EM 538-001: PRACTICAL MACHINE LEARNING FOR ENGINEERING ANALYTICS

LECTURE 005

Fred Livingston, Ph.D.

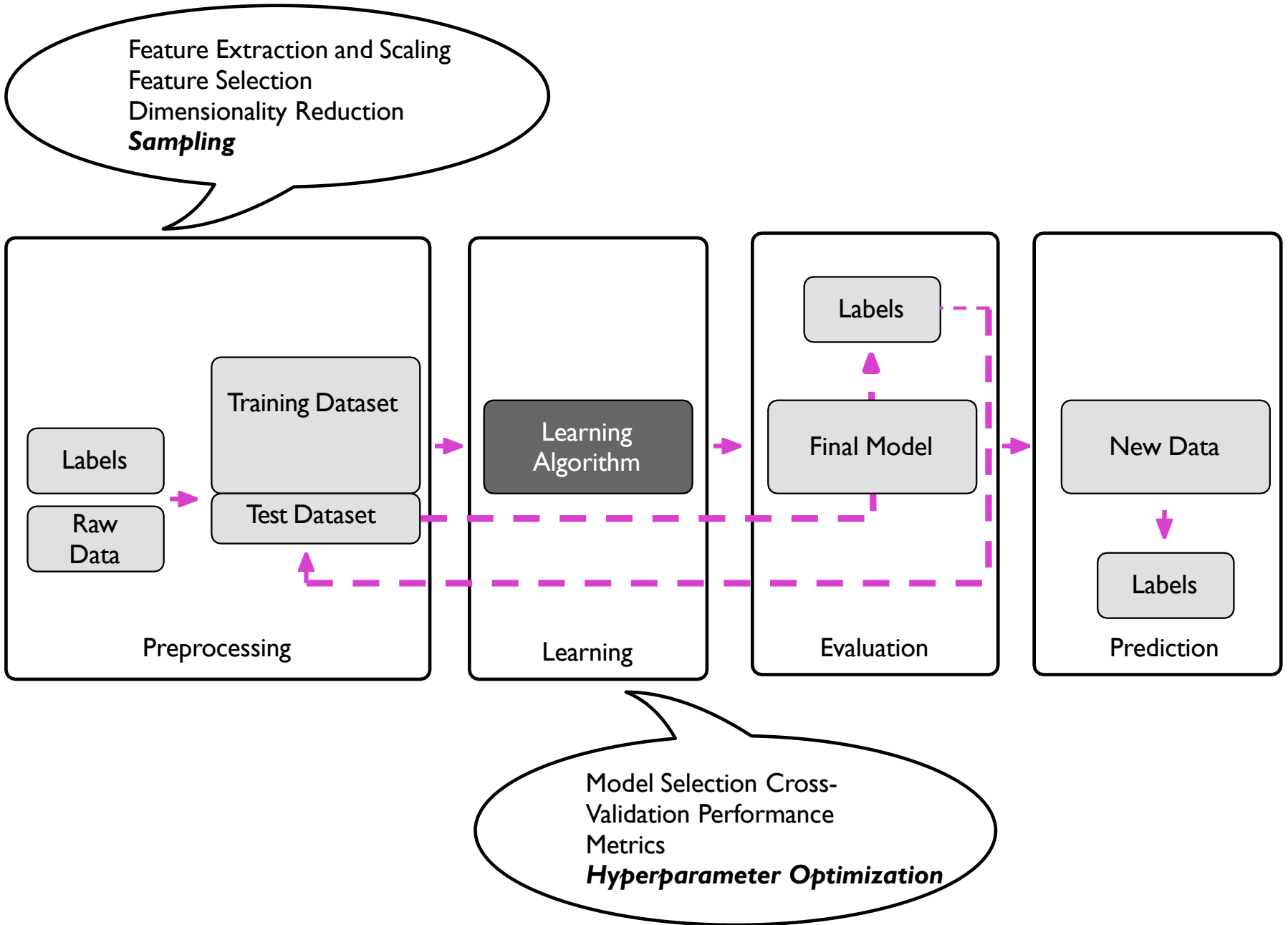
SUPERVISED MACHINE LEARNING

- k-Nearest Neighbors Revisit
- Preprocessing; Sampling
- Learning; Hyperparameter Optimization
- scikit-learn Library
- Homework 1 assigned next Lecture!

5 STEPS FOR APPROACHING A MACHINE LEARNING APPLICATION

1. Define the problem to be solved.
2. Collect (labeled) data.
3. Choose an algorithm class.
4. Choose an optimization metric or measure for learning the model.
5. Choose a metric or measure for evaluating the model.

MACHINE LEARNING WORKFLOW



SUPERVISED LEARNING NOTATION

Training set: $\mathcal{D} = \{\langle \mathbf{x}^{[i]}, y^{[i]} \rangle, i = 1, \ldots, n\}$,

Unknown function: $f(\mathbf{x}) = y$

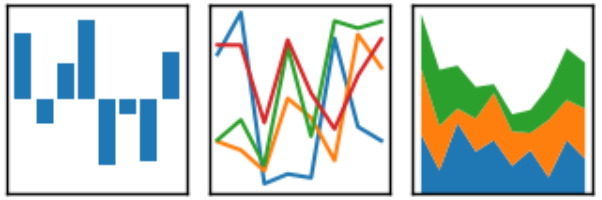
Hypothesis: $h(\mathbf{x}) = y$

DATA REPRESENTATION

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y^{[1]} \\ y^{[2]} \\ \vdots \\ y^{[n]} \end{bmatrix}$$

pandas

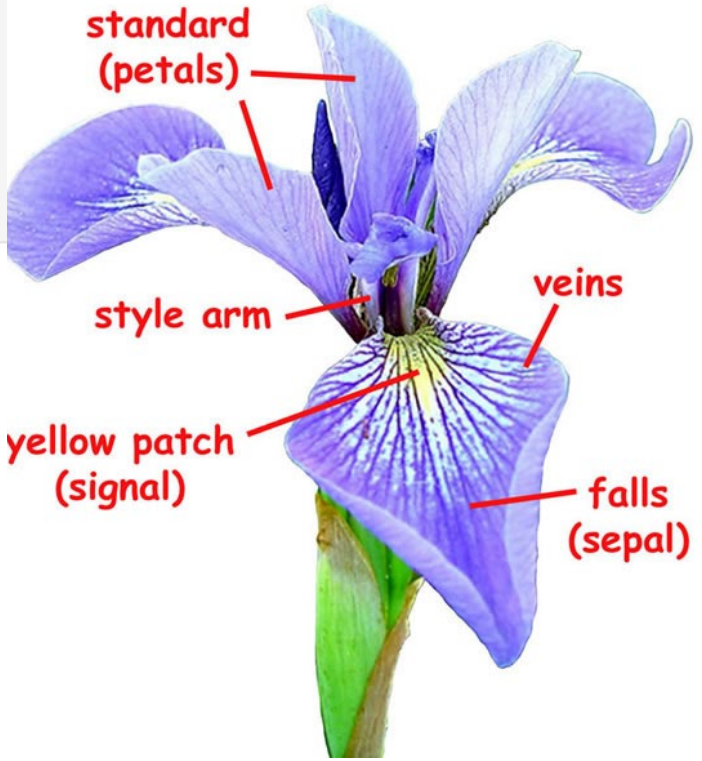
$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$



https://pandas.pydata.org

```
import pandas as pd


df = pd.read_csv('iris.csv')
df.head()
```

	Id	SepalLength[cm]	SepalWidth[cm]	PetalLength[cm]	PetalWidth[cm]	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa



DATA REPRESENTATION

$m = 4$

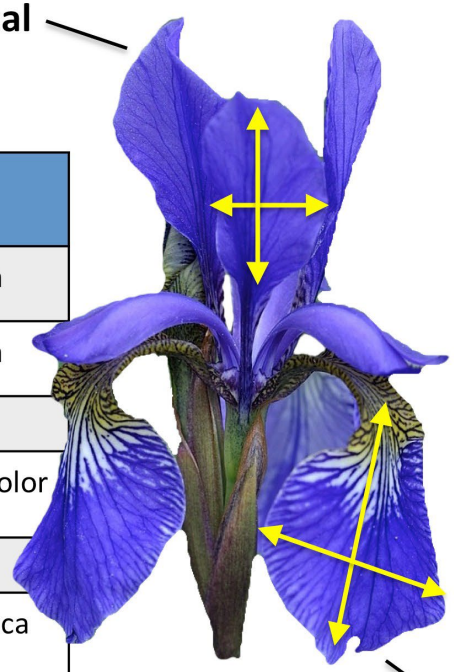
$n = 150$

Samples

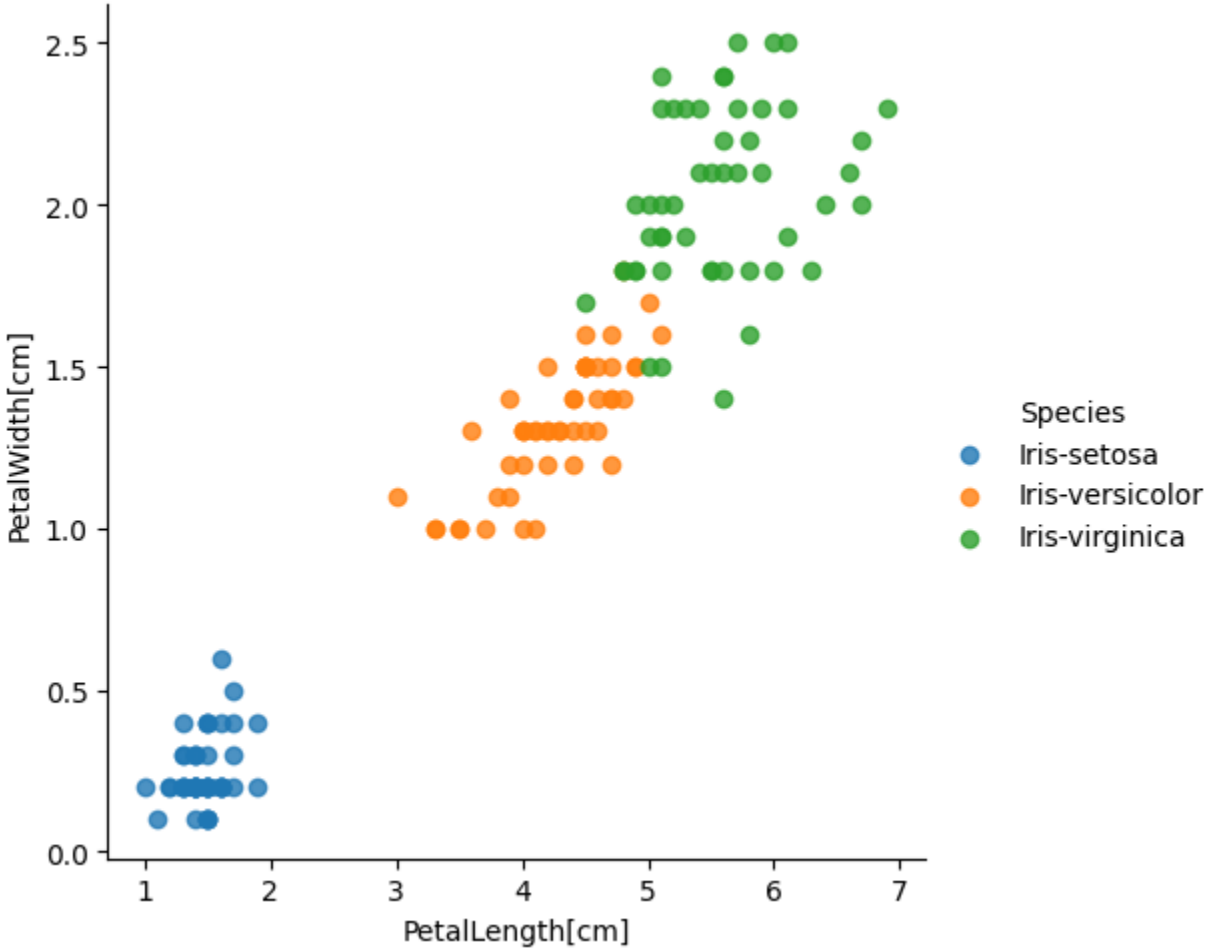
	Sepal length	Sepal width	Petal length	Petal width	
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
...					
50	6.4	3.5	4.5	1.2	Versicolor
...					
150	5.9	3.0	5.0	1.8	Virginica

Features
x

Class labels
(targets)



IRIS DATASET

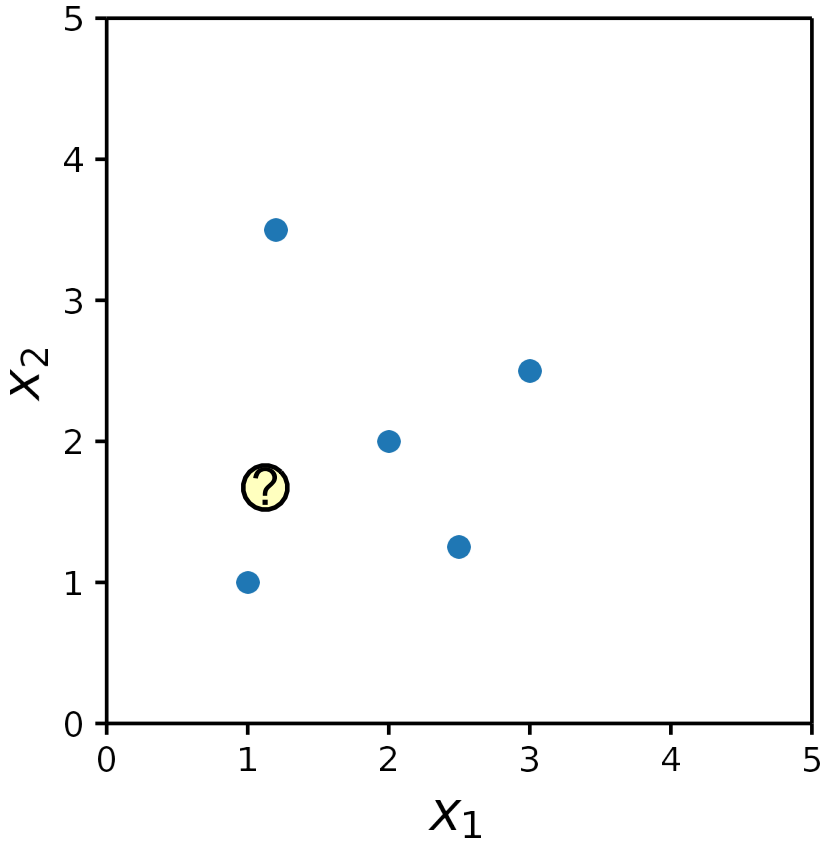


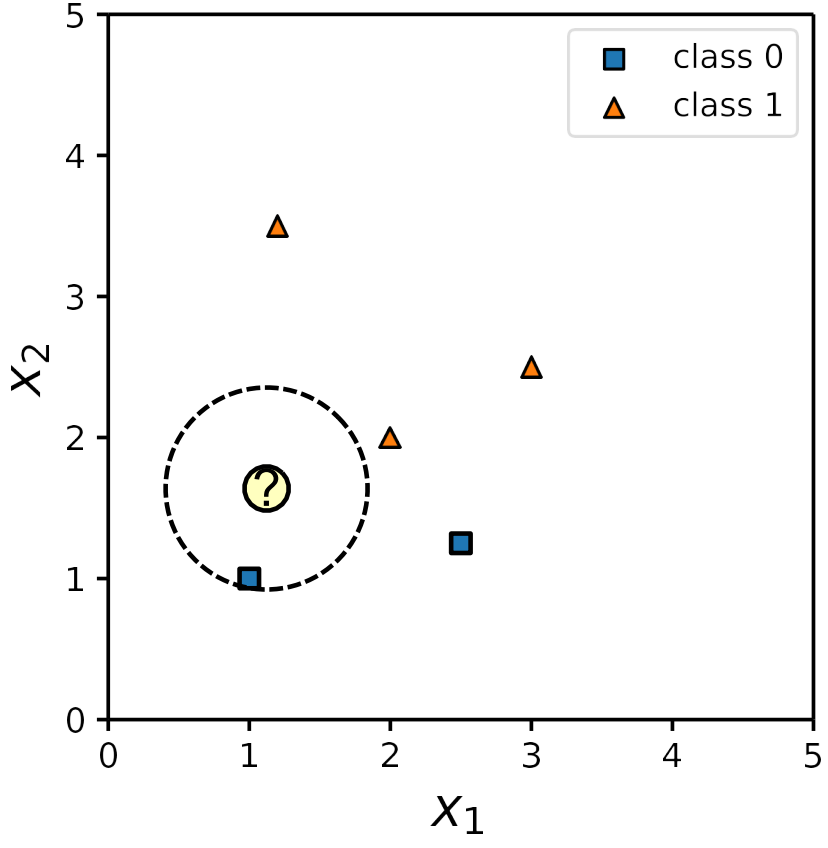
SUPREVISED LEARNING: K-NEAREST NEIGHBOR

Nearest neighbor algorithms are among the simplest supervised machine learning algorithms and have been well studied in the eld of pattern recognition over the last century.
While nearest neighbor algorithms are not as popular as they once were, they are still widely used in practice, and I highly recommend that you are at least considering the k-Nearest
Neighbor algorithm in classification projects as a predictive performance benchmark especially, when you are trying to develop more sophisticated models.

1-NEAREST NEIGHBOR

Task: predict the target / label of a new data point





How? Look at most "similar" data point in training set

1-NEAREST NEIGHBOR TRAINING STEP

$$\langle \mathbf{x}^{[i]}, y^{[i]} \rangle \in \mathcal{D} \quad (|\mathcal{D}| = n)$$

How do we "train" the 1-NN model?

1-NEAREST NEIGHBOR TRAINING STEP

$$\langle \mathbf{x}^{[i]}, y^{[i]} \rangle \in \mathcal{D} \quad (|\mathcal{D}| = n)$$

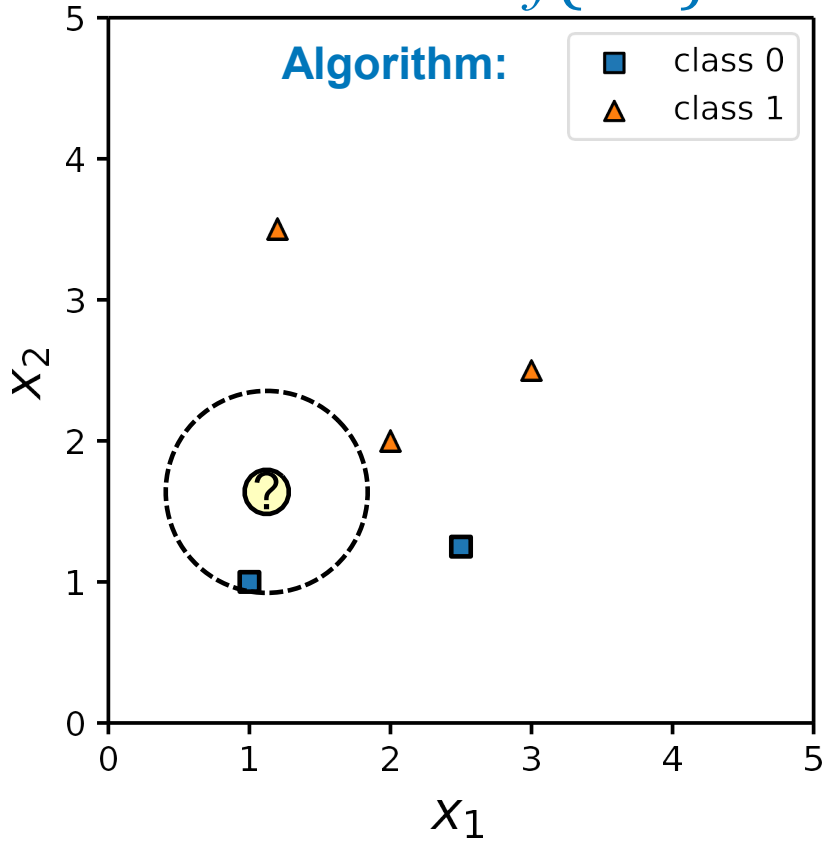
To train the 1-NN model, we simply "remember" the training dataset

1-NEAREST NEIGHBOR PREDICTION STEP

Given: $\langle \mathbf{x}^{[i]}, y^{[i]} \rangle \in \mathcal{D} \quad (|\mathcal{D}| = n)$

$\langle \mathbf{x}^{[q]}, ??? \rangle$

Predict: $f(\mathbf{x}^{[q]})$



closest_point := None

closest_distance := ∞

- for $i = 1, \ldots, n$:
  - current_distance := $d(\mathbf{x}^{[i]}, \mathbf{x}^{[q]})$
  - if current_distance < closest_distance:
    - closest_distance := current_distance
    - closest_point := $\mathbf{x}^{[i]}$
- return f(closest_point)

query point

COMMONLY USED: EUCLIDEAN DISTANCE (L_2)

$$d(\mathbf{x}^{[a]}, \mathbf{x}^{[b]}) = \sqrt{\sum_{j=1}^{m} (x_j^{[a]} - x_j^{[b]})^2}$$

SOME COMMON CONTINUOUS DISTANCE MEASURES

(L2) Euclidean

(L1) Manhattan

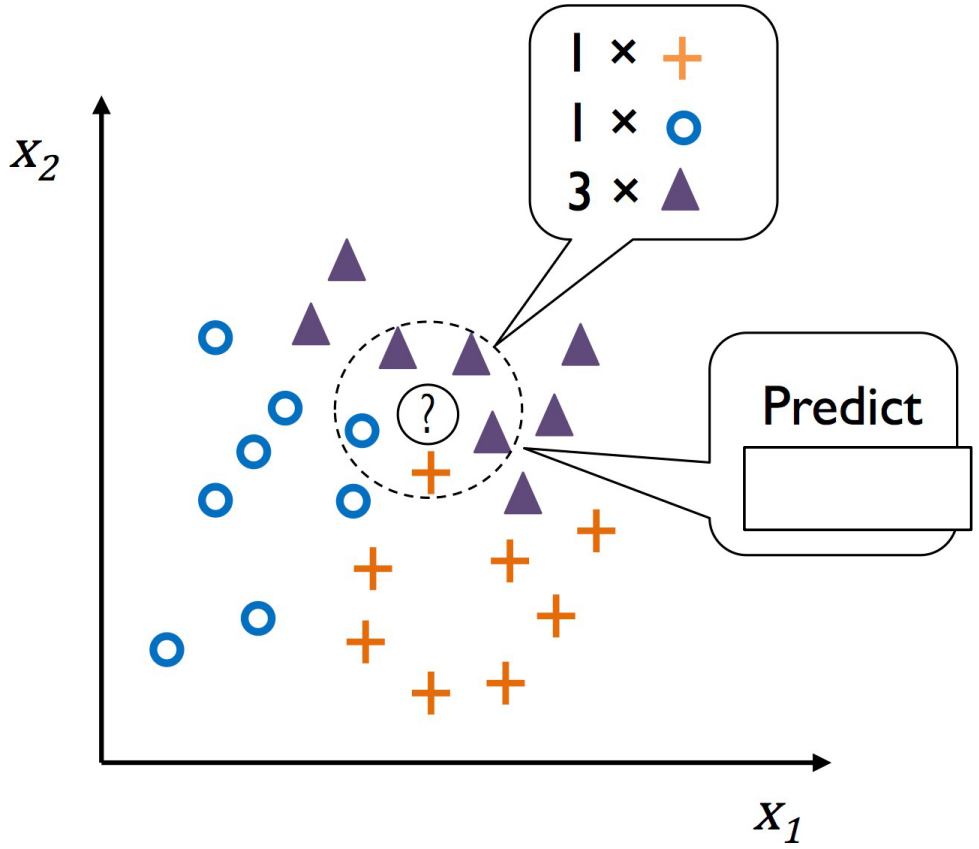
Minkowski: $d(\mathbf{x}^{[a]}, \mathbf{x}^{[b]}) = \left[\sum_{j=1}^{m} \left(\left| x_j^{[a]} - x_j^{[b]} \right| \right)^p \right]^{\frac{1}{p}}$

Mahalanobis

Cosine similarity

…

K-NEAREST NEIGHBORS



*k*NN for Classification

$$\mathcal{D}_k = \{\langle \mathbf{x}^{[1]}, f(\mathbf{x}^{[1]})\rangle, \ldots, \langle \mathbf{x}^{[k]}, f(\mathbf{x}^{[k]})\rangle\} \quad \mathcal{D}_k \subseteq \mathcal{D}$$

$$h(\mathbf{x}^{[q]}) = arg \max_{y \in \{1,\ldots,t\}} \sum_{i=1}^{k} \delta(y, f(\mathbf{x}^{[i]}))$$

$$\delta(a,b) = \begin{cases} 1, & \textbf{if } a = b, \\ 0, & \textbf{if } a \neq b. \end{cases}$$

$$h(\mathbf{x}^{[t]}) = \textbf{mode}(\{f(\mathbf{x}^{[1]}), \ldots, f(\mathbf{x}^{[k]})\})$$

KNN FOR REGRESSION

$$\mathcal{D}_k = \{\langle \mathbf{x}^{[1]}, f(\mathbf{x}^{[1]})\rangle, \ldots, \langle \mathbf{x}^{[k]}, f(\mathbf{x}^{[k]})\rangle\} \qquad \mathcal{D}_k \subseteq \mathcal{D}$$

$$h(\mathbf{x}^{[t]}) = \frac{1}{k}\sum_{i=1}^{k} f(\mathbf{x}^{[i]})$$

K-Nearest Neighbors Implementation

```
class KNNClassifier(object):
    def __init__(self, k, dist_fn=None):
        self.k = k
        if dist_fn is None:
            self.dist_fn = self._euclidean_dist

    def _euclidean_dist(self, a, b):
        dist = 0.
        for ele_i, ele_j in zip(a, b):
            dist += ((ele_i - ele_j)**2)
        dist = dist**0.5
        return dist

    def _find_nearest(self, x):
        dist_idx_pairs = []
        for j in range(self.dataset_.shape[0]):
            d = self.dist_fn(x, self.dataset_[j])
            dist_idx_pairs.append((d, j))

        sorted_dist_idx_pairs = sorted(dist_idx_pairs)

        return sorted_dist_idx_pairs

    def fit(self, X, y):
        self.dataset_ = X.copy()
        self.labels_ = y.copy()
        self.possible_labels_ = np.unique(y)

    def predict(self, X):
        predictions = np.zeros(X.shape[0], dtype=int)
        for i in range(X.shape[0]):
            k_nearest = self._find_nearest(X[i])[:self.k]
            indices = [entry[1] for entry in k_nearest]
            k_labels = self.labels_[indices]
            counts = np.bincount(k_labels,
                                 minlength=self.possible_labels_.shape[0])
            pred_label = np.argmax(counts)
            predictions[i] = pred_label
        return predictions
```

K-Nearest Neighbors Implementation

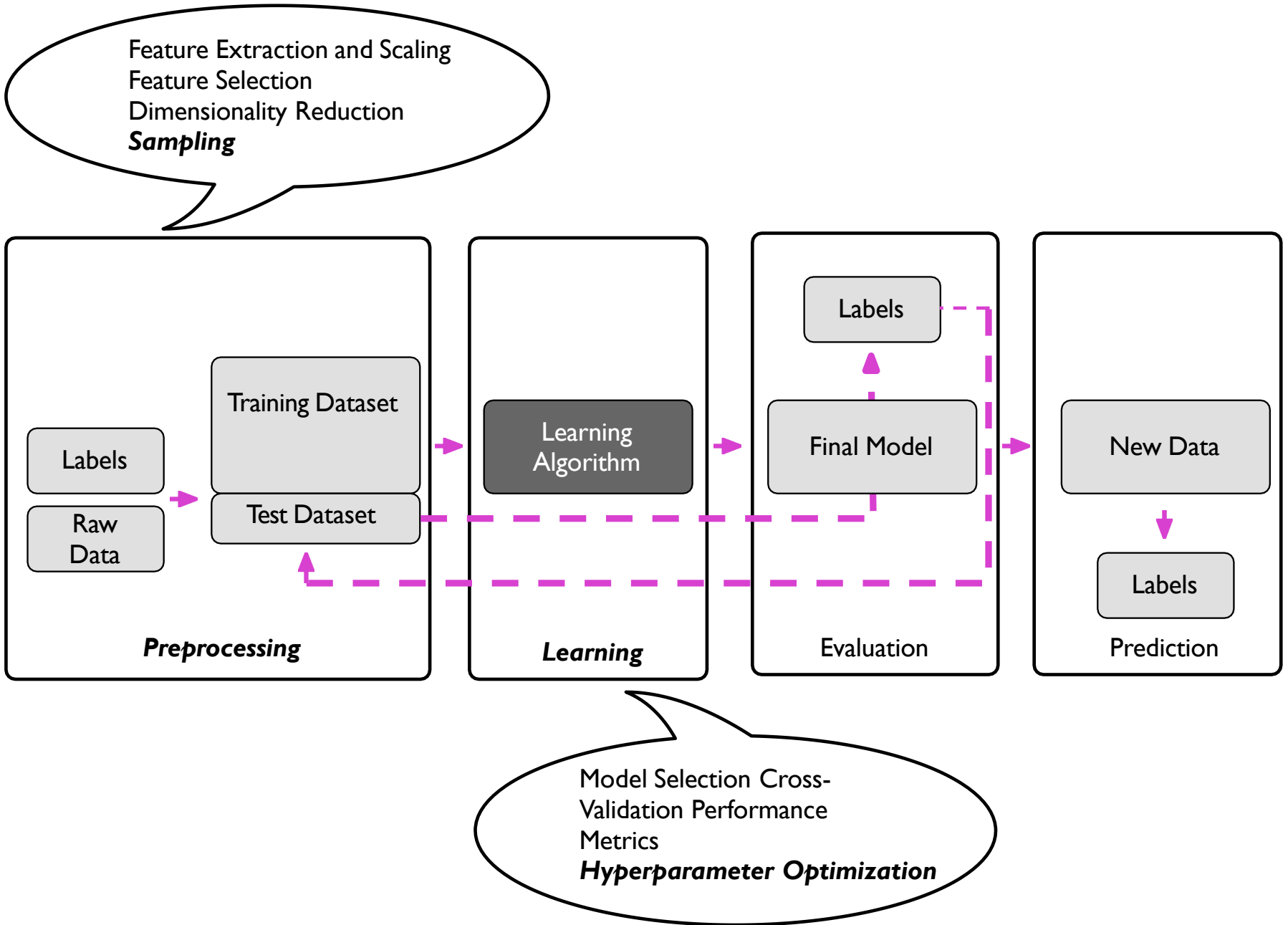
```
class KNNClassifier(object):
    def __init__(self, k, dist_fn=None):
        self.k = k
        if dist_fn is None:
            self.dist_fn = self._euclidean_dist

    def _euclidean_dist(self, a, b):
        dist = 0.
        for ele_i, ele_j in zip(a, b):
            dist += ((ele_i - ele_j)**2)
        dist = dist**0.5
        return dist

    def _find_nearest(self, x):
        dist_idx_pairs = []
        for j in range(self.dataset_.shape[0]):
            d = self.dist_fn(x, self.dataset_[j])
            dist_idx_pairs.append((d, j))

        sorted

        return

    def fit(se
        self.d
        self.l
        self.p

    def predic
        predic
        for i
            k_nearest = self._find_nearest(X[i])[:self.k]
            indices = [entry[1] for entry in k_nearest]
            k_labels = self.labels_[indices]
            counts = np.bincount(k_labels,
                                 minlength=self.possible_labels_.shape[0])
            pred_label = np.argmax(counts)
            predictions[i] = pred_label
        return predictions
```

```
knn_model = KNNClassifier(k=3)
knn_model.fit(X_train, y_train)
```

```
print(knn_model.predict(X_valid))
```

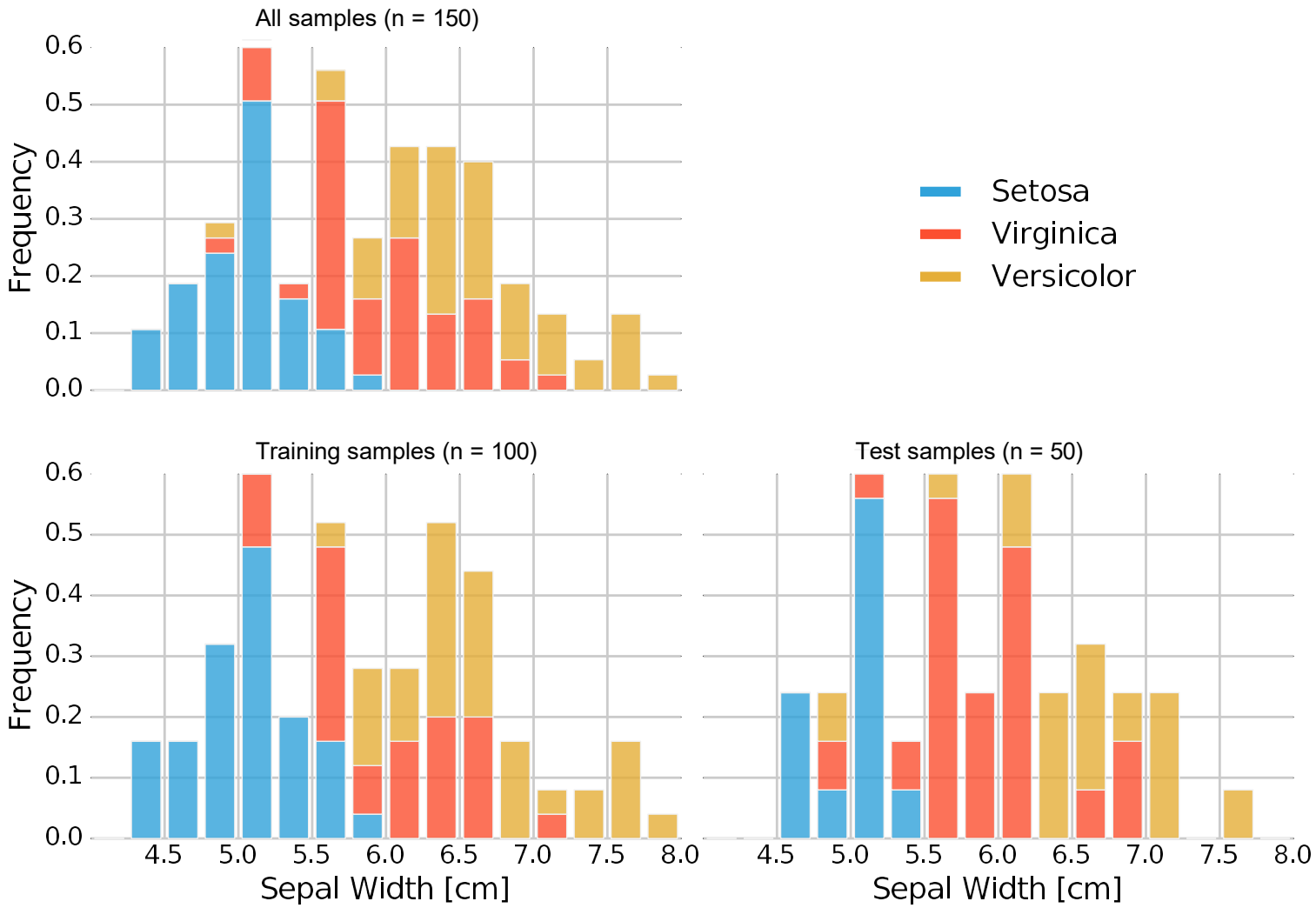
```
[0 1 2 1 1 1 0 0 1 2 0 0 1 1 1 2 1 1 1 2 0 0]
```

MACHINE LEARNING WORKFLOW



PREPROCESSING: SAMPLING

RANDOM SUBSAMPLING ...



LEARNING: ***HYPERPARAMETER OPTIMIZATION***

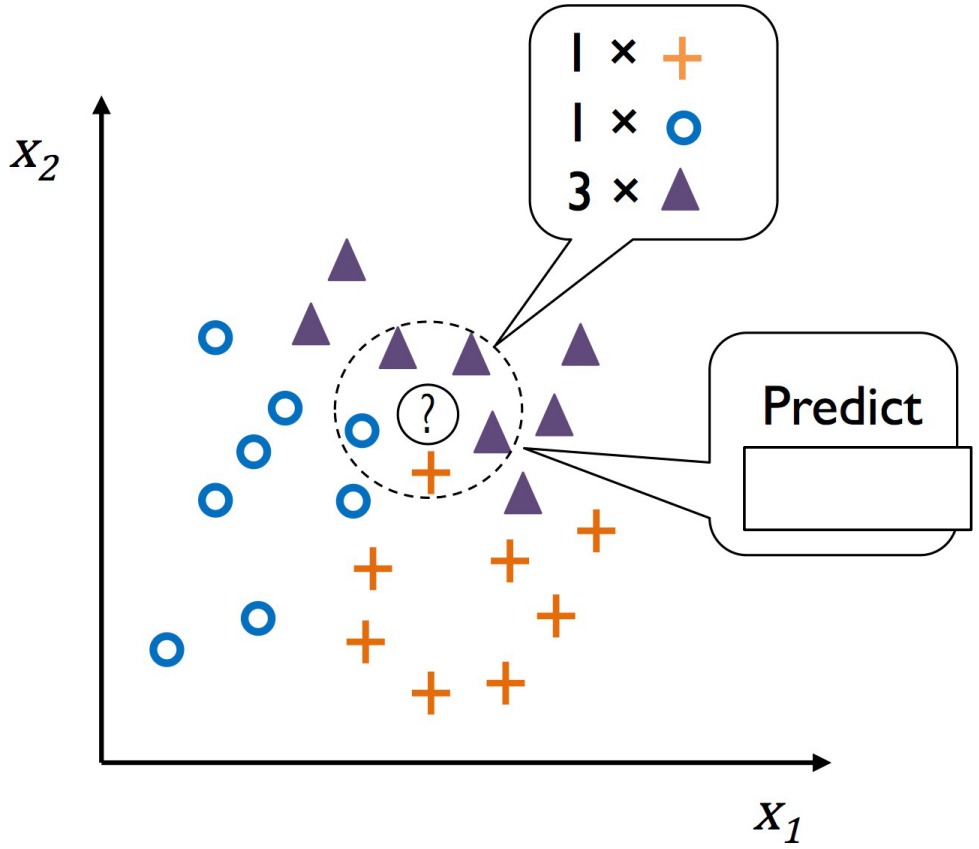
HYPERPARAMETERS

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

COMMONLY USED: EUCLIDEAN DISTANCE (L_2)

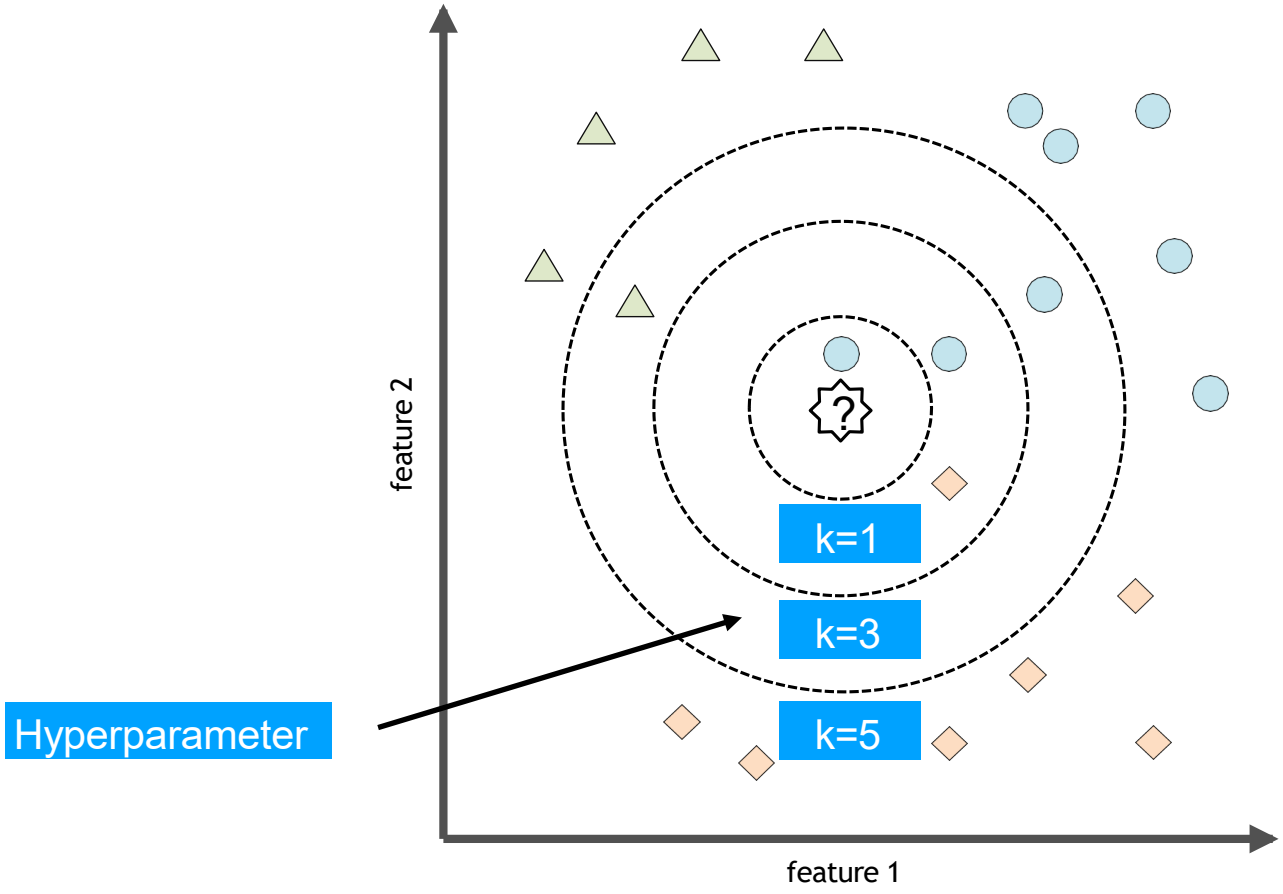
$$d(\mathbf{x}^{[a]}, \mathbf{x}^{[b]}) = \sqrt{\sum_{j=1}^{m} (x_j^{[a]} - x_j^{[b]})^2}$$

K-NEAREST NEIGHBORS



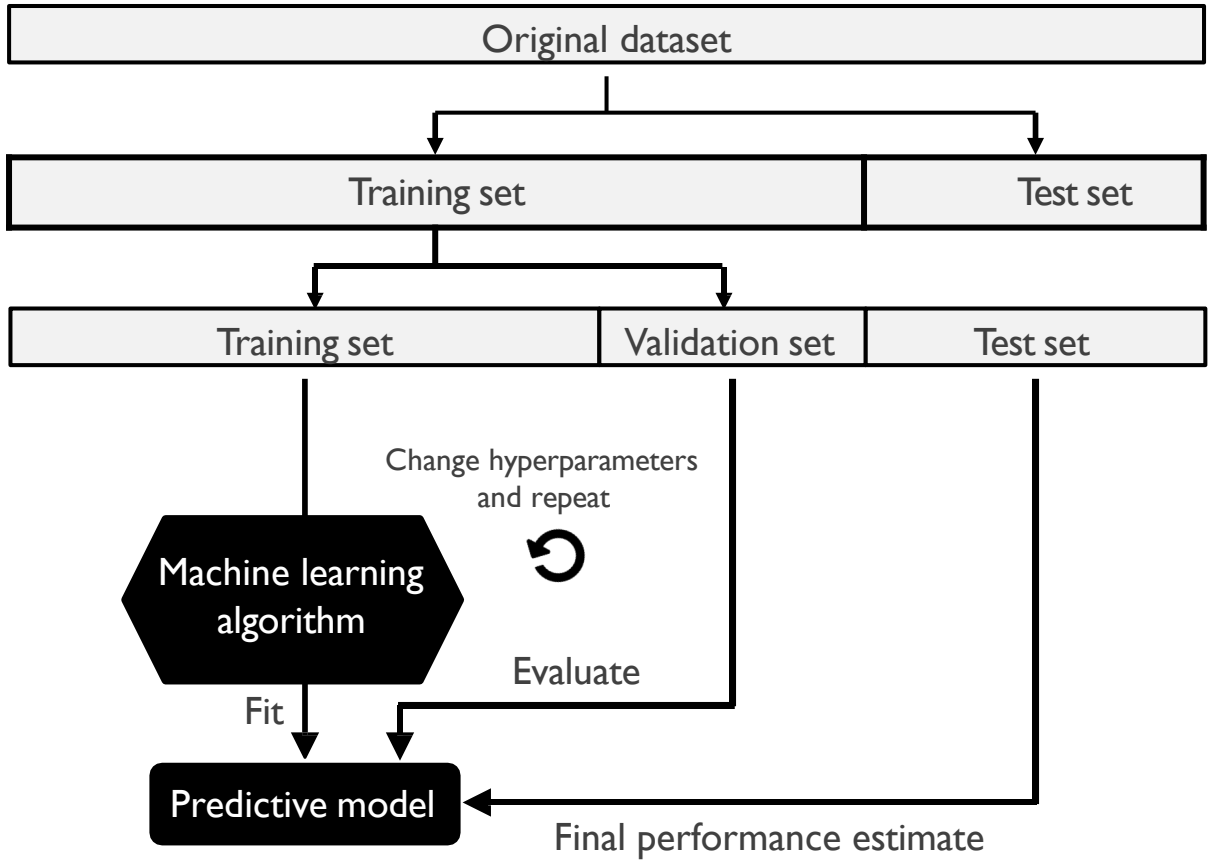
HYPERPARAMETERS

nonparametric model: k-nearest neighbors



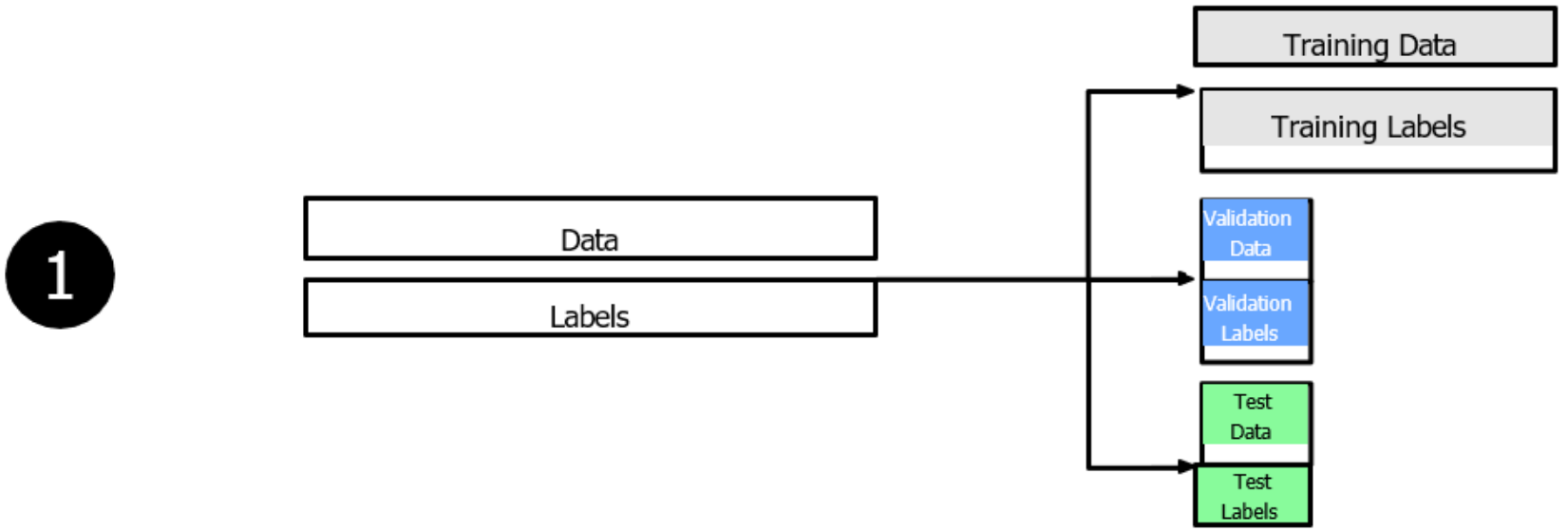
PREPROCESSING: SAMPLING REVIST

SIMPLE HOLDOUT METHOD



3-WAY HOLDOUT

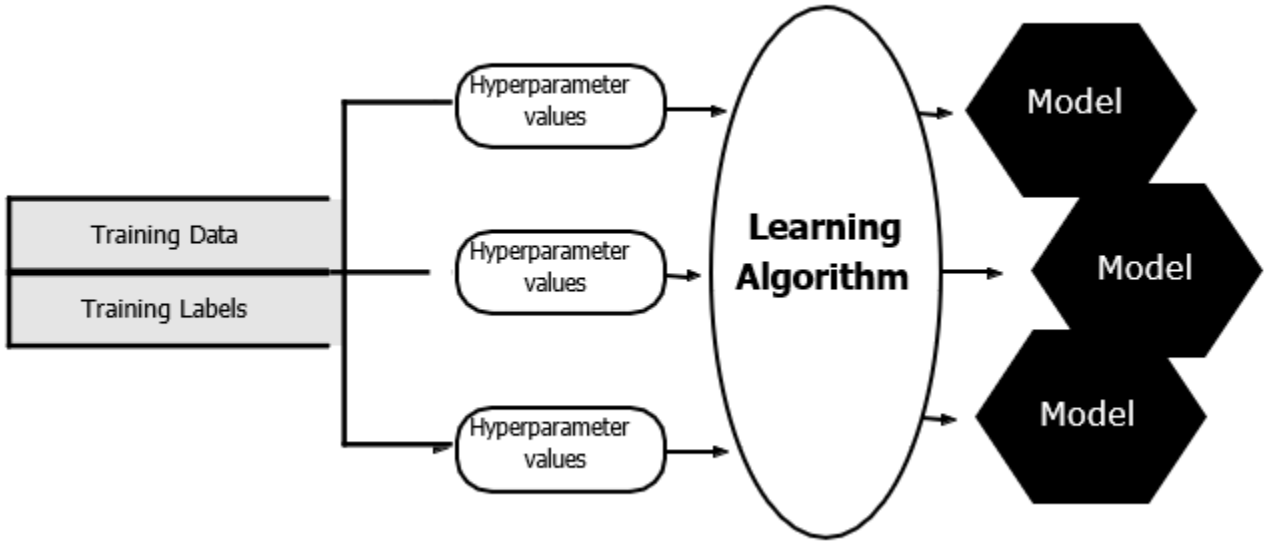
instead of "regular" holdout to avoid "***data leakage***" during hyperparameter optimization



3-WAY HOLDOUT

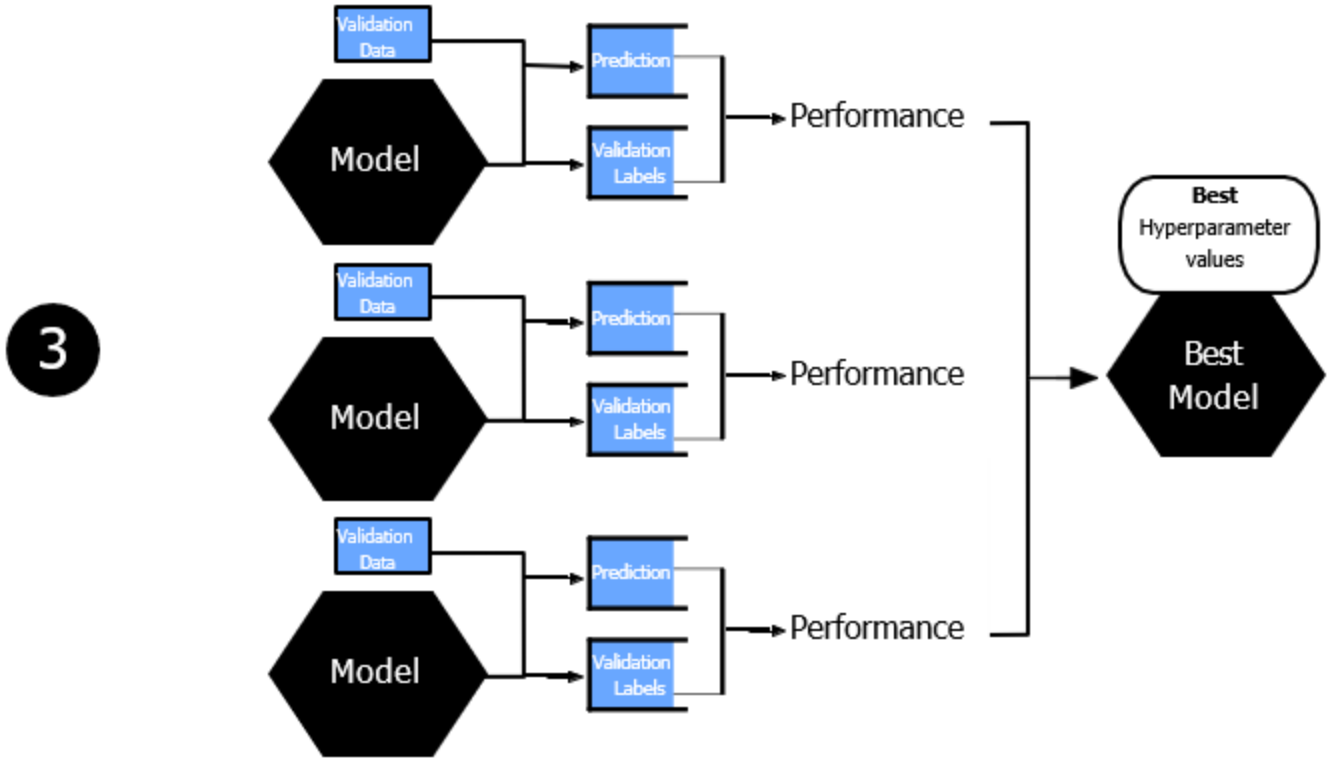
instead of "regular" holdout to avoid "***data leakage***" during hyperparameter optimization

2



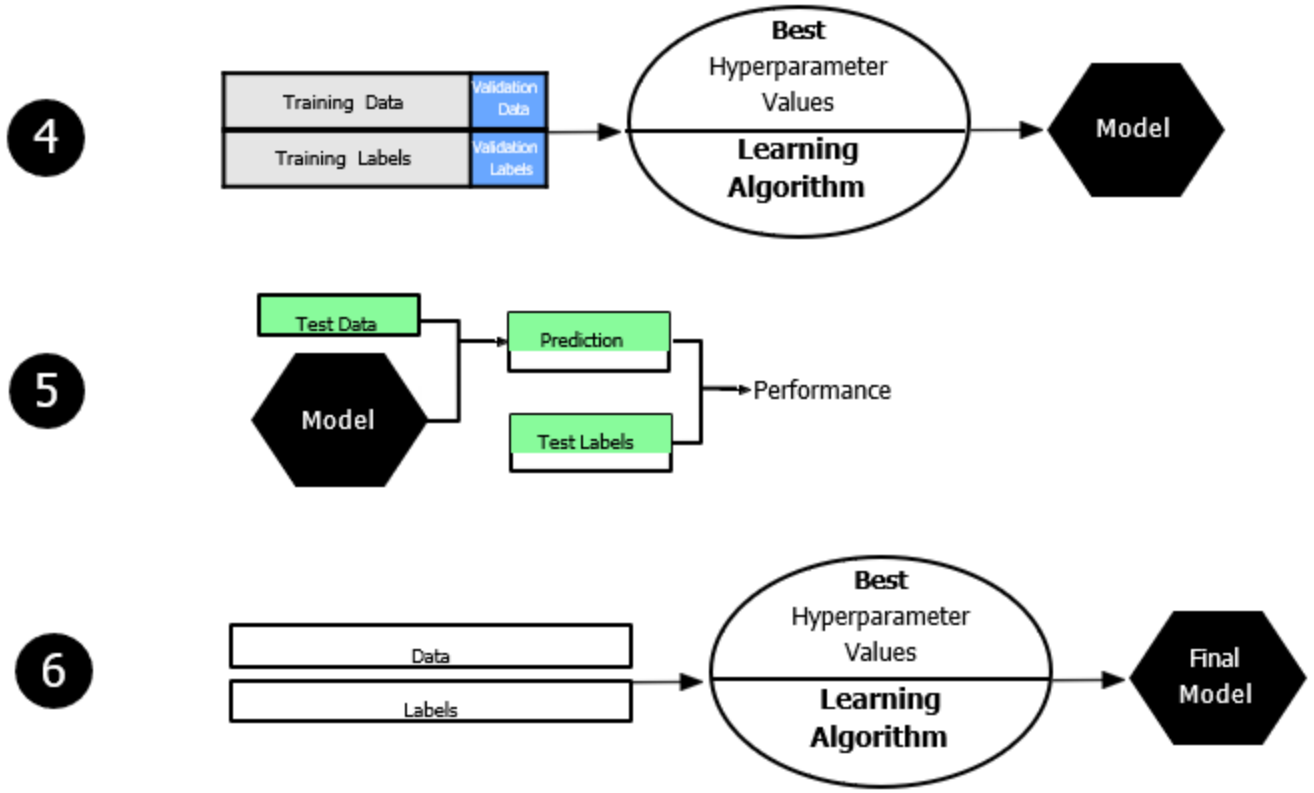
3-WAY HOLDOUT

instead of "regular" holdout to avoid "data leakage" during hyperparameter optimization



3-WAY HOLDOUT

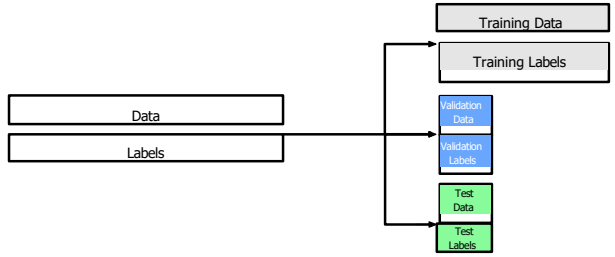
instead of "regular" holdout to avoid "data leakage" during hyperparameter optimization



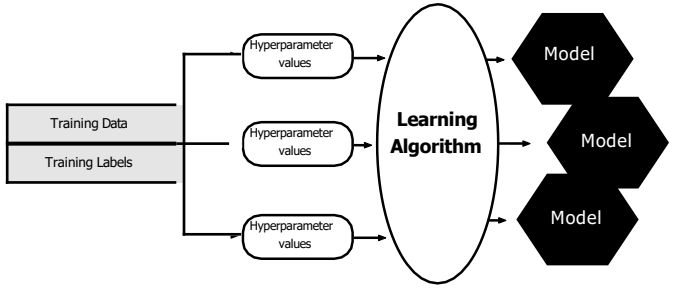
3-WAY HOLDOUT

instead of "regular" holdout to avoid "data leakage" during hyperparameter optimization

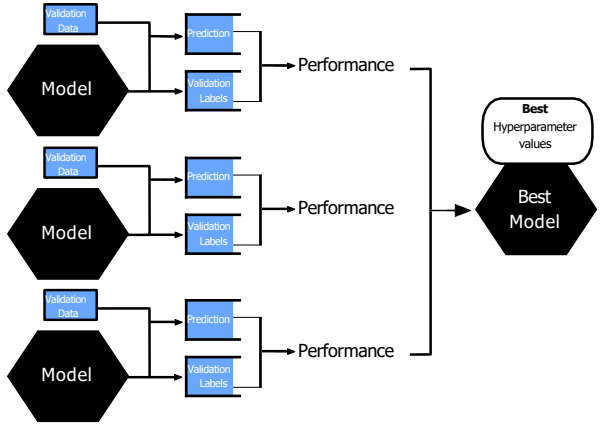
1



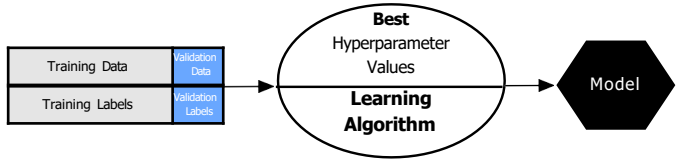
2



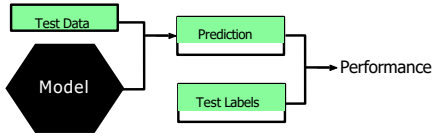
3



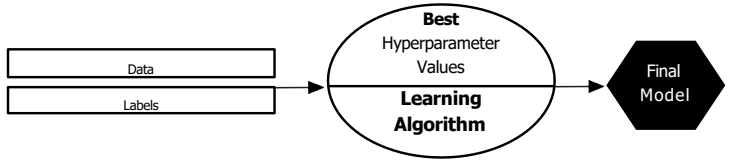
4



5



6



MAIN POINTS WHY WE EVALUATE THE PREDICTIVE PERFORMANCE OF A MODEL:

1. Want to estimate the generalization performance, the predictive performance of our model on future (unseen) data.

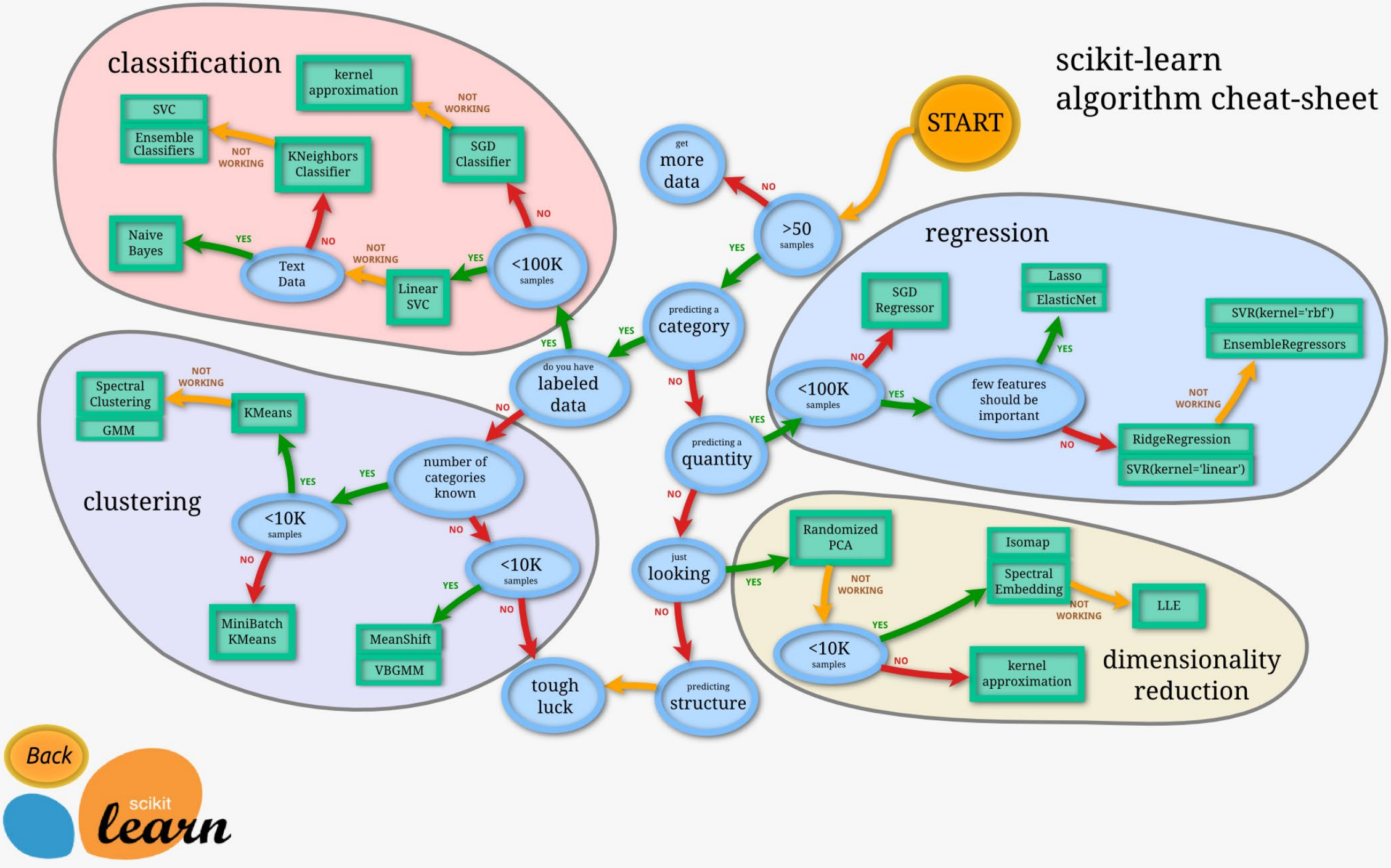
2. Want to increase the predictive performance by tweaking the learning algorithm and selecting the best performing model from a given hypothesis space.

3. Want to identify the ML algorithm that is best-suited for the problem at hand; thus, we want to compare different algorithms, selecting the best-performing one as well as the best performing model from the algorithm's hypothesis space.

MACHINE LEARNING LIBRARIES

- scikit-learn =>1.5.0
- scipy=>1.14.0

```
pip install scikit-learn scipy
```


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

THE SCIKIT-LEARN ESTIMATOR API (AN OOP PARADIGM)

```
class SupervisedEstimator(...):

    def __init__(self, hyperparam_1, ...):
        self.hyperparm_1
        ...

    def fit(self, X, y):
        ...
        self.fit_attribute_
        return self

    def predict(self, X):
        ...
        return y_pred

    def score(self, X, y):
        ...
        return score

    def _private_method(self):
        ...
    ...
```

closest_point := None

closest_distance := ∞

query point

- for $i = 1, \ldots, n$:
  - current_distance := $d(\mathbf{x}^{[i]}, \mathbf{x}^{[q]})$
  - if current_distance < closest_distance:
    - closest_distance := current_distance
    - closest_point := $\mathbf{x}^{[i]}$
- return f(closest_point)

STRATIFIED SPLITS

```
from sklearn.model_selection import train_test_split

X_temp, X_test, y_temp, y_test = \
        train_test_split(X, y, test_size=0.2,
                         shuffle=True, random_state=123, stratify=y)
np.bincount(y_temp)
```

```
array([40, 40, 40])
```

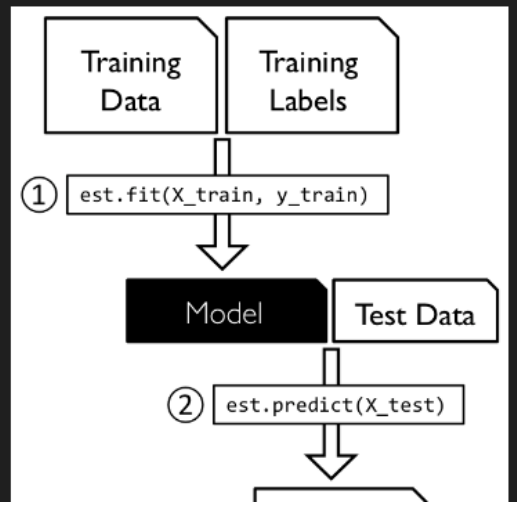
```
X_train, X_valid, y_train, y_valid = \
        train_test_split(X_temp, y_temp, test_size=0.2,
                         shuffle=True, random_state=123, stratify=y_temp)
X_train.shape
```

```
(96, 4)
```

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

Dataset

Simple Holdout Method

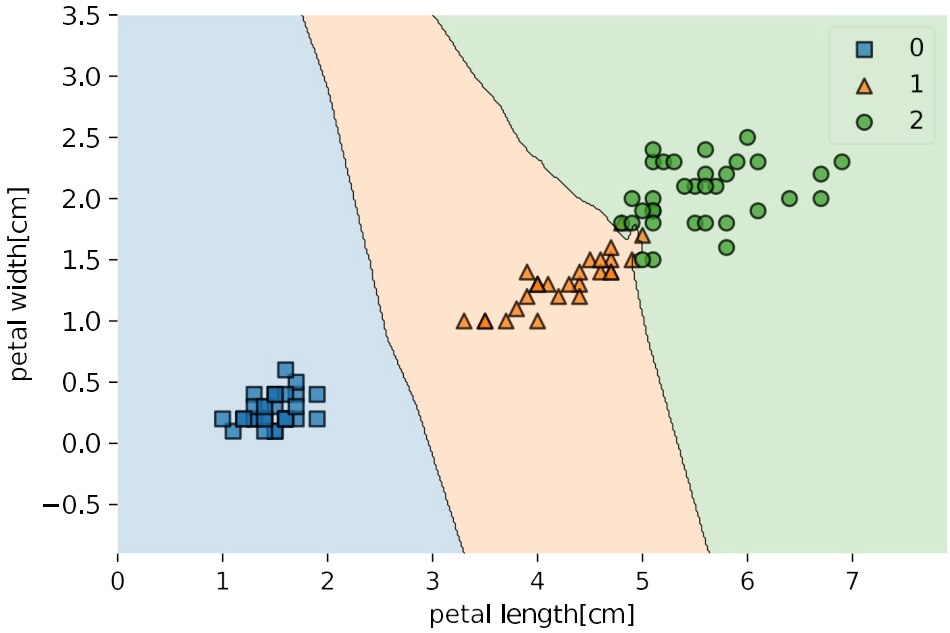


SUPREVISED LEARNING: K-NEAREST NEIGHBOR

https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

A 3-NEAREST NEIGHBOR CLASSIFIER & 2 IRIS FEATURES

```
from sklearn.neighbors import KNeighborsClassifier
from mlxtend.plotting import plot_decision_regions

knn_model = KNeighborsClassifier(n_neighbors=3)
knn_model.fit(X_train[:, 2:], y_train)
plot_decision_regions(X_train[:, 2:], y_train, knn_model)
plt.xlabel('petal length[cm]')
plt.ylabel('petal width[cm]')
plt.savefig('images/decisionreg.pdf')
plt.show()
```



1-knn_class_sklearn.ipynb

G: > My Drive > Academics > Machine Learning > Spring 2025 > Lectures > Lecture 005 > lecture5_code_examples > 1-knn_class_sklearn.ipynb > The Scikit-Learn Estimator API > empty cell

Generate | Code | Markdown | Run All | Clear All Outputs | Outline | ... | Select Kernel

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

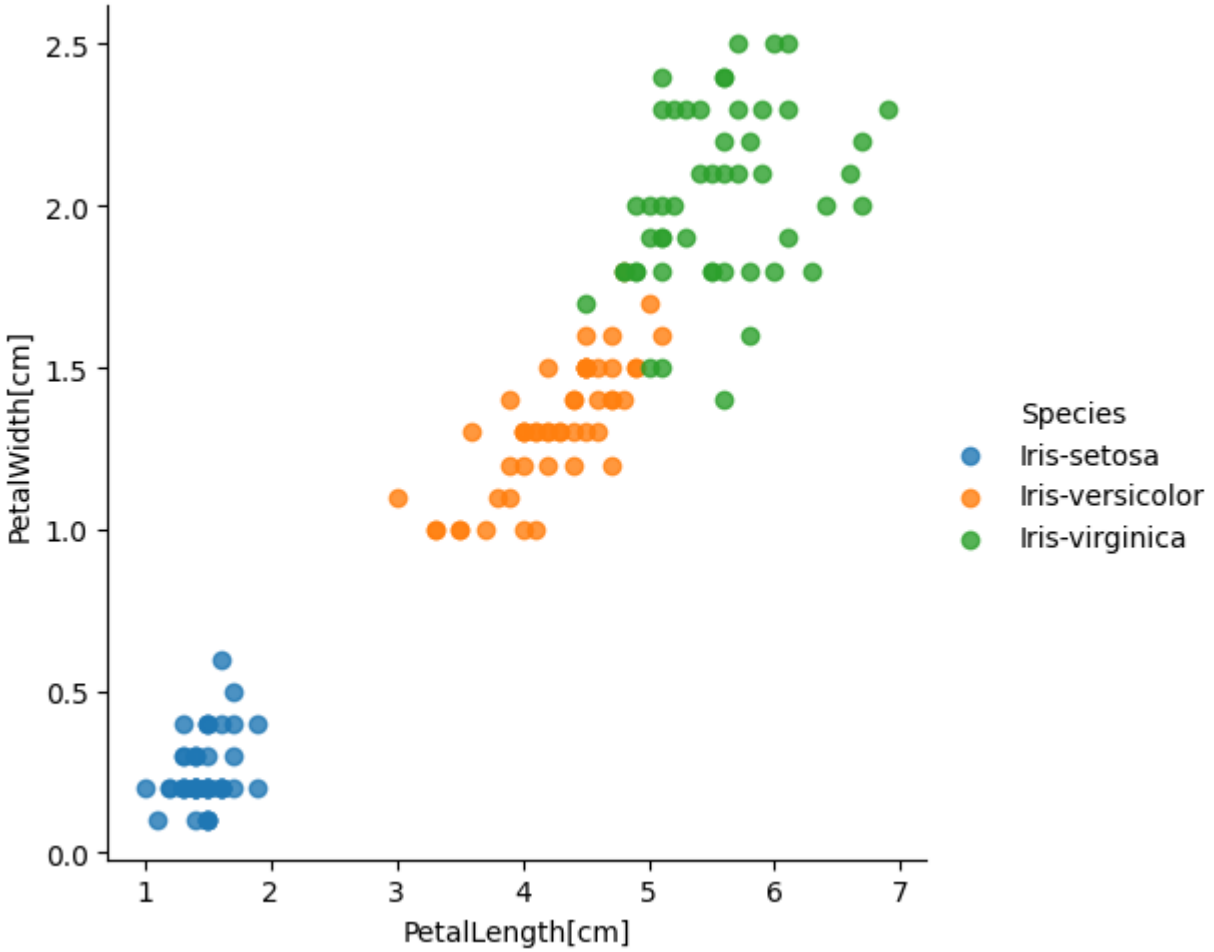
K-Nearest Neighbors Implementation

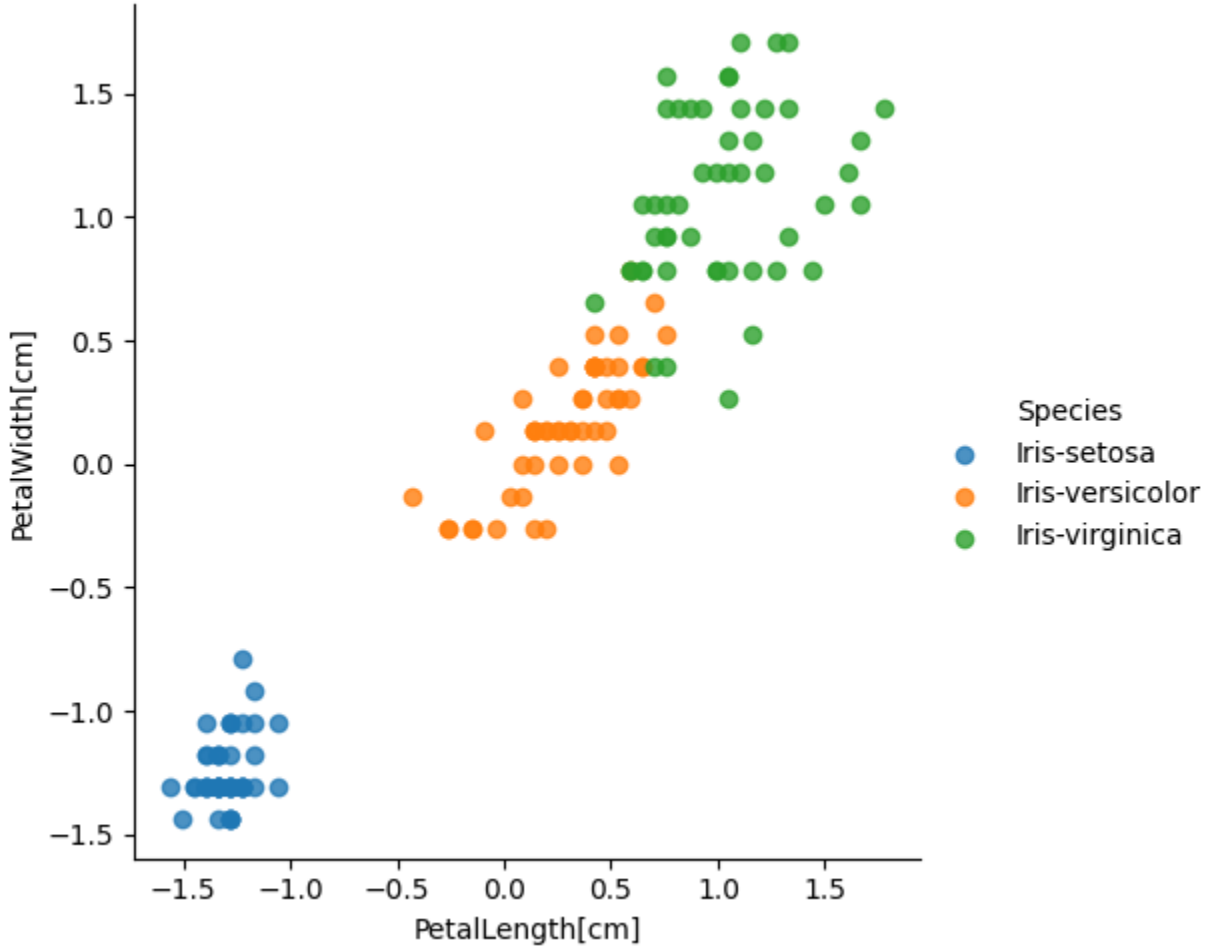
- Below is a very simple implementation of a K-nearest Neighbor classifier.
- This is a very slow and inefficient implementation, and in real-world problems, it is always recommended to use established libraries (like scikit-learn) instead of implementing algorithms from scratch.
- The scikit-learn library, for example, implements *k*NN much more efficiently and robustly
- A scenario where it is useful to implement algorithms from scratch is for learning and teaching purposes, or if we want to try out new algorithms, hence, the implementation below, which gently introduces how things are implemented in scikit-learn.

[41]

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
import pandas as pd

df_iris = pd.read_csv('data/iris.csv')
X = df_iris[['PetalLength[cm]', 'PetalWidth[cm]']]
y = df_iris['Species']
```

NORMALIZATION: STANDARDIZATION





With Standardization

THE SCIKIT-LEARN TRANSFORMER

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler


scaler = StandardScaler()
scaler.fit(X_train)
X_train_std = scaler.transform(X_train)
X_valid_std = scaler.transform(X_valid)
X_test_std = scaler.transform(X_test)
```

2-transforming_dataset_normalization.ipynb

G: > My Drive > Academics > Machine Learning > Spring 2025 > Lectures > Lecture 005 > lecture5_code_examples > 2-transforming_dataset_normalization.ipynb > from sklearn.datasets import load_iris

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

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

```python
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
import numpy as np

iris = load_iris()
X, y = iris.data[:, 2:], iris.target
```

[ ] Python

```python
X_temp, X_test, y_temp, y_test = \
        train_test_split(X, y, test_size=0.2,
                         shuffle=True, random_state=123, stratify=y)
```

[ ] Python

```python
X_train, X_valid, y_train, y_valid = \
        train_test_split(X_temp, y_temp, test_size=0.2,
                         shuffle=True, random_state=123, stratify=y_temp)
```

[ ]

3-sklearn_pipelines.ipynb

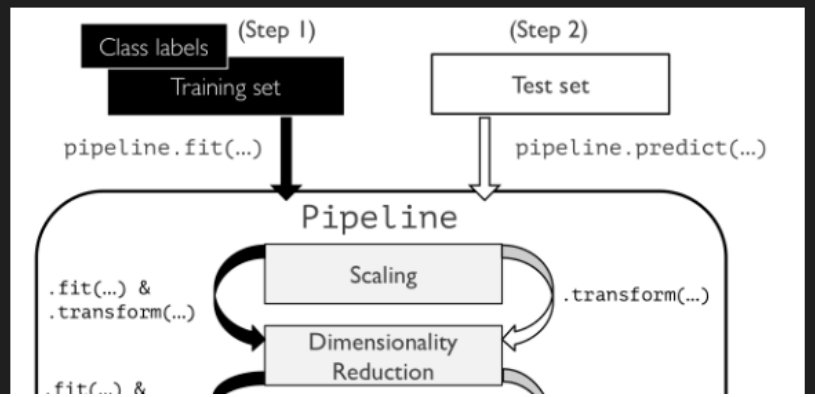
G: > My Drive > Academics > Machine Learning > Spring 2025 > Lectures > Lecture 005 > lecture5_code_examples > 3-sklearn_pipelines.ipynb > Scikit-Learn Pipelines > empty cell

Generate | Code | Markdown | Run All | Clear All Outputs | Outline | ... | Select Kernel

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

Scikit-Learn Pipelines

- Scikit-learn pipelines are an extremely convenient and powerful concept -- one of the things that sets scikit-learn apart from other machine learning libraries.
- Pipelines basically let us define a series of perprocessing steps together with fitting an estimator.
- Pipelines will automatically take care of pitfalls like estimating feature scaling parameters from the training set and applying those to scale new data
- Below is an visualization of how pipelines work.



Q/A?

