Labels
Best Hyperparameter Values
Learning Algorithm
Final Model

GRID SEARCH

`GridSearchCV`

- Exhaustive search
- Thorough but expensive
- Specify grid for parameter search
- Can be run in parallel
- Can suffer from poor coverage
- Often run with multiple resolutions



Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *The Journal of Machine Learning Research*, *13*(1), 281-305.

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

RANDOMIZED SEARCH

`RandomizedSearchCV`

- Search based on a time budget
- Preferred if there are many hyperparameters (e.g. > 3 distinct ones)
- specify distribution for parameter search
- can be run in parallel

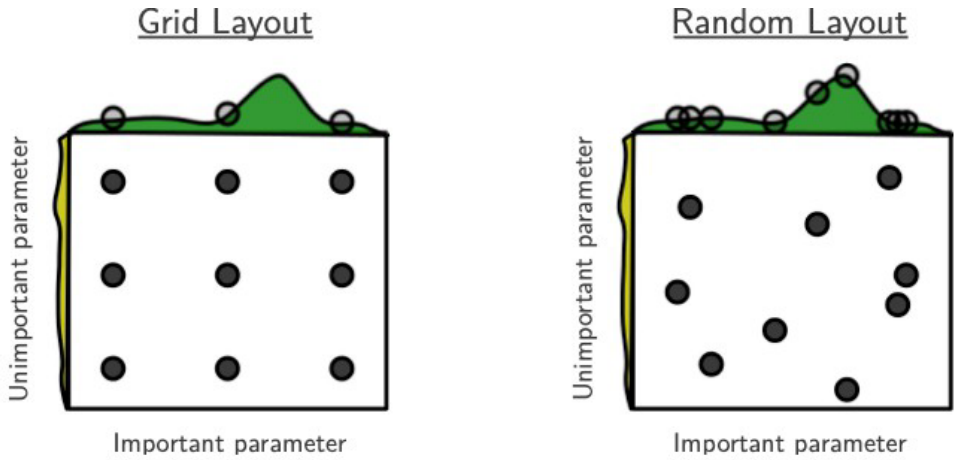
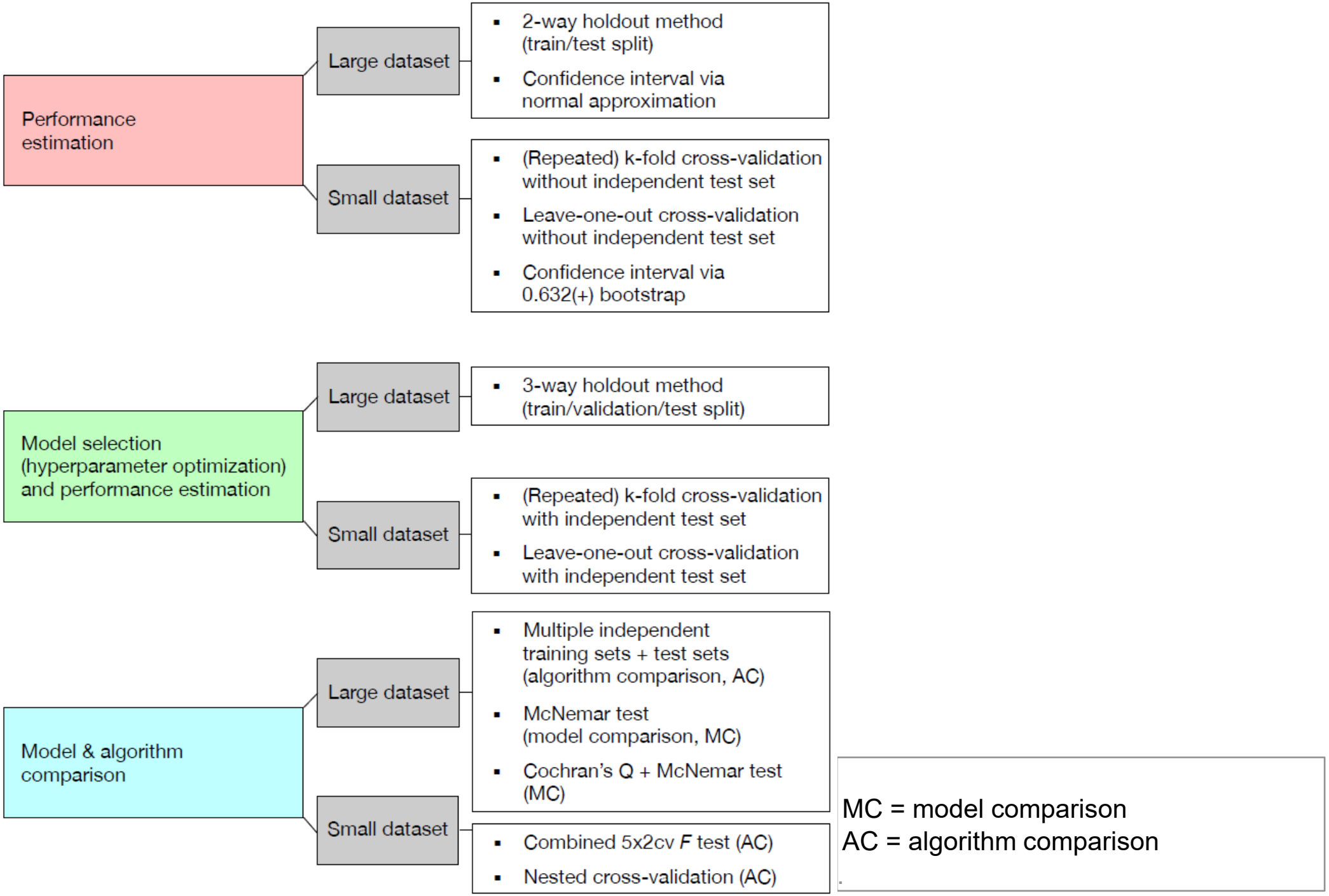


Figure 1: Grid and random search of nine trials for optimizing a function $f(x,y) = g(x) + h(y) \approx g(x)$ with low effective dimensionality. Above each square $g(x)$ is shown in green, and left of each square $h(y)$ is shown in yellow. With grid search, nine trials only test $g(x)$ in three distinct places. With random search, all nine trials explore distinct values of g. This failure of grid search is the rule rather than the exception in high dimensional hyper-parameter optimization.

Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *The Journal of Machine Learning Research*, *13*(1), 281-305.


Performance estimation
Large dataset
2-way holdout method (train/test split)
Confidence interval via normal approximation
Small dataset
(Repeated) k-fold cross-validation without independent test set
Leave-one-out cross-validation without independent test set
Confidence interval via 0.632(+) bootstrap
Model selection (hyperparameter optimization) and performance estimation
Large dataset
3-way holdout method (train/validation/test split)
Small dataset
(Repeated) k-fold cross-validation with independent test set
Leave-one-out cross-validation with independent test set
Model & algorithm comparison
Large dataset
Multiple independent training sets + test sets (algorithm comparison, AC)
McNemar test (model comparison, MC)
Cochran's Q + McNemar test (MC)
Small dataset
Combined 5x2cv F test (AC)
Nested cross-validation (AC)
MC = model comparison
AC = algorithm comparison

EM 538-001: Practical Machine Learning for Enginering Analystics (Spring 2025)
Instructor: Fred Livingston (fjliving@ncsu.edu)

Load and Prepare Datasets

```python
from sklearn.model_selection import train_test_split
import pandas as pd

df_iris = pd.read_csv('iris.csv')

X = df_iris[['PetalLength[cm]', 'PetalWidth[cm]']]
y = df_iris['Species']

X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.33,
                                                    random_state=123,
                                                    shuffle=True, stratify=y)
```

[ ] Python

```python
X_train.shape
```

[ ] Python

```python
X_test.shape
```

[ ] Python

ACCURACY SCORE

3.4.4.2. Accuracy score

The `accuracy_score` function computes the accuracy, either the fraction (default) or the count (normalize=False) of correct predictions.

In multilabel classification, the function returns the subset accuracy. If the entire set of predicted labels for a sample strictly match with the true set of labels, then the subset accuracy is 1.0; otherwise it is 0.0.

If $\hat{y}_i$ is the predicted value of the i-th sample and y_i is the corresponding true value, then the fraction of correct predictions over n_{samples} is defined as

$$\texttt{accuracy}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} 1(\hat{y}_i = y_i)$$

https://scikit-learn.org/stable/modules/model_evaluation.html#scoring-callable

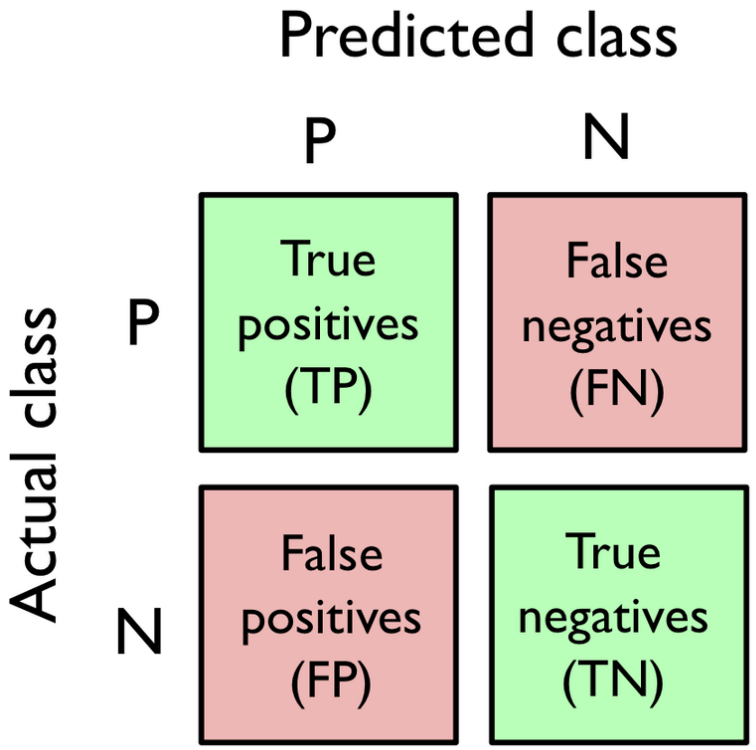
Make Pipeline prediction using Test set

```
pipe.fit(X_train, y_train)
y_pred = pipe.predict(X_test)
y_pred
```

[11] ✓ 0.0s Open 'y_pred' in Data Wrangler Python

```
array(['Iris-setosa', 'Iris-virginica', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-virginica',
       'Iris-virginica', 'Iris-setosa', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-setosa', 'Iris-setosa', 'Iris-virginica', 'Iris-virginica',
       'Iris-setosa', 'Iris-setosa', 'Iris-versicolor', 'Iris-setosa',
       'Iris-setosa', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-versicolor', 'Iris-setosa',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-virginica', 'Iris-setosa', 'Iris-versicolor',
       'Iris-virginica', 'Iris-virginica', 'Iris-versicolor',
       'Iris-virginica', 'Iris-setosa', 'Iris-virginica',
       'Iris-virginica', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
```

2X2 CONFUSION



$$ERR = \frac{FP + FN}{FP + FN + TP + TN} = 1 - ACC$$

$$ACC = \frac{TP + TN}{FP + FN + TP + TN} = 1 - ERR$$

COMPUTE CONFUSION MATRIX

```
pipe.fit(X_train, y_train)
pipe.predict(X_test)
```

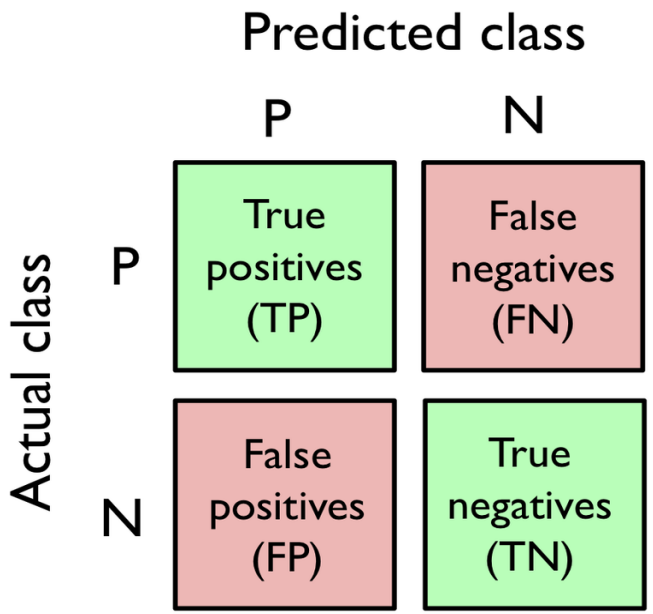
[3] ✓ 0.0s Python

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

```
y_test
```

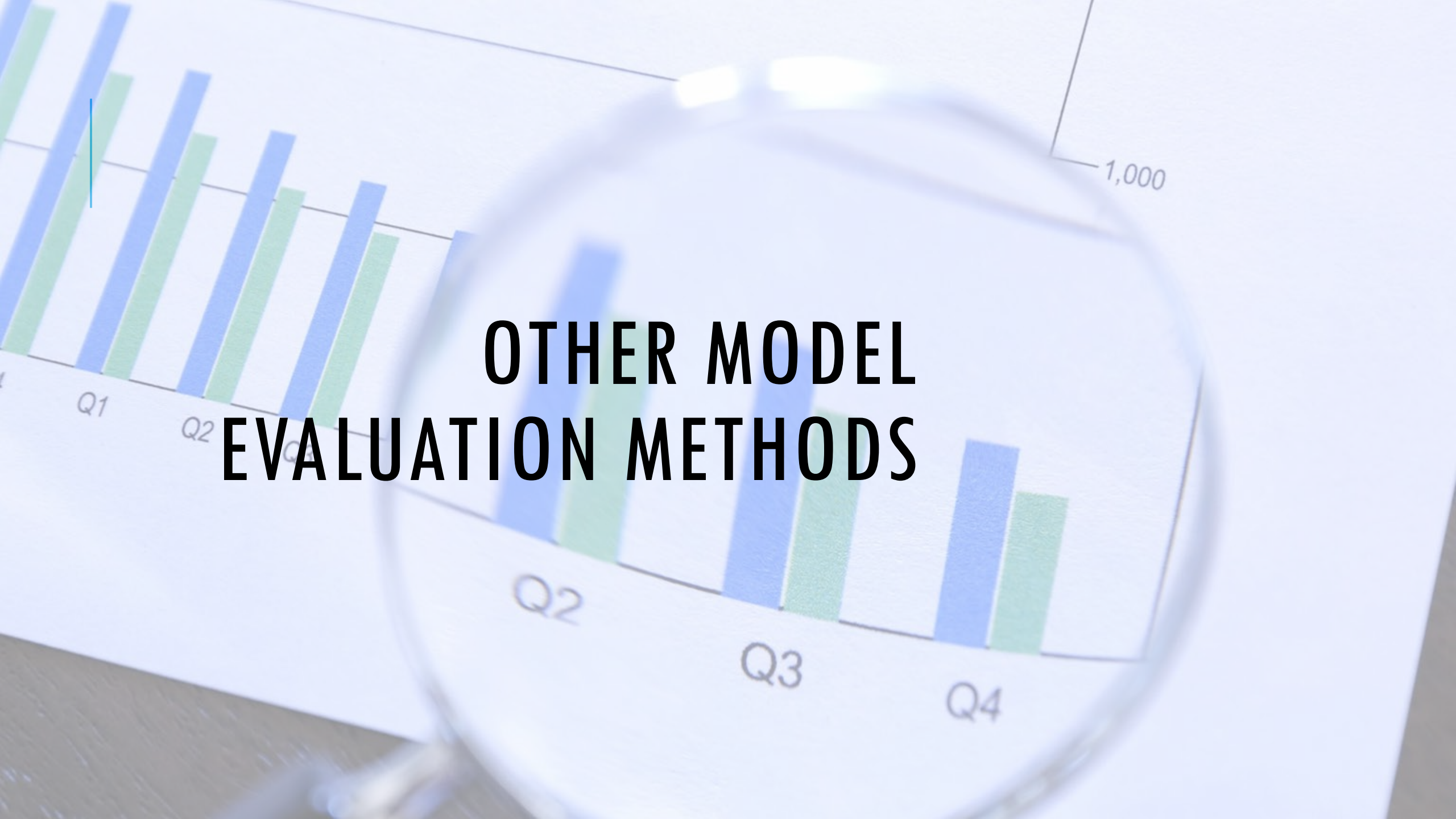
[5] ✓ 0.0s Python

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



$$ERR = \frac{FP + FN}{FP + FN + TP + TN} = 1 - ACC$$

$$ACC = \frac{TP + TN}{FP + FN + TP + TN} = 1 - ERR$$



OTHER MODEL EVALUATION METHODS

False Positive Rate and False Negative Rate

$$\text{TPR}^* = \frac{\text{TP}}{\text{P}} = \frac{\text{TP}}{\text{TP} + \text{FN}} = 1 - \text{FNR}$$

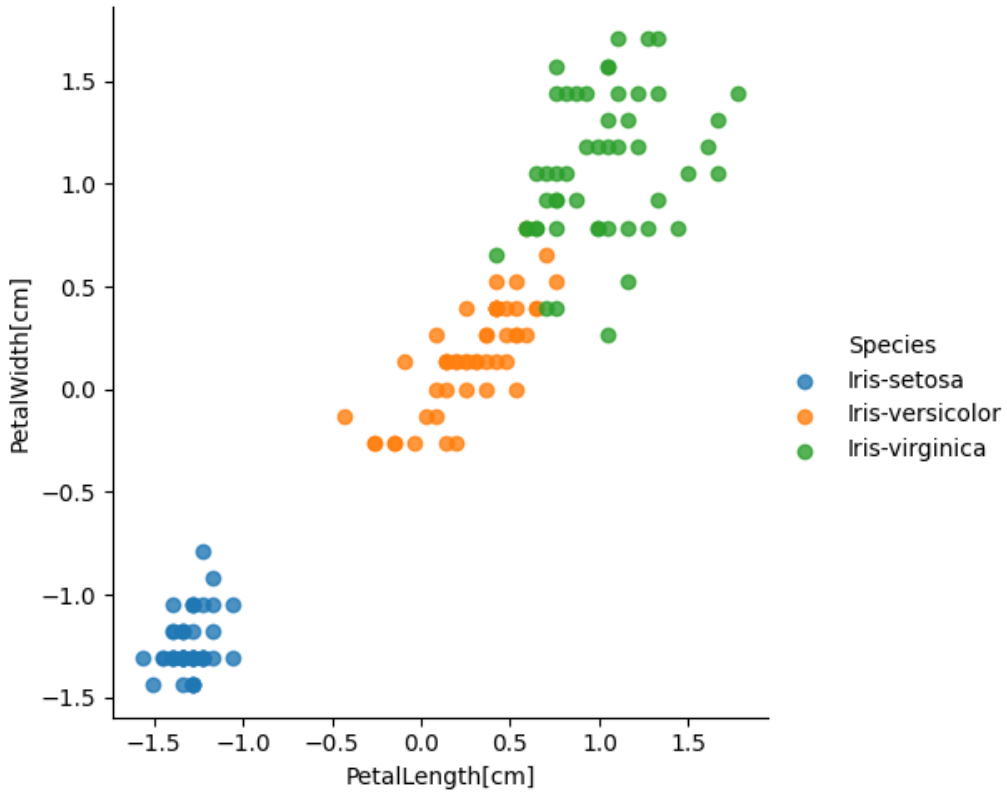
* Relevant later for ROC

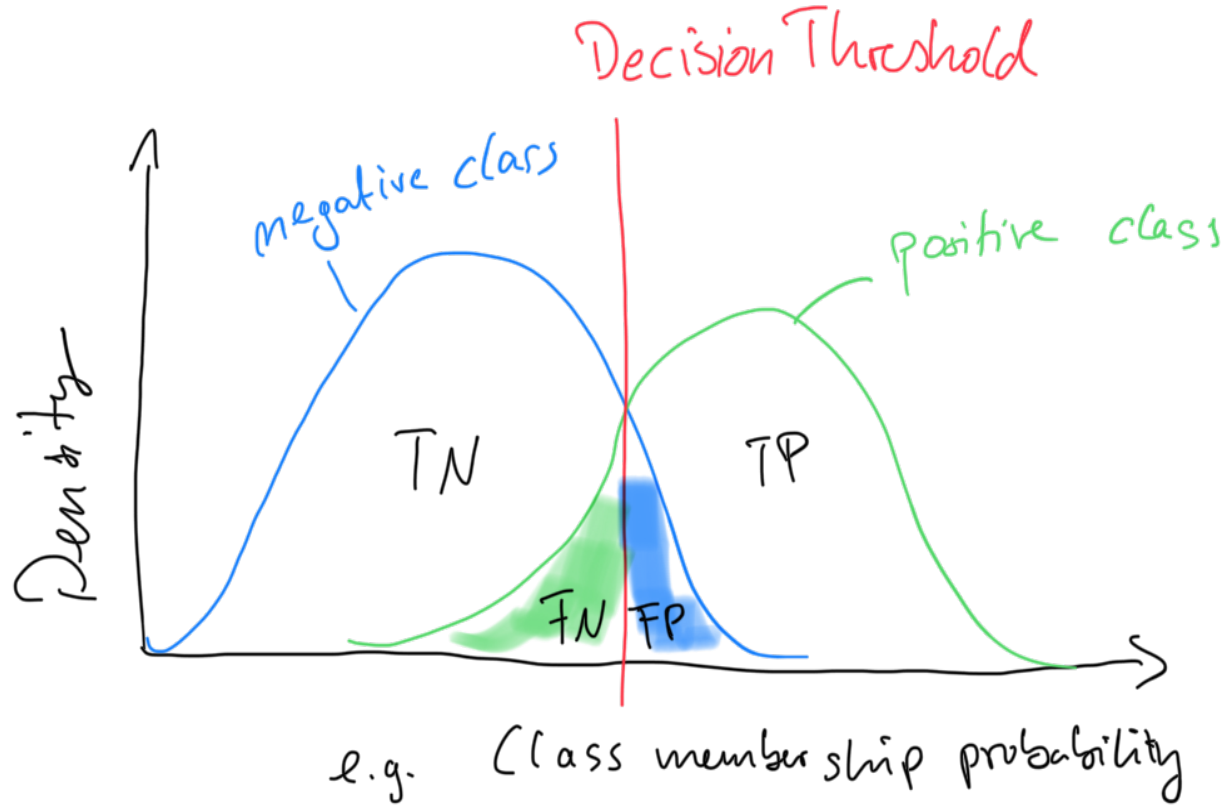
$$\text{FPR}^* = \frac{\text{FP}}{\text{N}} = \frac{\text{FP}}{\text{FP} + \text{TN}} = 1 - \text{TNR}$$

$$\text{FNR} = \frac{\text{FN}}{\text{P}} = \frac{\text{FN}}{\text{FN} + \text{TP}} = 1 - \text{TPR}$$

$$\text{TNR} = \frac{\text{TN}}{\text{N}} = \frac{\text{TN}}{\text{TN} + \text{FP}} = 1 - \text{FPR}$$

Think of it in a spam classification problem (what are true positives, and if you had to pick one at the expense of the other: would you rather decrease the FPR or increase the TPR)


PetalWidth[cm]
PetalLength[cm]
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
Species
Iris-setosa
Iris-versicolor
Iris-virginica


Decision Threshold
negative class
positive class
TN
TP
FN
FP
Density
e.g. Class membership probability

PRECISION, RECALL, AND F1 SCORE

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

$$REC = TPR = \frac{TP}{P} = \frac{TP}{FN + TP}$$

$$F_1 = 2 \cdot \frac{PRE \cdot REC}{PRE + REC}$$

- Terms that are more popular in Information Technology
- Recall is actually just another term for True Positive Rate (or "sensitivity")

OTHERS: MATTHEW'S CORRELATION COEFFICIENT

- Matthews correlation coefficient (MCC) was first formulated by Brian W. Matthews [1] in 1975 to assess the performance of protein secondary structure predictions
- The MCC can be understood as a specific case of a linear correlation coefficient (Pearson r) for a binary classification setting
- Considered as especially useful in unbalanced class settings
- The previous metrics take values in the range between 0 (worst) and 1 (best)
- The MCC is bounded between the range 1 (perfect correlation between ground truth and predicted outcome) and -1 (inverse or negative correlation) — a value of 0 denotes a random prediction.

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \qquad (10)$$

[1] Brian W Matthews. Comparison of the predicted and observed secondary structure of T4 phage lysozyme. Biochimica et Biophysica Acta (BBA)- Protein Structure, 405(2):442–451, 1975.

CONFUSION MATRIX FOR MULTI-CLASS SETTINGS

$$ACC = \frac{T}{n}$$

Predicted Labels

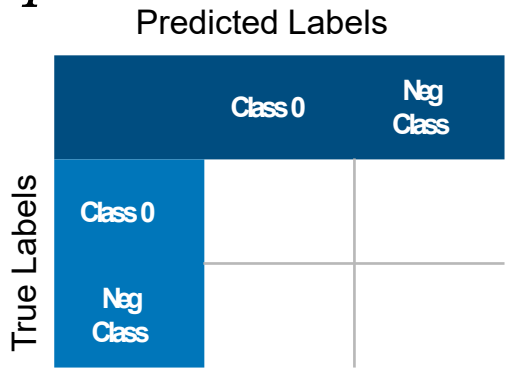
True Labels

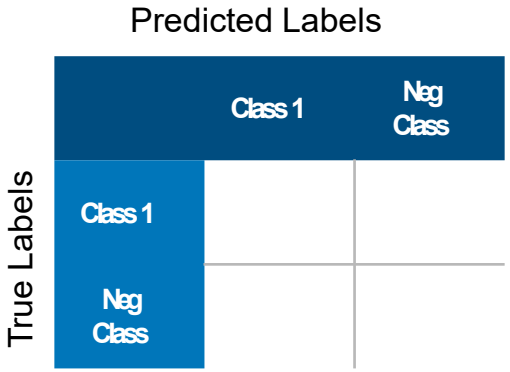
	Class 0	Class 1	Class 2
Class 0	T(0,0)		
Class 1		T(1,1)	
Class 2			T(2,2)

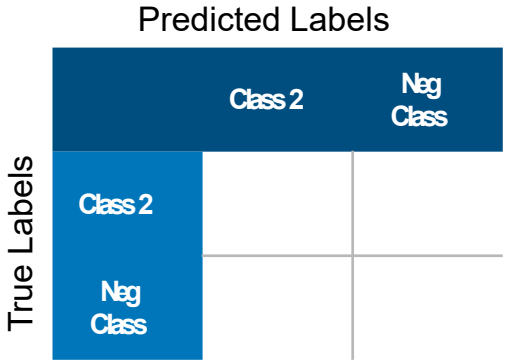
Confusions readily matrices are traditionally for binary class problems but we can be generalized it to multi-class settings

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

$$ACC = \frac{T}{n}$$

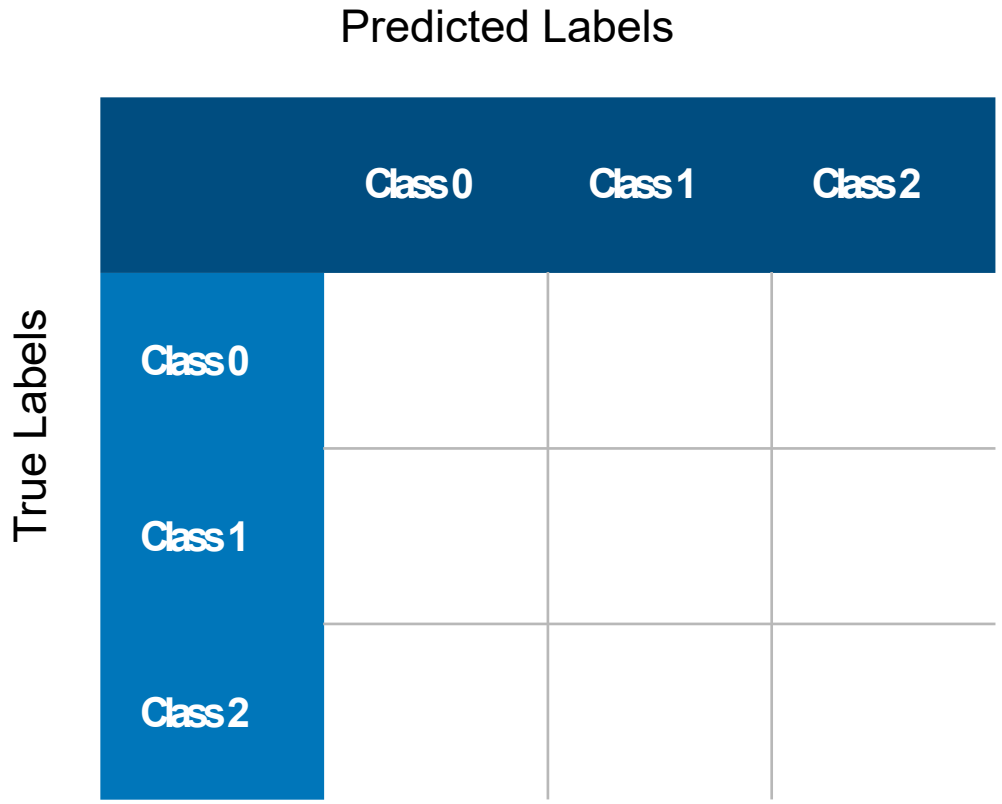







```
[3]  ✓ 0.0s
...  array([1, 0, 2, 2, 0, 0, 2, 2, 2, 0, 0, 2, 2, 1, 2, 1, 0, 0, 0, 0, 0, 2,
            2, 1, 2, 2, 1, 1, 1, 1])

     y_test
[5]  ✓ 0.0s
...  array([1, 0, 2, 2, 0, 0, 2, 1, 2, 0, 0, 1, 2, 1, 2, 1, 0, 0, 0, 0, 0, 2,
            2, 1, 2, 2, 1, 1, 1, 1])
```



CLASSIFICATION METRICS

Scoring string name	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`	
'd2_log_loss_score'	`metrics.d2_log_loss_score`	

https://scikit-learn.org/stable/modules/model_evaluation.html#scoring-callable

REGRESSION METRICS HW1

Regression		
'explained_variance'	metrics.explained_variance_score	
'neg_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.root_mean_squared_error	
'neg_mean_squared_log_error'	metrics.mean_squared_log_error	
'neg_root_mean_squared_log_error'	metrics.root_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	

https://scikit-learn.org/stable/modules/model_evaluation.html#scoring-callable

CONFUSION MATRIX FOR KNN

0-knn_accuracy.ipynb | 1-knn_metrics.ipynb

G: > My Drive > Academics > Machine Learning > Spring 2025 > Lectures > Lecture 007 > lecture7_code_examples > 1-knn_metrics.ipynb > ...

Generate | Code | Markdown | Run All | Restart | Clear All Outputs | View data | Jupyter Variables | Outline | ... | pyml (Python 3.11.9)

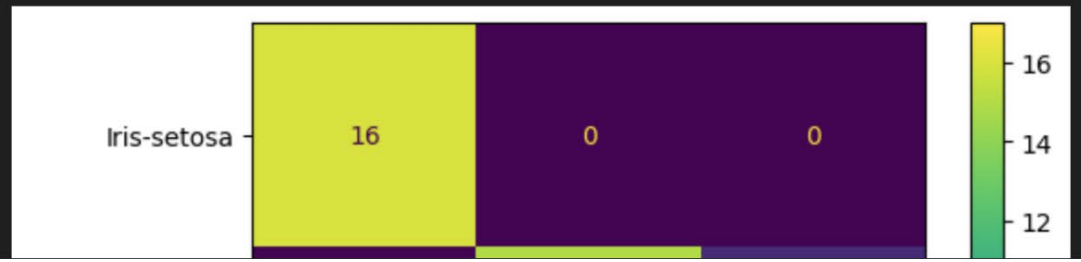
```
from sklearn.metrics import confusion_matrix
confmat = confusion_matrix(y_test, y_pred)
print(confmat)
```

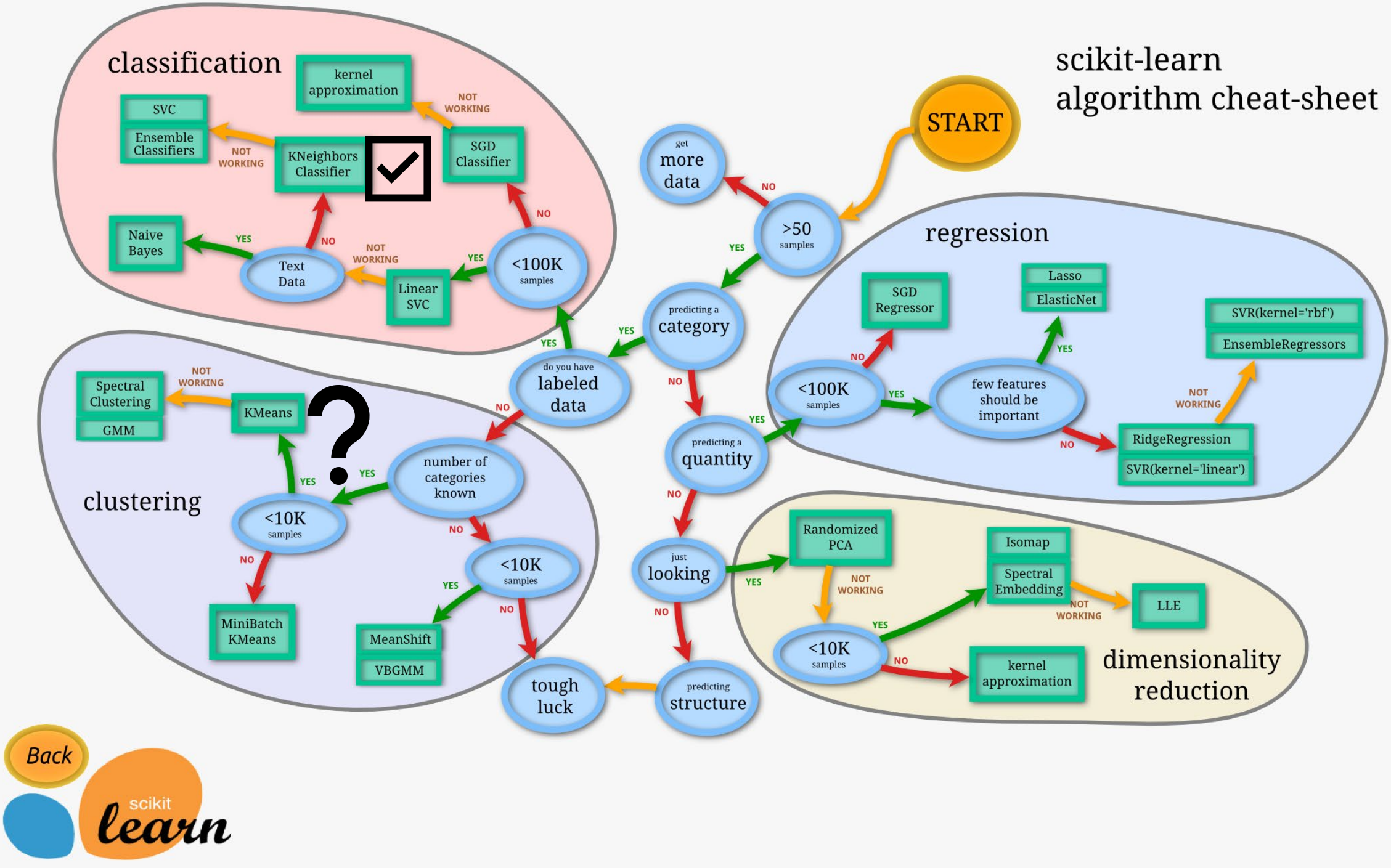
[5] ✓ 0.0s Open 'confmat' in Data Wrangler Python

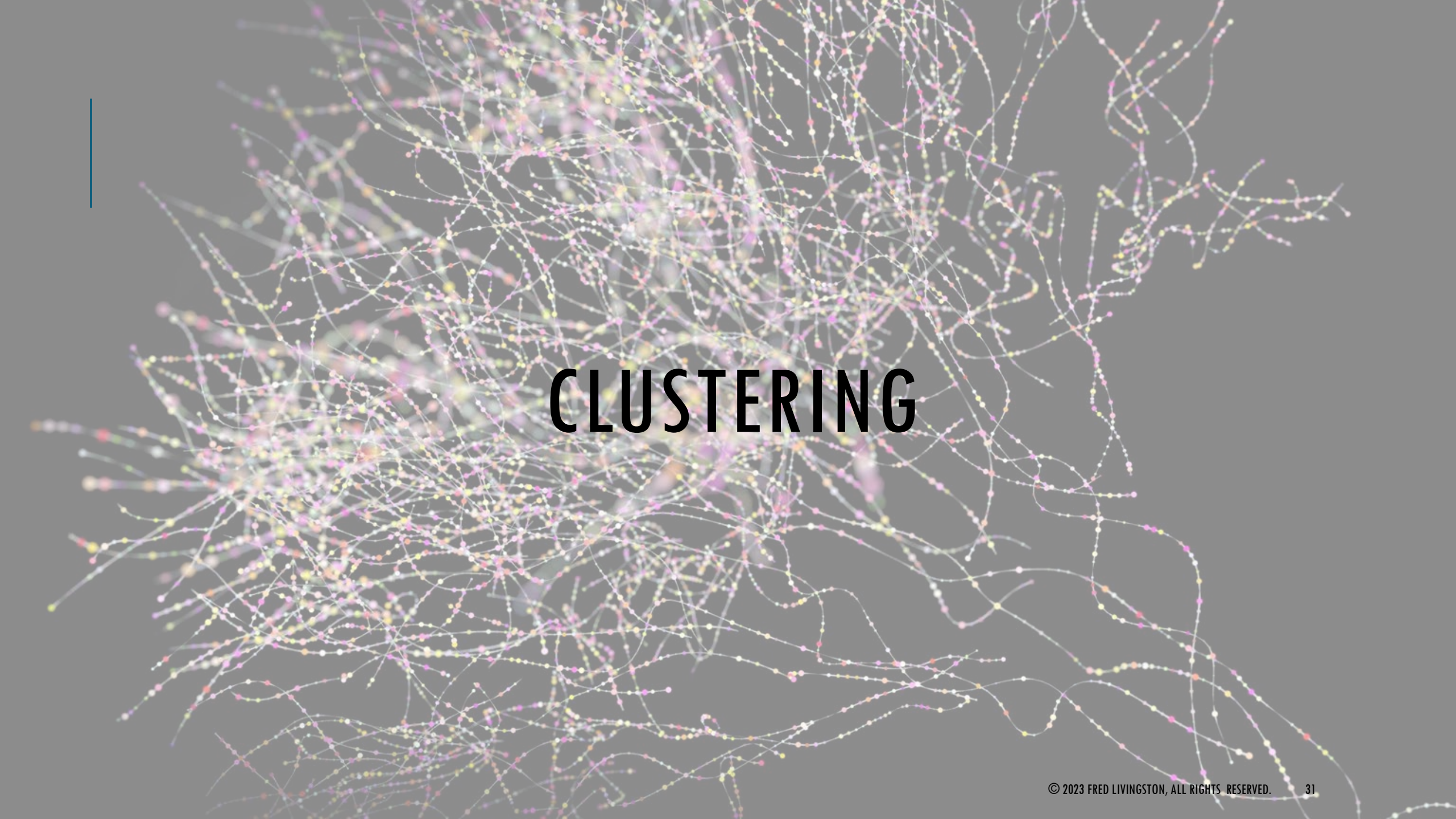
```
[[16  0  0]
 [ 0 15  2]
 [ 0  0 17]]
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import ConfusionMatrixDisplay
disp = ConfusionMatrixDisplay(confusion_matrix=confmat,
                              display_labels=pipe.classes_)
disp.plot()
plt.show()
```

[6] ✓ 1.0s Python




scikit-learn
algorithm cheat-sheet
START
get more data
NO
>50 samples
YES
predicting a category
YES
NO
do you have labeled data
YES
NO
<100K samples
YES
NO
Linear SVC
NOT WORKING
Text Data
YES
NO
Naive Bayes
YES
NO
KNeighbors Classifier
NOT WORKING
SVC
Ensemble Classifiers
NOT WORKING
kernel approximation
NOT WORKING
SGD Classifier
classification
predicting a quantity
YES
NO
<100K samples
NO
YES
SGD Regressor
Lasso
ElasticNet
few features should be important
YES
NO
RidgeRegression
SVR(kernel='linear')
NOT WORKING
SVR(kernel='rbf')
EnsembleRegressors
regression
number of categories known
YES
NO
<10K samples
YES
NO
KMeans
NOT WORKING
Spectral Clustering
GMM
MiniBatch KMeans
<10K samples
YES
NO
MeanShift
VBGMM
clustering
just looking
YES
NO
Randomized PCA
NOT WORKING
Isomap
Spectral Embedding
NOT WORKING
LLE
<10K samples
YES
NO
kernel approximation
dimensionality reduction
predicting structure
tough luck
Back
scikit learn

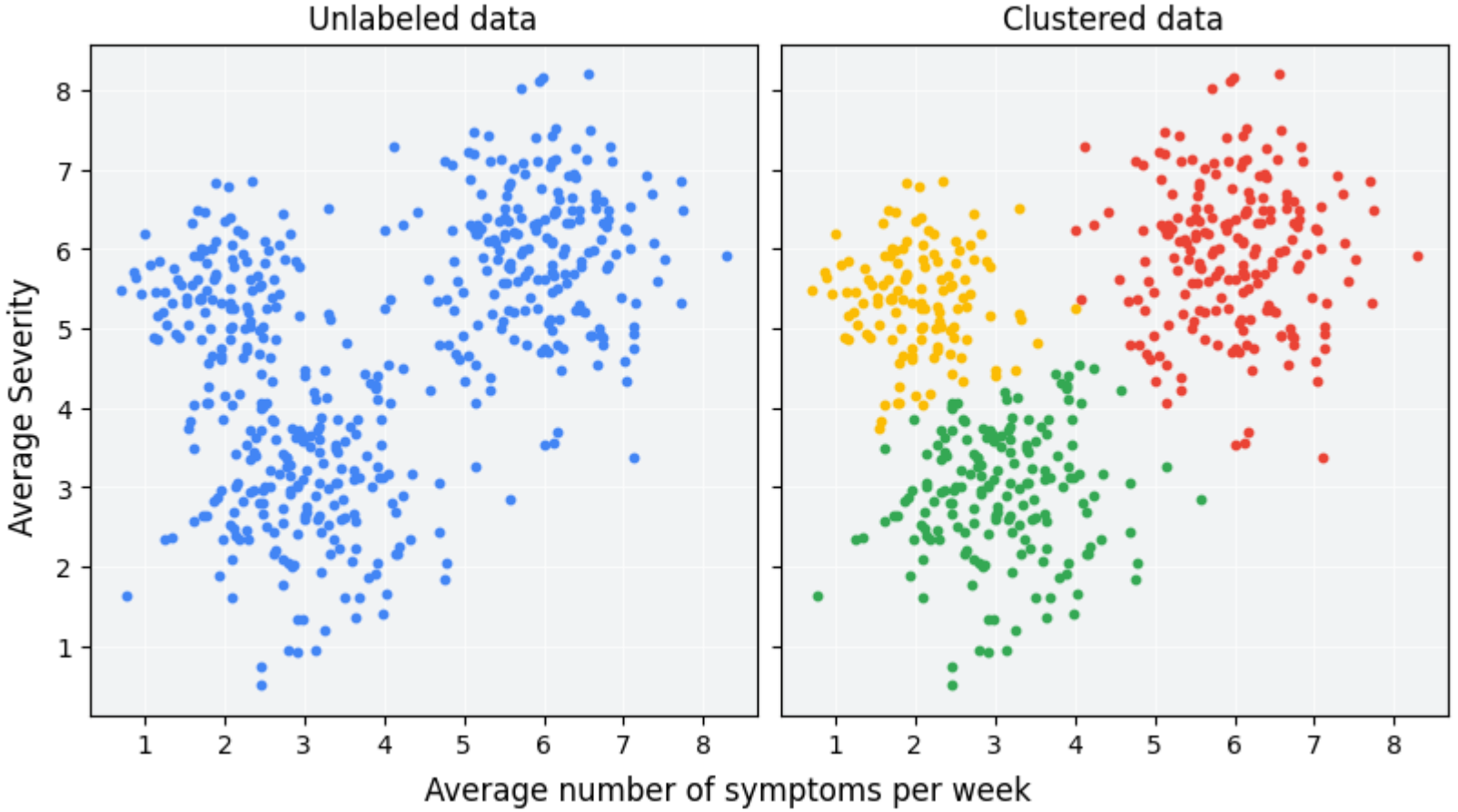


CLUSTERING

WHAT IS CLUSTERING

Clustering is an unsupervised machine learning technique designed to group **unlabeled examples** based on their similarity to each other.

WHAT IS CLUSTERING



After clustering, each group is assigned a unique label called a **cluster ID**. Clustering is powerful because it can simplify large, complex datasets with many features to a single cluster ID.

CLUSTERING USE CASES

Clustering is useful in a variety of industries. Some common applications for clustering:

- Market segmentation
- Social network analysis
- Search result grouping
- Medical imaging
- Image segmentation
- Anomaly detection

IMPUTATION

When some examples in a cluster have missing feature data, you can infer the missing data from other examples in the cluster. This is called imputation. For example, less popular videos can be clustered with more popular videos to improve video recommendations.

DATA COMPRESSION

As discussed, the relevant cluster ID can replace other features for all examples in that cluster. This substitution reduces the number of features and therefore also reduces the resources needed to store, process, and train models on that data. For very large datasets, these savings become significant.

To give an example, a single YouTube video can have feature data including:

- viewer location, time, and demographics
- comment timestamps, text, and user IDs
- video tags

Clustering YouTube videos replaces this set of features with a single cluster ID, thus compressing the data.

PRIVACY PRESERVATION

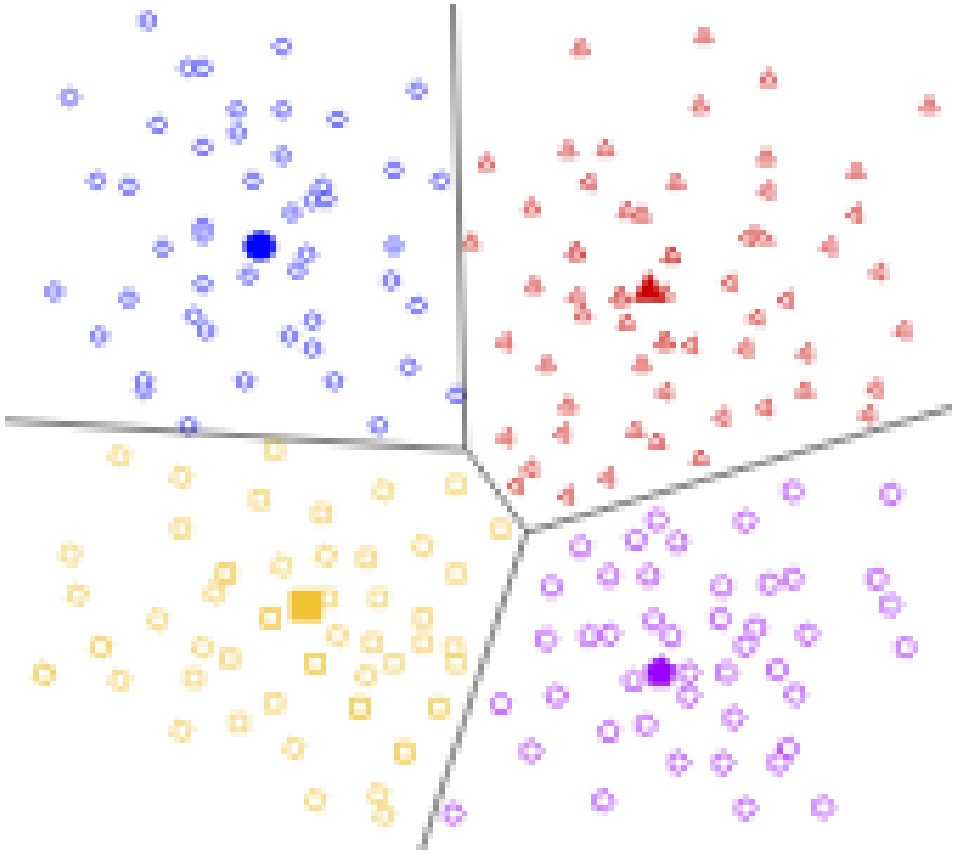
You can preserve privacy somewhat by clustering users and associating user data with cluster IDs instead of user IDs. To give one possible example, say you want to train a model on YouTube users' watch history. Instead of passing user IDs to the model, you could cluster users and pass only the cluster ID. This keeps individual watch histories from being attached to individual users. Note that the cluster must contain a sufficiently large number of users in order to preserve privacy.

CLUSTERING ALGORITHMS

Machine learning datasets can have millions of examples, but not all clustering algorithms scale efficiently. Many clustering algorithms compute the similarity between all pairs of examples, which means their runtime increases as the square of the number of examples n, denoted as O(n2) in complexity notation. O(n2) algorithms are not practical for datasets with millions of examples.

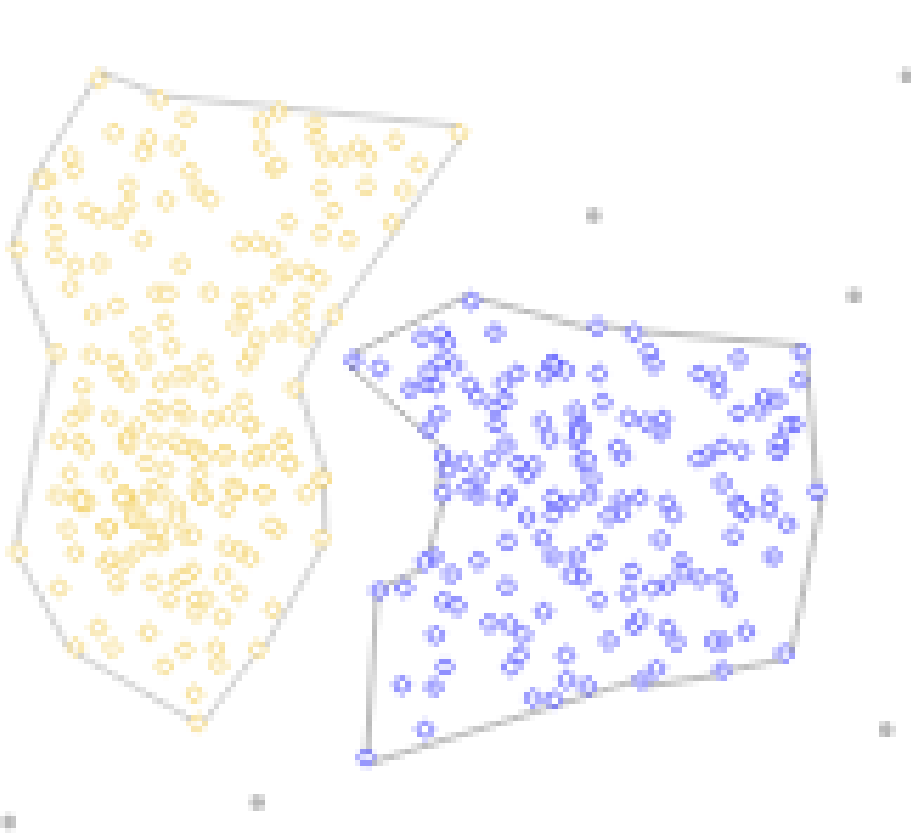
CENTROID-BASED CLUSTERING

The centroid of a cluster is the arithmetic mean of all the points in the cluster. **Centroid-based clustering** organizes the data into non-hierarchical clusters. Centroid-based clustering algorithms are efficient but sensitive to initial conditions and outliers. Of these, k-means is the most widely used. It requires users to define the number of centroids, *k*, and works well with clusters of roughly equal size.



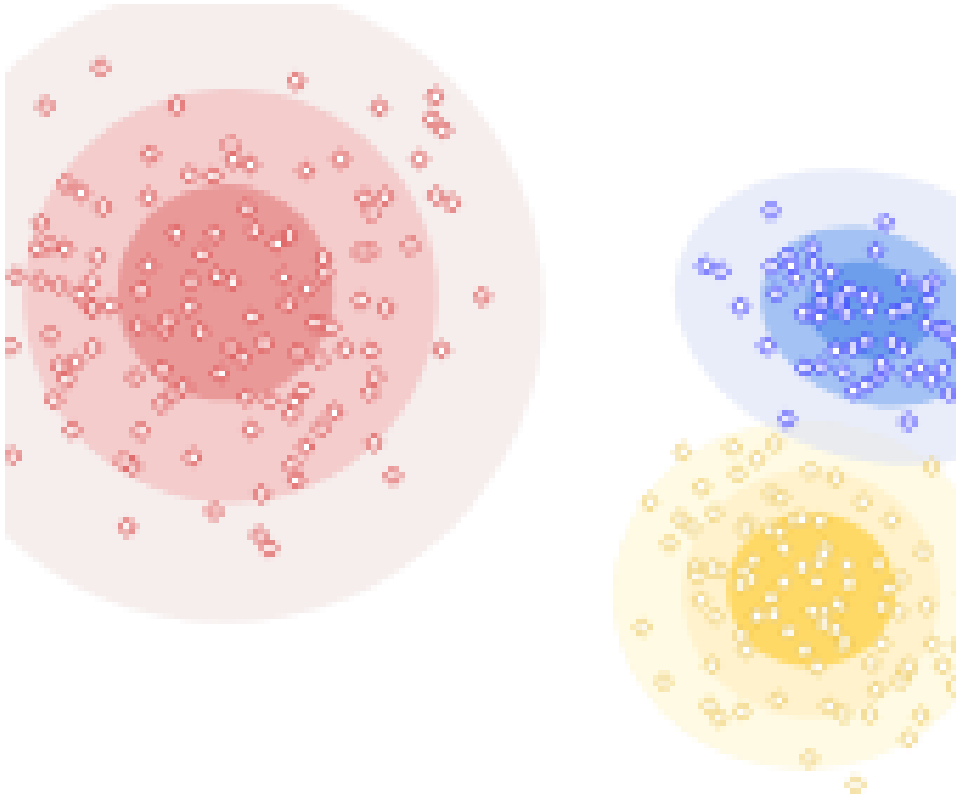
DENSITY-BASED CLUSTERING

Density-based clustering connects contiguous areas of high example density into clusters. This allows for the discovery of any number of clusters of any shape. Outliers are not assigned to clusters. These algorithms have difficulty with clusters of different density and data with high dimensions.



DISTRIBUTION-BASED CLUSTERING

This clustering approach assumes data is composed of probabilistic distributions, such as **Gaussian distributions**. As distance from the distribution's center increases, the probability that a point belongs to the distribution decreases. The bands show that decrease in probability.



K-MEANS CLUSTERING

As previously mentioned, many clustering algorithms don't scale to the datasets used in machine learning, which often have millions of examples. For example, agglomerative or divisive hierarchical clustering algorithms look at all pairs of points and have complexities of O(n^2log(n)) and O(n^2), respectively.

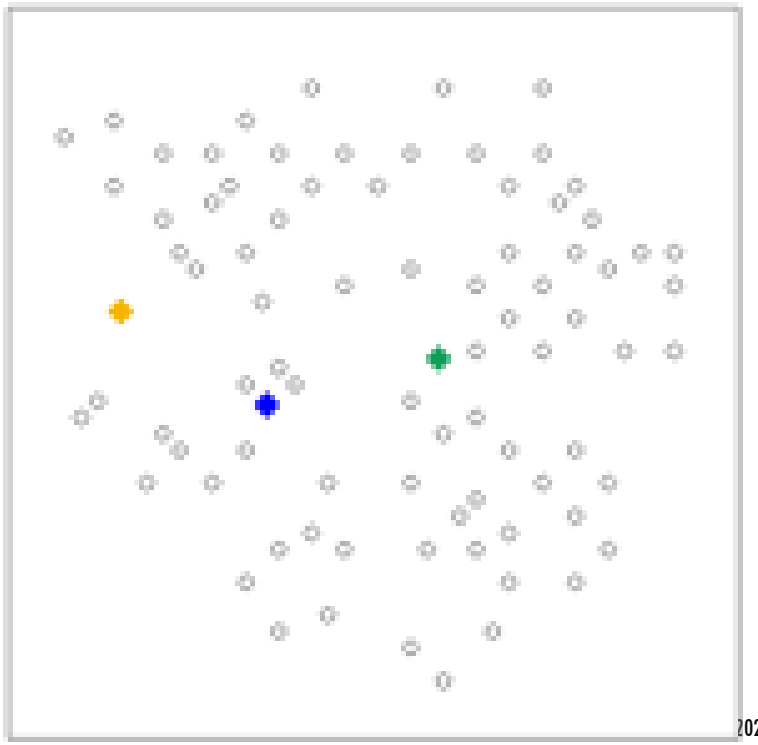
k-means scales as O(nk), where k is the number of clusters chosen by the user. This algorithm groups points into k clusters by minimizing the distances between each point and its cluster's centroid

As a result, k-means effectively treats data as composed of a number of roughly circular distributions, and tries to find clusters corresponding to these distributions. But real-world data contains outliers and density-based clusters and might not match the assumptions underlying k-means.

K-MEANS CLUSTERING ALGORITHM

The algorithm follows these steps:

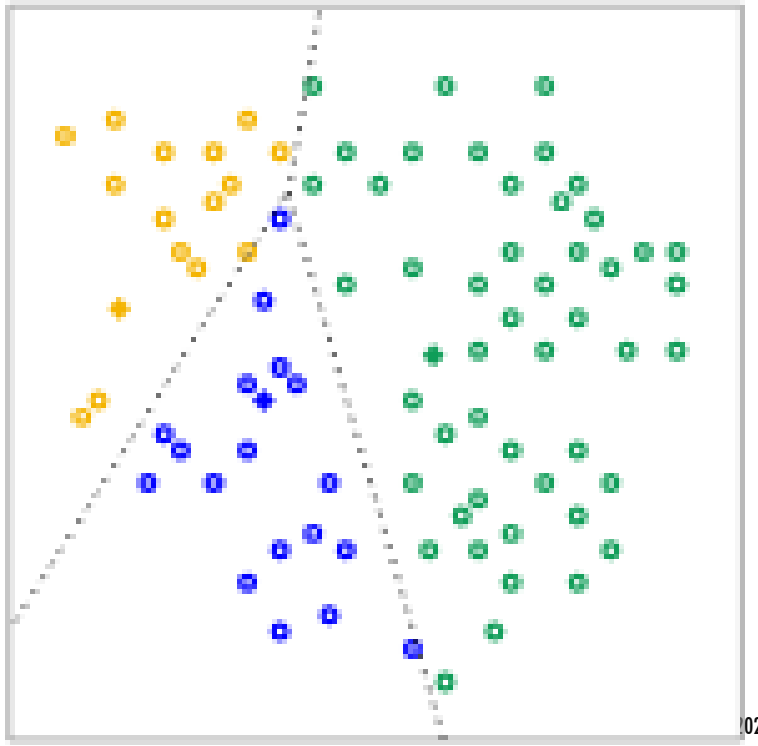
1. Provide an initial guess for **k**, which can be revised later. For this example, we choose **k=3**.
2. Randomly choose **k** centroids.



K-MEANS CLUSTERING ALGORITHM

The algorithm follows these steps:

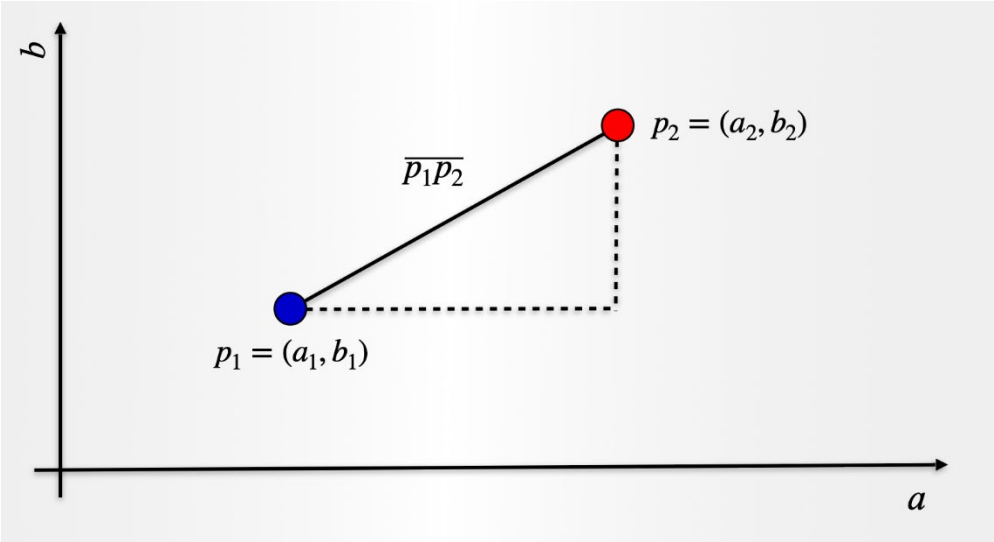
3. Assign each point to the **nearest centroid** to get **k** initial clusters.



MANUAL SIMILARITY MEASURE

As just shown, k-means assigns points to their closest centroid. But what does "closest" mean?

To apply k-means to feature data, you will need to define a measure of similarity that combines all the feature data into a single numeric value, called a **manual similarity measure**.



ROOT MEAN SQUARED ERROR (RMSE) AND SIMILARITY

Combine features by calculating the root mean squared error (RMSE). This rough measure of similarity is given by

$$\sqrt{\frac{(s_i - s_j)^2 + (p_i - p_j)^2}{2}}$$

Similarity = 1 - RMSE

SIMILARITY EXAMPLE

Consider a shoe dataset. If that dataset has shoe size as its only feature, you can define the similarity of two shoes in terms of the difference between their sizes. The smaller the numerical difference between sizes, the greater the similarity between shoes.

If that shoe dataset had two numeric features, size and price, you can combine them into a single number representing similarity. First scale the data so both features are comparable:

- Size (s): Shoe size probably forms a Gaussian distribution. Then normalize the data.
- Price (p): The data is probably a Poisson distribution. If you have enough data, convert the data to quantiles and scale to [0,1].

SIMILARITY EXAMPLE

simple example, calculate similarity for two shoes with US sizes 8 and 11, and prices 120 and 150. Since we don't have enough data to understand the distribution, we'll scale the data without normalizing or using quantiles.

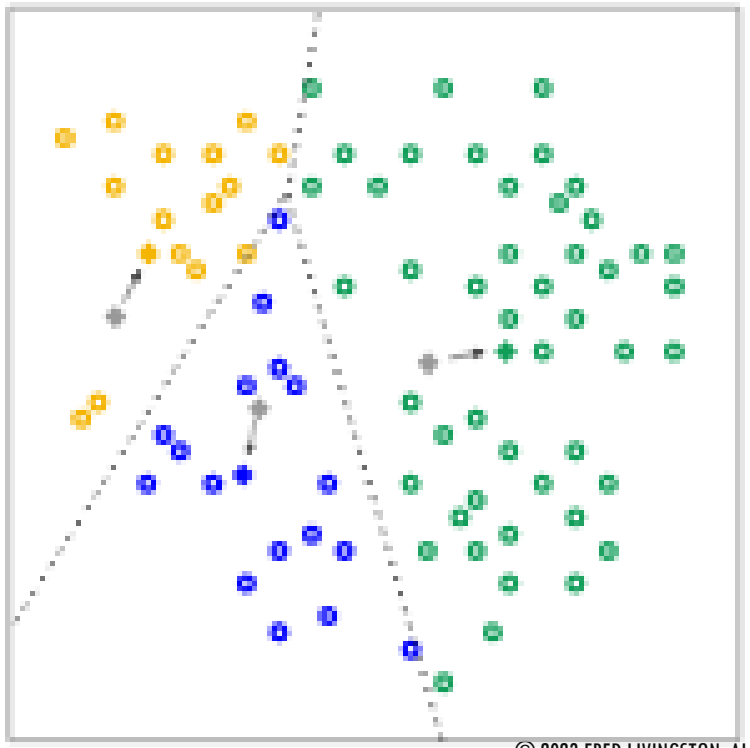
Action	Method
Scale the size.	Assume a maximum possible shoe size of 20. Divide 8 and 11 by the maximum size 20 to get 0.4 and 0.55.
Scale the price.	Divide 120 and 150 by the maximum price 150 to get 0.8 and 1.
Find the difference in size.	$0.55 - 0.4 = 0.15$
Find the difference in price.	$1 - 0.8 = 0.2$
Calculate the RMSE.	$\sqrt{\frac{0.2^2+0.15^2}{2}} = 0.17$

$$\text{Similarity} = 1 - 0.17 = 0.83$$

K-MEANS CLUSTERING ALGORITHM

The algorithm follows these steps:

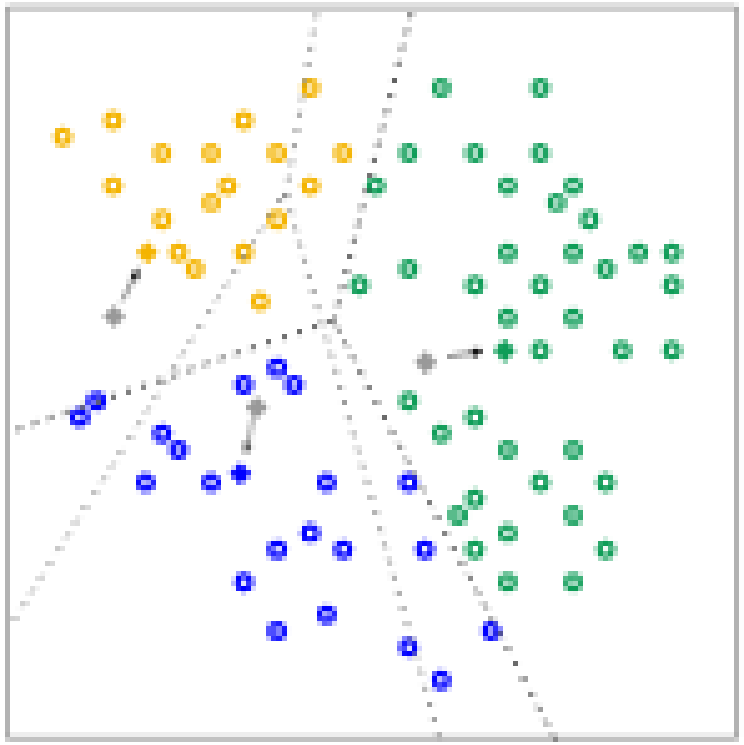
4. For each cluster, calculate a new centroid by taking the mean position of all points in the cluster. The arrows show the change in centroid positions.



K-MEANS CLUSTERING ALGORITHM

The algorithm follows these steps:

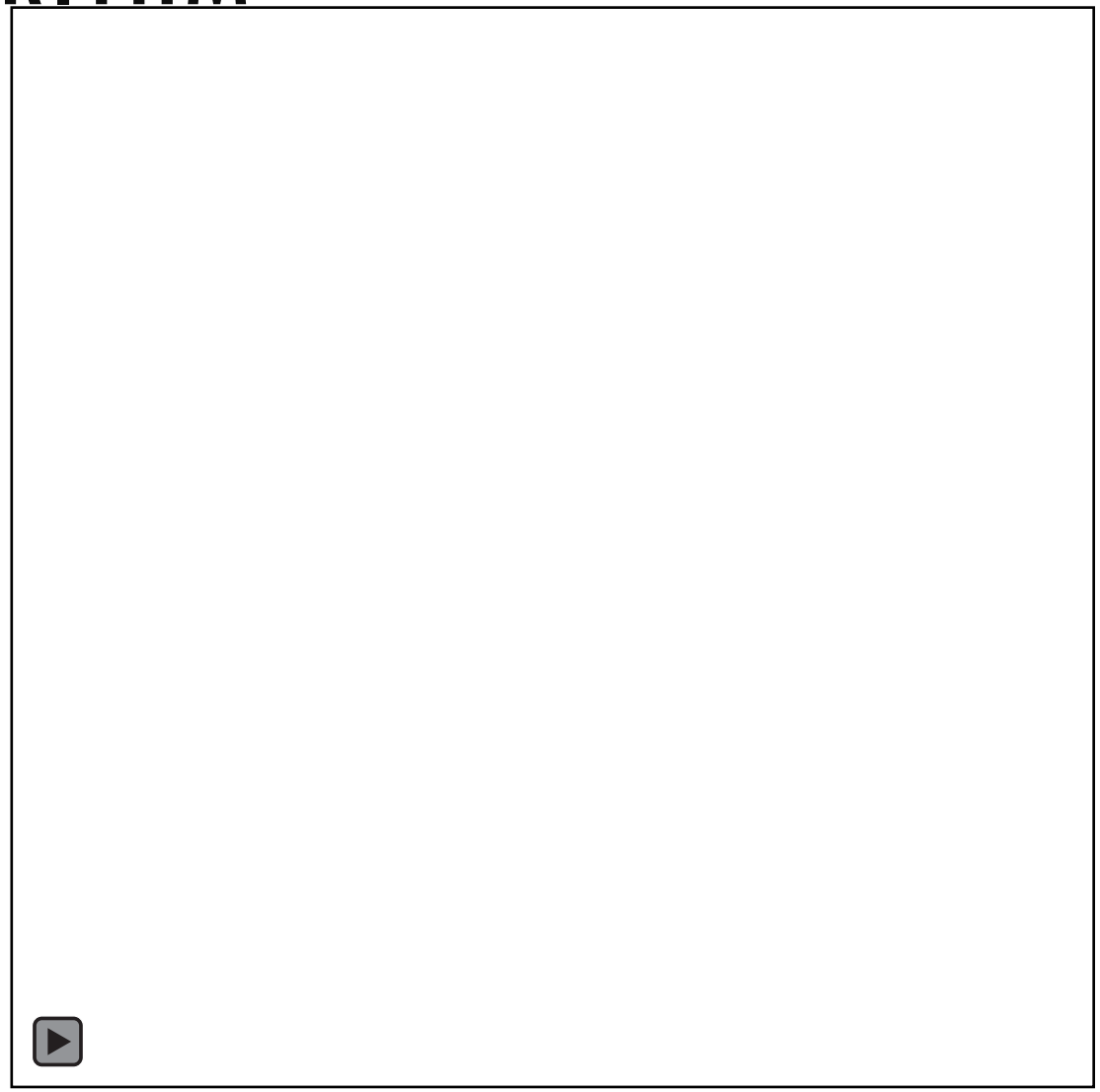
5. Reassign each point to the nearest new centroid.



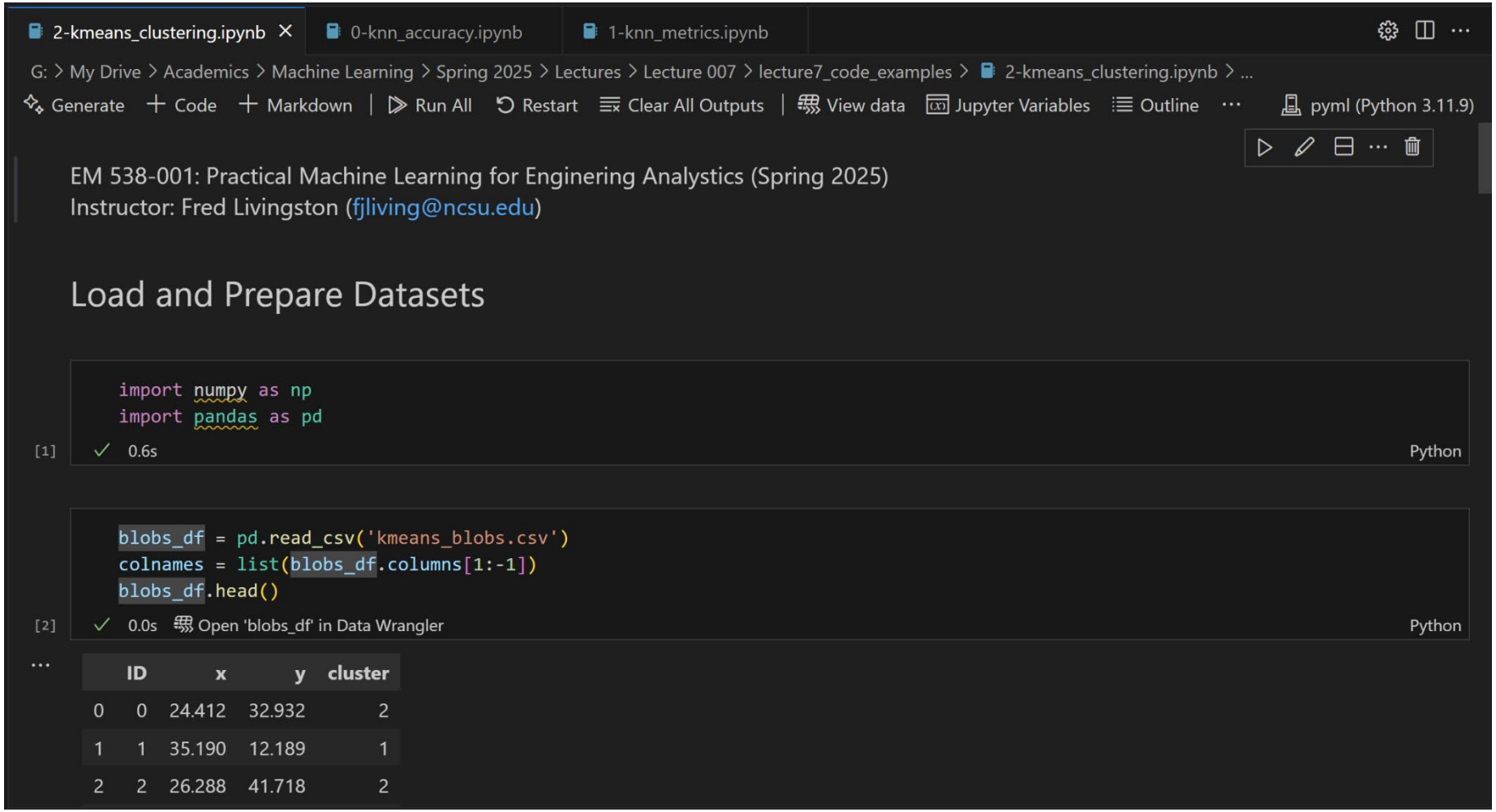
K-MEANS CLUSTERING ALGORITHM

The algorithm follows these steps:

6. Repeat steps 4 and 5, recalculating centroids and cluster membership, until points no longer change clusters. In the case of large datasets, you can stop the algorithm before convergence based on other criteria.



K-MEANS CLUSTERING PYTHON ALGORITHM



	ID	x	y	cluster
0	0	24.412	32.932	2
1	1	35.190	12.189	1
2	2	26.288	41.718	2

Q/A?

