SUPERVISED LEARNING

with Python Scikit-Learn
k-Nearest Neighbors

February 2023

Objectives

- Introduce the general “pipeline” when using machine learning methods
- Demonstrate a Python application for k-Nearest Neighbors (kNN) using scikit-learn

Overview

The Black Box

- Using models or methods without the need to know how it works
- “Plug and Play”: use inputs, get output
- In this slide set, we use a black box kNN model; this can be swapped with other machine learning methods

Dataset Examples

Scholarship Application

- Dataset for scholarship approval (fabricated)
- From previous lesson

Instance	number siblings	household income	HS grade	**approve**
1	3	0.8	3	**yes**
2	4	3	2	**no**
3	5	2	3.7	**yes**
4	2	1.5	3	**no**
5	0	1.2	2	**no**
6	1	3.5	3	**no**
7	2	1.6	3	**yes**
8	1	0.5	3.5	**yes**
9	2	0.8	2.9	**no**
10	5	2.3	3.8	**yes**

Iris Dataset

- Dataset for different Iris Species
- From scikit-learn

Iris setosa

Iris versicolor

Iris viriginica

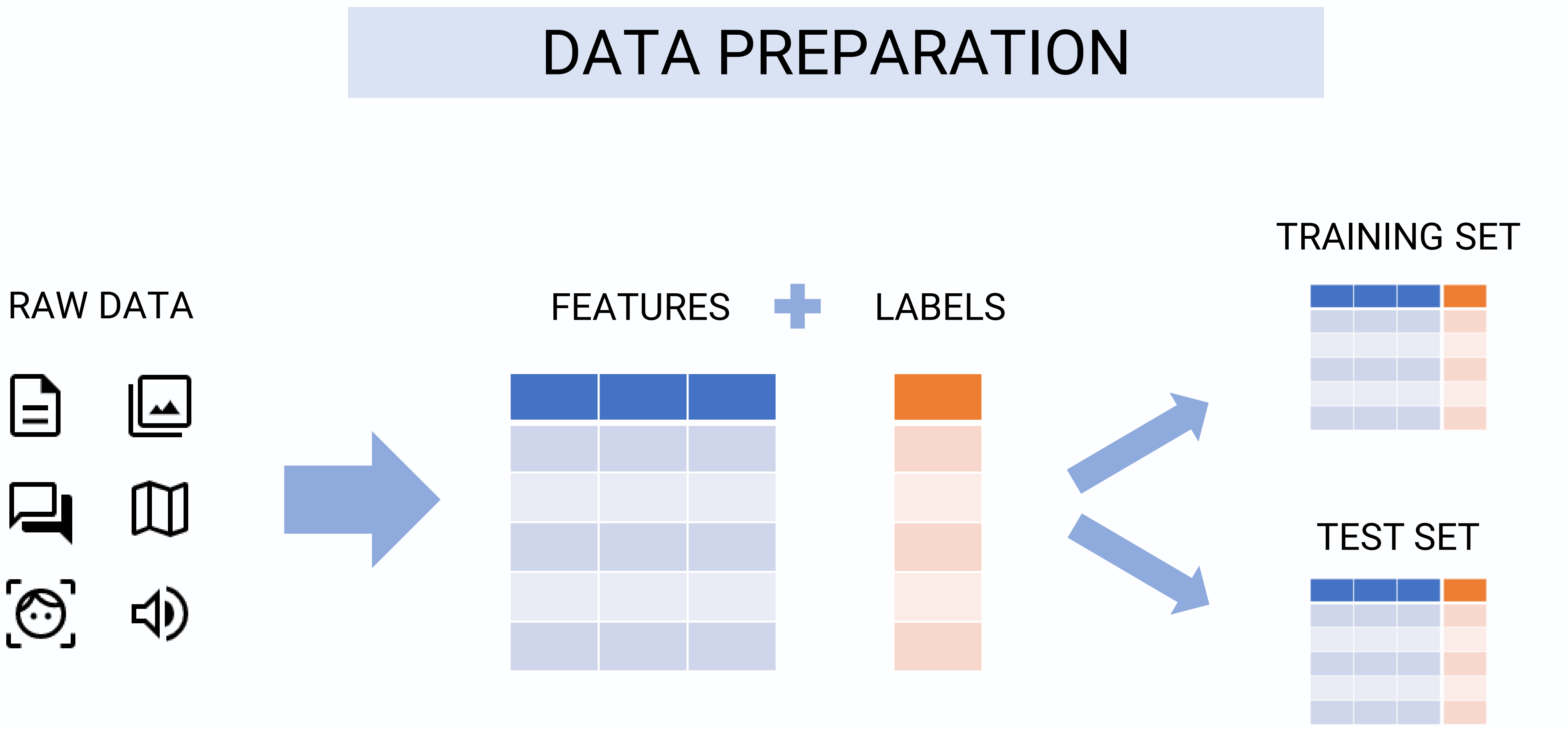
Images from [Iris Flower Dataset Wikipedia page]

Overview

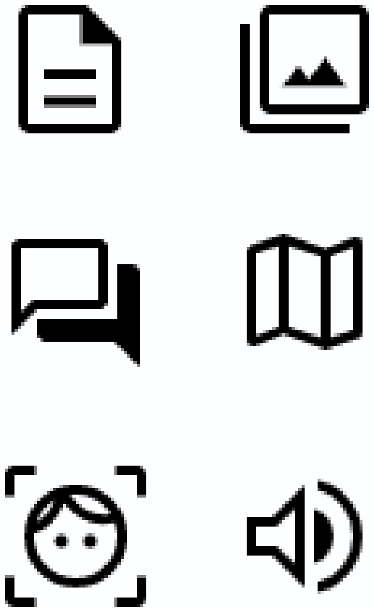
When working with most models:

1. Prepare the Dataset
2. Train the Model
3. Test the Model
4. Evaluate Results

Overview


DATA PREPARATION
RAW DATA
FEATURES
LABELS
TRAINING SET
TEST SET

Data Preparation

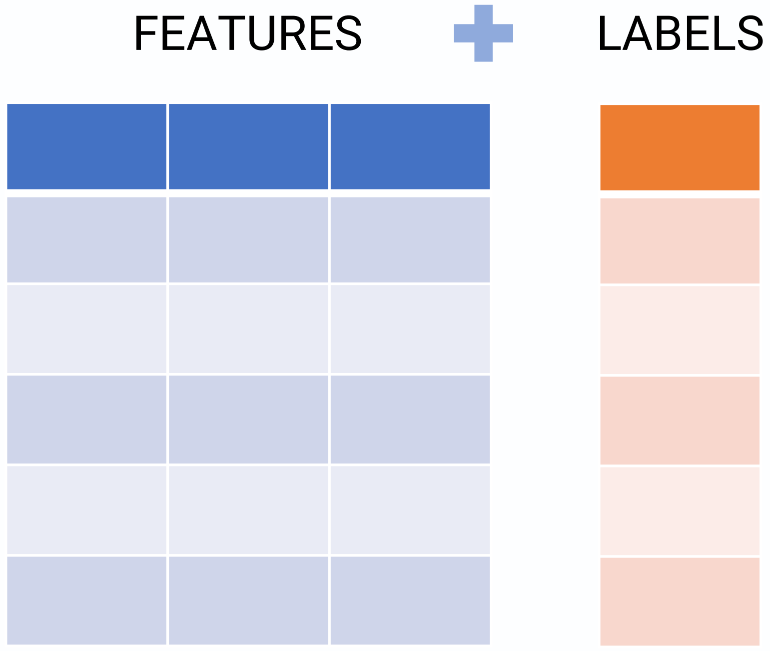


How do we translate “objects” into data that machines can understand?

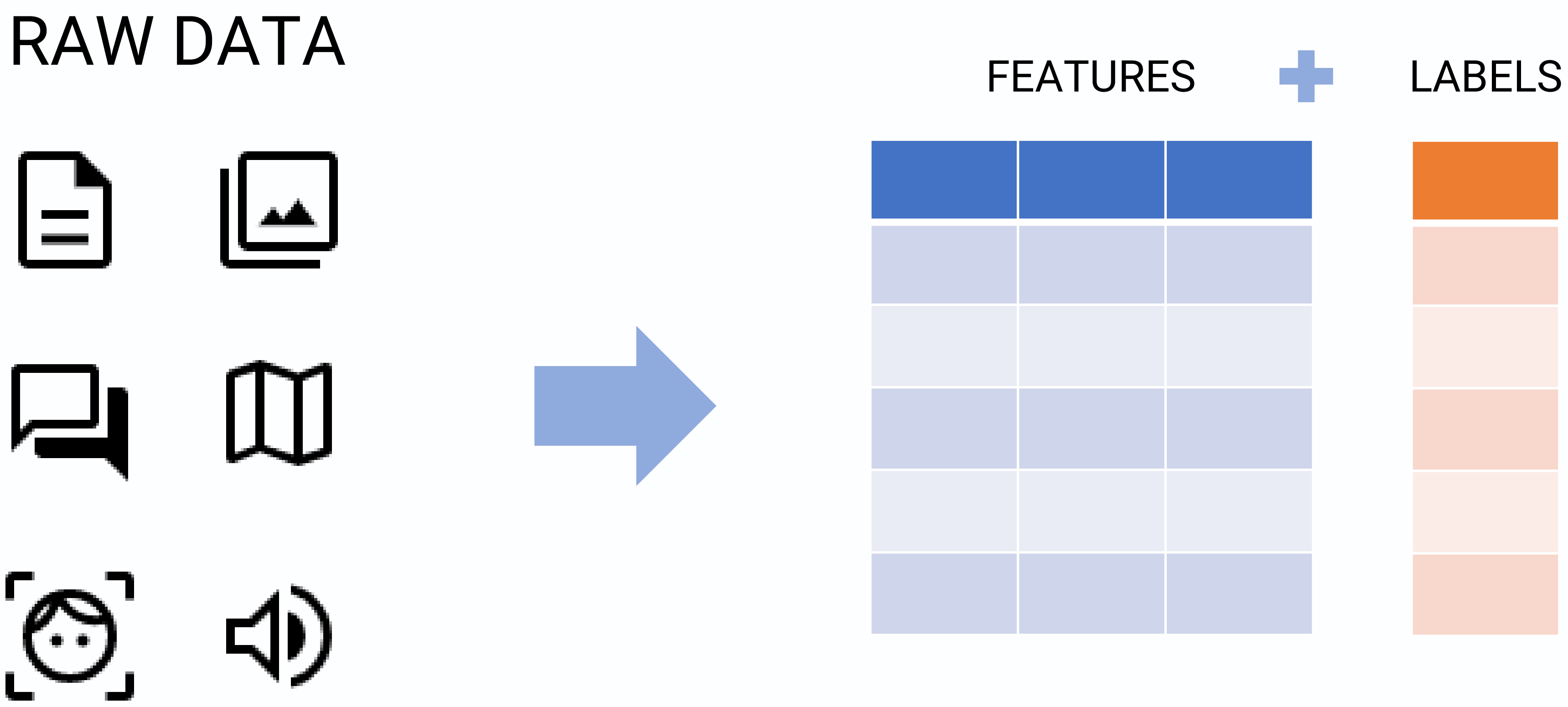
Data Preparation

Structured Data

- Instances (rows)
- Features (columns)
- Label



Data Preparation



Data Preparation

RAW DATA

Love this game! So fun and casual friendly

Wow gotta love how this crashes every minute :)

dead game

Data Preparation

RAW DATA

Love this game! So fun and casual friendly

Wow gotta love how this crashes every minute :)

dead game

FEATURES + LABELS

Positive words	Negative words	Total words	
4	0	8	Positive
3	1	9	Negative
0	1	2	Negative

Data Preparation

SCHOLARSHIP APPLICATIONS

FEATURES			LABELS
number siblings	household income	HS grade	approve
3	0.8	3	yes
4	3	2	no
5	2	3.7	yes
2	1.5	3	no
0	1.2	2	no
1	3.5	3	no
2	1.6	3	yes
1	0.5	3.5	yes
2	0.8	2.9	no
5	2.3	3.8	yes

Data Preparation

IRIS FLOWERS

FEATURES

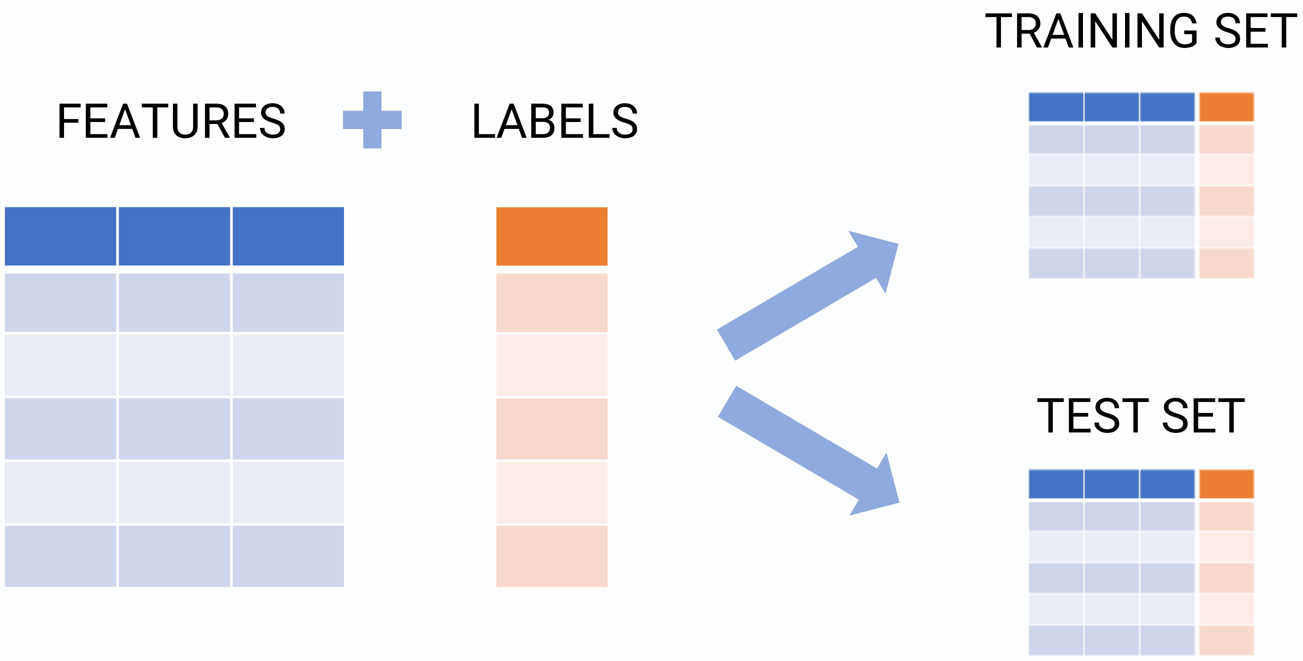
sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
6.0	3.0	4.8	1.8
5.5	2.4	3.8	1.1
6.0	2.2	5.0	1.5
4.8	3.1	1.6	0.2

LABELS

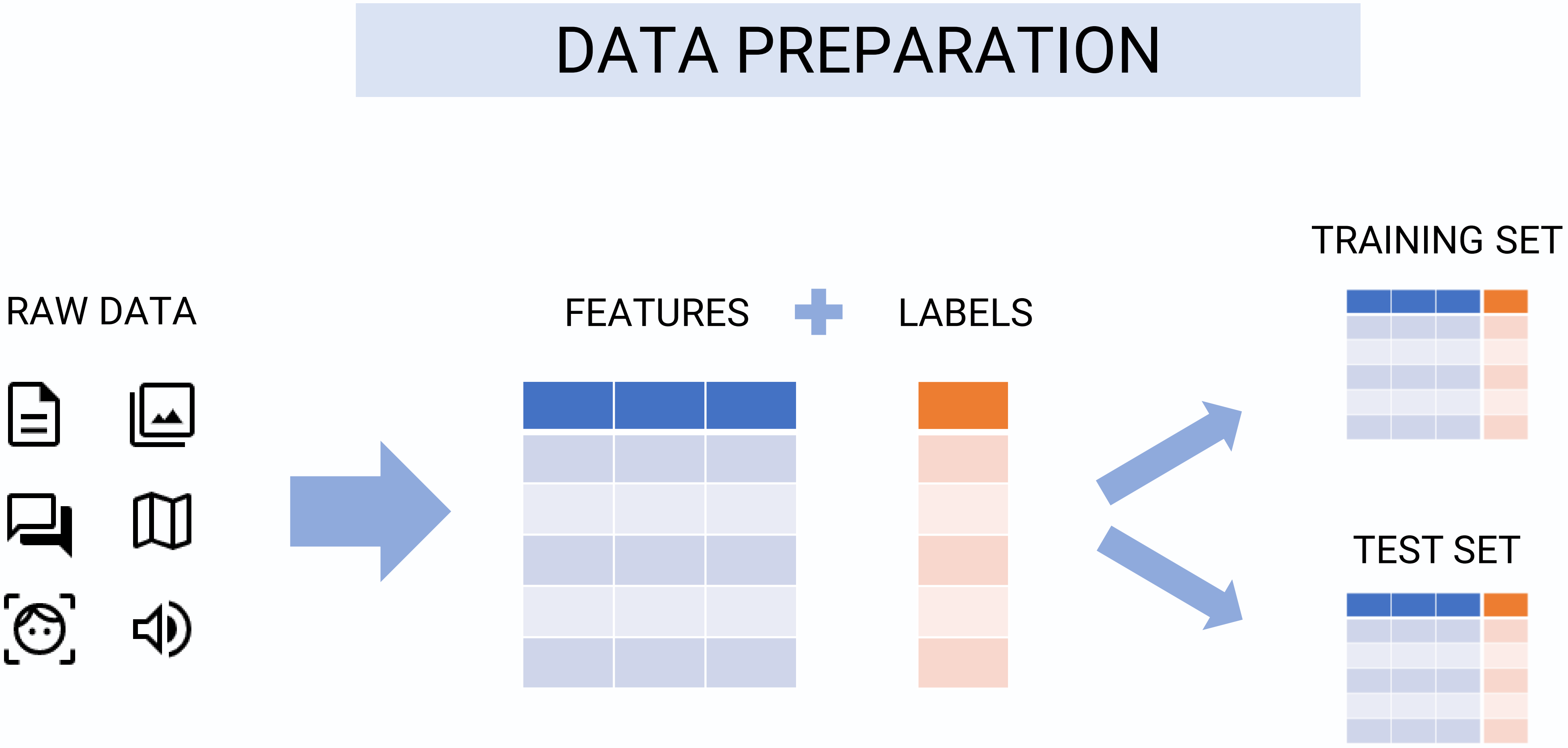
label
2
1
2
0

Data Preparation

To evaluate a classification model, we need separate datasets for training and testing



Overview


DATA PREPARATION
TRAINING SET
RAW DATA
FEATURES
LABELS
TEST SET

Data Preparation (Python Application)

Structured Data

- Can be stored in a format like .csv (comma separated values)
- Can be retrieved from some data sources or packages
- Often in table format

Scholar.csv

```
Sibs,Income,Grade,Approve
3,0.8,3,yes
4,3,2,no
5,2,3.7,yes
2,1.5,3,no
0,1.2,2,no
1,3.5,3,no
```

Data Preparation (Python Application)

Scholarship Application

- Saved in a .csv file named "scholar.csv"
- Retrieved as a DataFrame using pandas read_csv()

In [1]:

```
import pandas as pd
df = pd.read_csv('scholar.csv')
df
```

Out[1]:

	Sibs	Income	Grade	Approve
0	3	0.8	3.0	yes
1	4	3.0	2.0	no
2	5	2.0	3.7	yes
3	2	1.5	3.0	no
4	0	1.2	2.0	no
5	1	3.5	3.0	no
6	2	1.6	3.0	yes
7	1	0.5	3.5	yes
8	2	0.8	2.9	no
9	5	2.3	3.8	yes
10	1	1.0	4.0	NaN
11	0	3.0	4.0	NaN
12	1	1.0	2.5	NaN
13	5	1.0	2.5	NaN

Data Preparation (Python Application)

Scholarship Application

- Training set: first 10 instances
- Variables
  - X - features
  - y - labels

In [2]:

```
df_train = df.head(10)

X = df_train[['Sibs','Income','Grade']].values
y = df_train['Approve']
df_train
```

Out[2]:

	Sibs	Income	Grade	Approve
0	3	0.8	3.0	yes
1	4	3.0	2.0	no
2	5	2.0	3.7	yes
3	2	1.5	3.0	no
4	0	1.2	2.0	no
5	1	3.5	3.0	no
6	2	1.6	3.0	yes
7	1	0.5	3.5	yes
8	2	0.8	2.9	no
9	5	2.3	3.8	yes

Data Preparation (Python Application)

Scholarship Application

- Test set: last 4 instances
- Variables
  - `samples` - features

```
In [3]:
df_test = df.tail(4)
samples = df_test[['Sibs','Income','Grade']].values
df_test
```

Out[3]:

	Sibs	Income	Grade	Approve
10	1	1.0	4.0	NaN
11	0	3.0	4.0	NaN
12	1	1.0	2.5	NaN
13	5	1.0	2.5	NaN

Data Preparation (Application)

Iris Dataset

- from Python's scikit-learn datasets
- Saved in a DataFrame

In [1]:
```
from sklearn.datasets import load_iris
import pandas as pd

iris = load_iris()
iris_df = pd.DataFrame(data=iris.data,
                       columns=iris.feature_names)

iris_df
```

Out[1]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

Data Preparation (Application)

Iris Dataset

- Training set: 75% of the original dataset (random sample)
- Variables
  - `iris_train_df` - features
  - `iris_train_y` - labels

```
In [2]: iris_train_df = iris_df.sample(frac = 0.75)
        iris_train_y = iris.target[iris_train_df.index]

        iris_train_df.head()
```

Out[2]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
9	4.9	3.1	1.5	0.1
133	6.3	2.8	5.1	1.5
15	5.7	4.4	1.5	0.4
98	5.1	2.5	3.0	1.1
117	7.7	3.8	6.7	2.2

Data Preparation (Application)

Iris Dataset

- Test set: other 25% of the dataset
- Variables
  - `iris_test_df` - features
  - `iris_test_y` - labels

Out[3]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
5	5.4	3.9	1.7	0.4
6	4.6	3.4	1.4	0.3
8	4.4	2.9	1.4	0.2

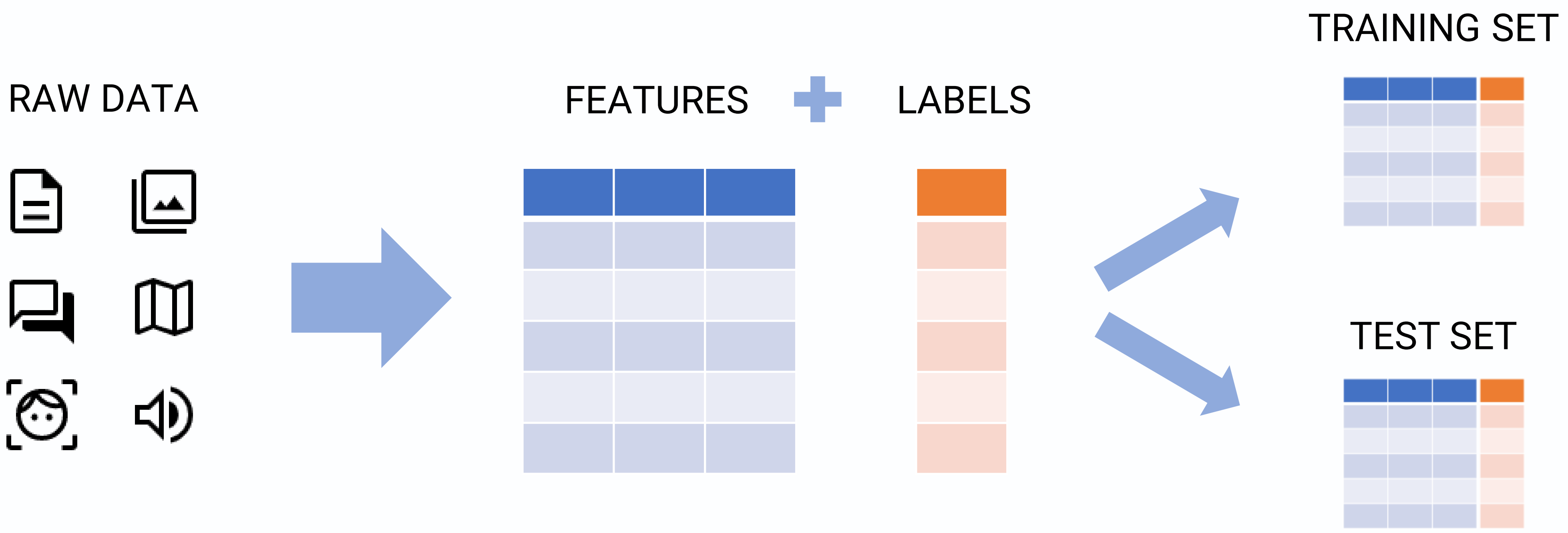
In [3]:

```
iris_test_df = iris_df.drop(iris_train_df.index)
iris_test_y = iris.target[iris_test_df.index]

iris_test_df.head()
```

Overview

DATA PREPARATION

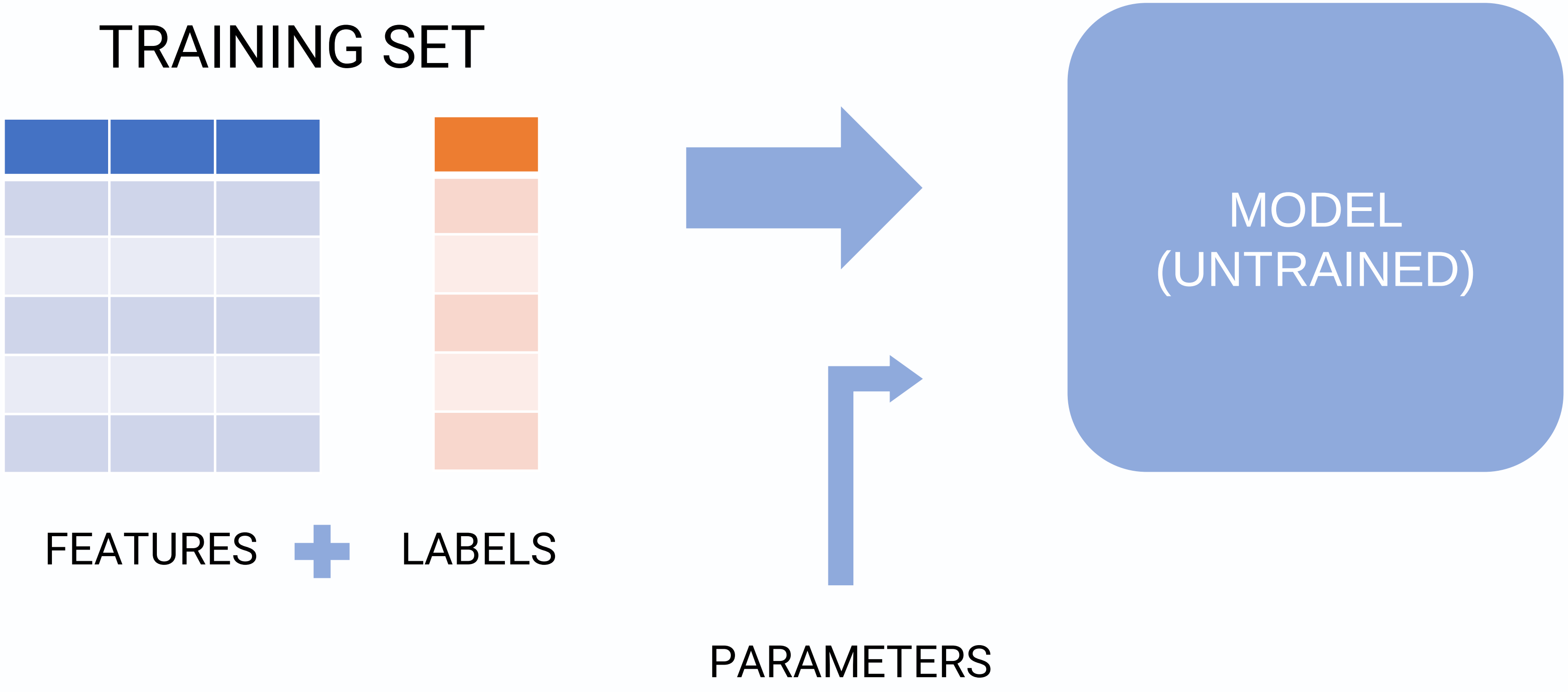


Overview

DATA PREPARATION

Overview

TRAINING THE MODEL



Training The Model

Model

- Source: from scratch or import from an existing package (e.g. sklearn)

Input: Training Set

- Features
- Labels
- Parameters
    - e.g. : for kNN, k is a parameter

Training The Model (Application)

1. Import the model from a package (sklearn)
2. Create an instance of the model
3. Train the model (often using the fit() function)

In [1]:
```
# KNN
from sklearn.neighbors import KNeighborsClassifier
```

...

In [6]:
```
knn1 = KNeighborsClassifier(n_neighbors=1)
knn1.fit(iris_train_df, iris_train_y)
```

Training The Model (Application)

You can create multiple models with different parameters

In [5]:
```python
knn1 = KNeighborsClassifier(n_neighbors=1)
knn1.fit(iris_train_df, iris_train_y)
```

Out[5]:
```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=1)
```

k = 1

In [6]:
```python
knn3 = KNeighborsClassifier(n_neighbors=3)
knn3.fit(iris_train_df, iris_train_y)
```

Out[6]:
```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3)
```

k = 3

Training The Model (Application)

Scholarship Application

- Training set: first 10 instances
  - X - features
  - y - labels

```
In [4]:  knn1 = KNeighborsClassifier(n_neighbors=1)
         knn1.fit(X, y)
         knn3 = KNeighborsClassifier(n_neighbors=3)
         knn3.fit(X, y)
```

Training The Model (Application)

Iris Dataset

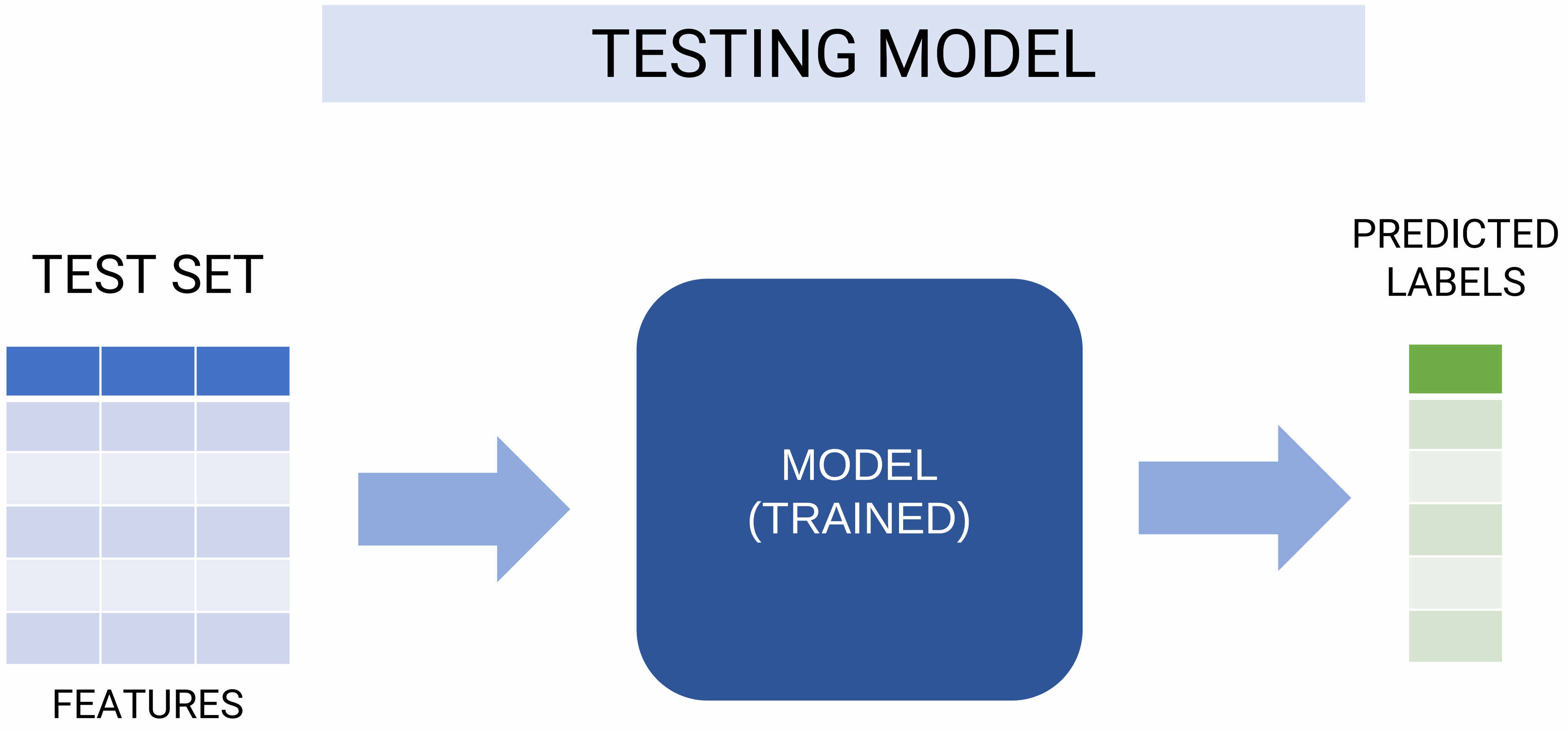
- Training set: 75% of the original dataset

  Variables

  - `iris_train_df` - features
  - `iris_train_y` - labels

In [5]:
```
knn1 = KNeighborsClassifier(n_neighbors=1)
knn1.fit(iris_train_df, iris_train_y)
knn3 = KNeighborsClassifier(n_neighbors=3)
knn3.fit(iris_train_df, iris_train_y)
```

Overview


TESTING MODEL
TEST SET
FEATURES
MODEL
(TRAINED)
PREDICTED
LABELS

Testing The Model

Input

- Test set: features only

Output

- The trained model’s predicted label for each instance

Testing The Model (Application)

Scholarship Application

- Test set: last 4 instances
  - `samples` – features of test set
- Output: array

In [5]:
```
knn1_prediction = knn1.predict(samples)
knn3_prediction = knn3.predict(samples)
```

Testing The Model (Application)

Scholarship Application

In [6]:
```
df_test_display = df_test.copy()
df_test_display["knn1"] = knn1_prediction
df_test_display["knn3"] = knn3_prediction
df_test_display
```

Out[6]:

	Sibs	Income	Grade	Approve	knn1	knn3
10	1	1.0	4.0	NaN	yes	no
11	0	3.0	4.0	NaN	no	no
12	1	1.0	2.5	NaN	no	no
13	5	1.0	2.5	NaN	yes	yes

Testing The Model (Application)

Iris Dataset

- Test set: 25% of the original dataset

  Variables

  - `iris_test_df` - features

In [8]:

```
knn1_prediction = knn1.predict(iris_test_df)
knn3_prediction = knn3.predict(iris_test_df)
```

Testing The Model (Application)

Iris Dataset

In [9]:
```
result_df = pd.DataFrame({"KNN1": knn1_prediction, "KNN3": knn3_prediction})
result_df.sample(frac = 0.2)
```

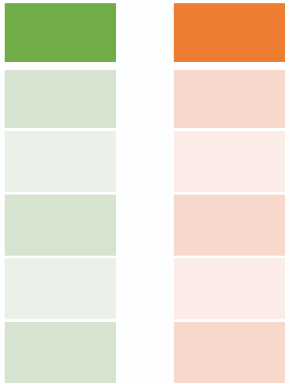
Out[9]:

	KNN1	KNN3
30	2	2
17	1	1
26	1	1
16	1	1
22	1	1
10	0	0
34	2	2
1	0	0

Overview

RESULT EVALUATION

PREDICTED LABELS VS TRUE LABELS



$$\text{accuracy} = \frac{\#\ \text{correct predictions}}{\#\ \text{total predictions}}$$

predicted label

true label		YES	NO
	YES		
	NO		

Evaluate Results

Evaluation

- We can evaluate results if we have the true labels for the test set

Evaluation Method

- Accuracy
- Confusion Matrix

Testing The Model (Application)

Scholarship Application

- No way to verify if predicted label is correct with a fabricated dataset (no “ground truth”)

Iris Dataset

- We can evaluate results through:
    - the model’s `score()` method: returns the accuracy of results
    - sklearn’s `confusion_matrix()` function

Testing The Model (Application)

Iris Dataset

- `model_name.score(features, true_labels)`

```
In [7]:  acc1 = knn1.score(iris_test_df, iris_test_y)
          acc3 = knn3.score(iris_test_df, iris_test_y)
```

- `confusion_matrix(labels, predicted_labels)`

```
In [8]:  cm1 = confusion_matrix(iris_test_y,knn1_prediction)
          cm3 = confusion_matrix(iris_test_y,knn3_prediction)
```

Testing The Model (Application)

Iris Dataset

```
print("kNN, k = 1 accuracy:",acc1)
print(cm1)

print("kNN, k = 3 accuracy:",acc3)
print(cm3)
```

```
kNN, k = 1 accuracy: 0.9210526315789473
[[13  0  0]
 [ 0 13  1]
 [ 0  2  9]]
kNN, k = 3 accuracy: 0.9210526315789473
[[13  0  0]
 [ 0 13  1]
 [ 0  2  9]]
```

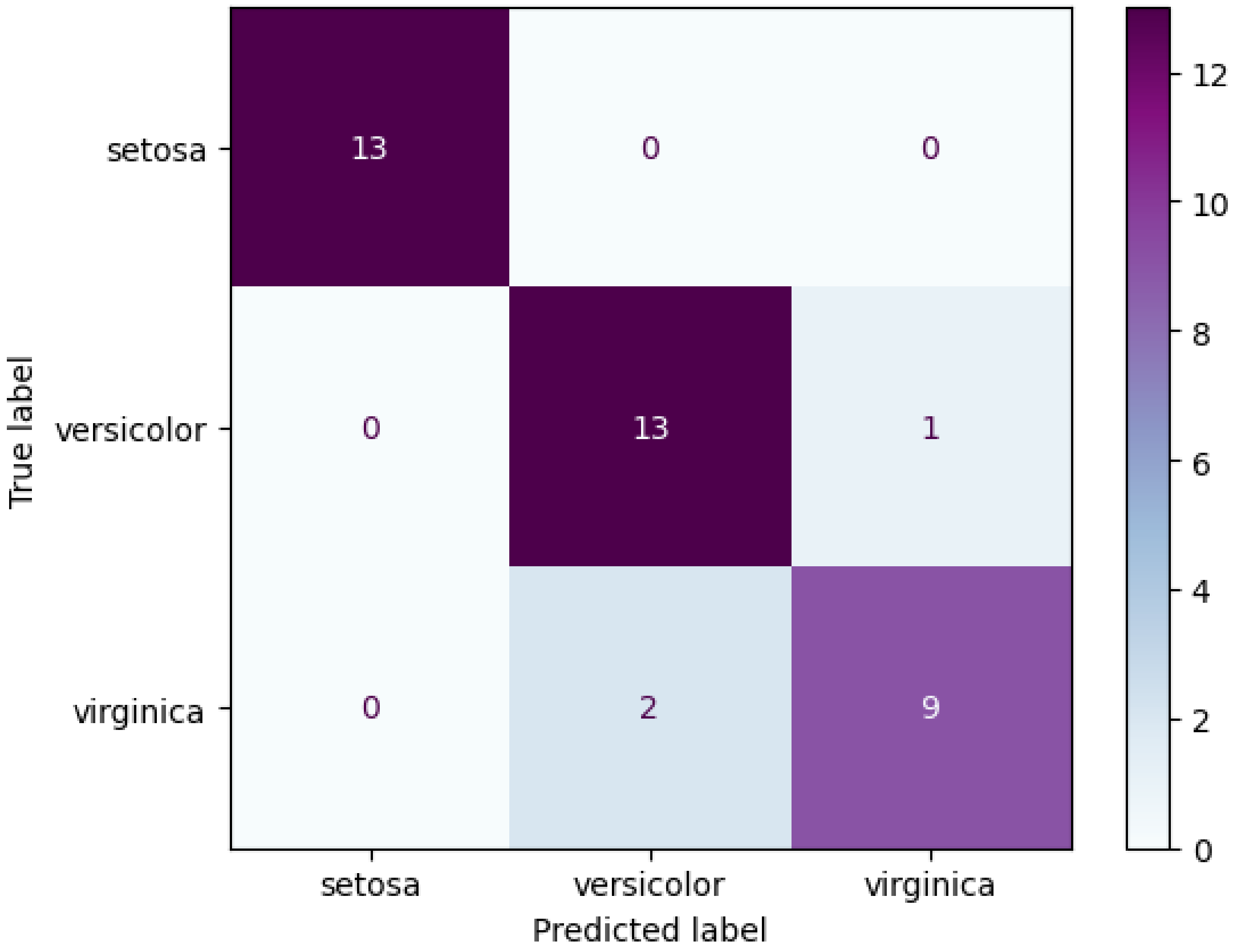
In this run, kNN with k=1 and k=3 yield the same results.

Note: the train and test set in this slides' example are generated through random sampling, so the results will be different when the code is run again because the test set is different

Testing The Model (Application)

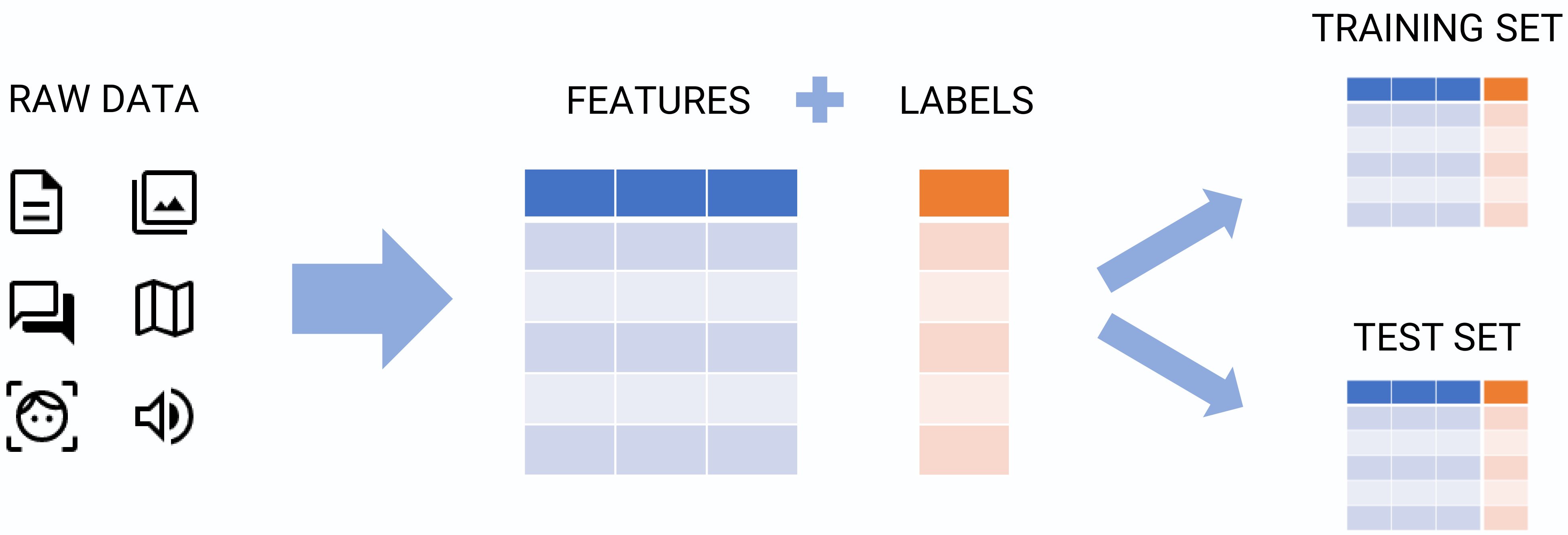
Iris Dataset

- Plot uses matplotlib and sklearn
- Labels show the three Iris species being classified (setosa, versicolor, virginica)



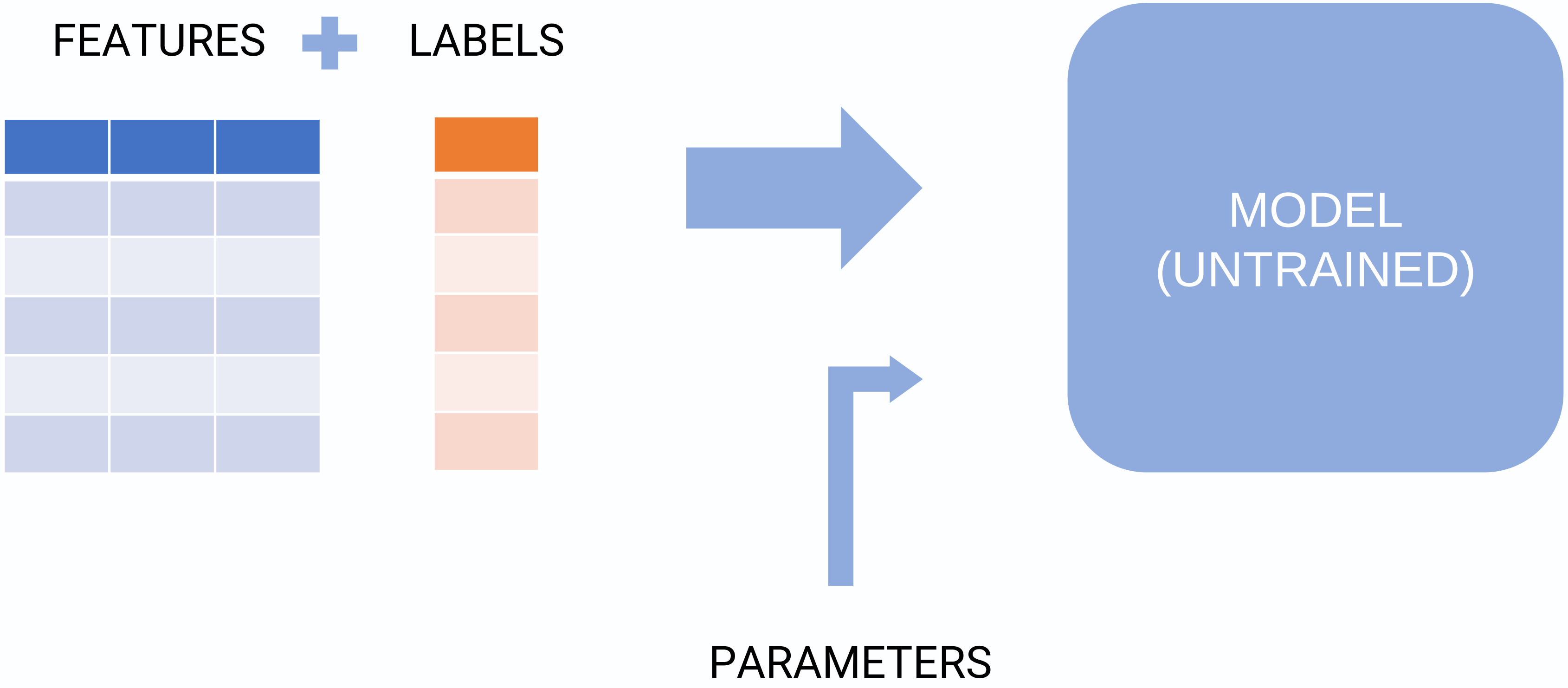
Overview

DATA PREPARATION



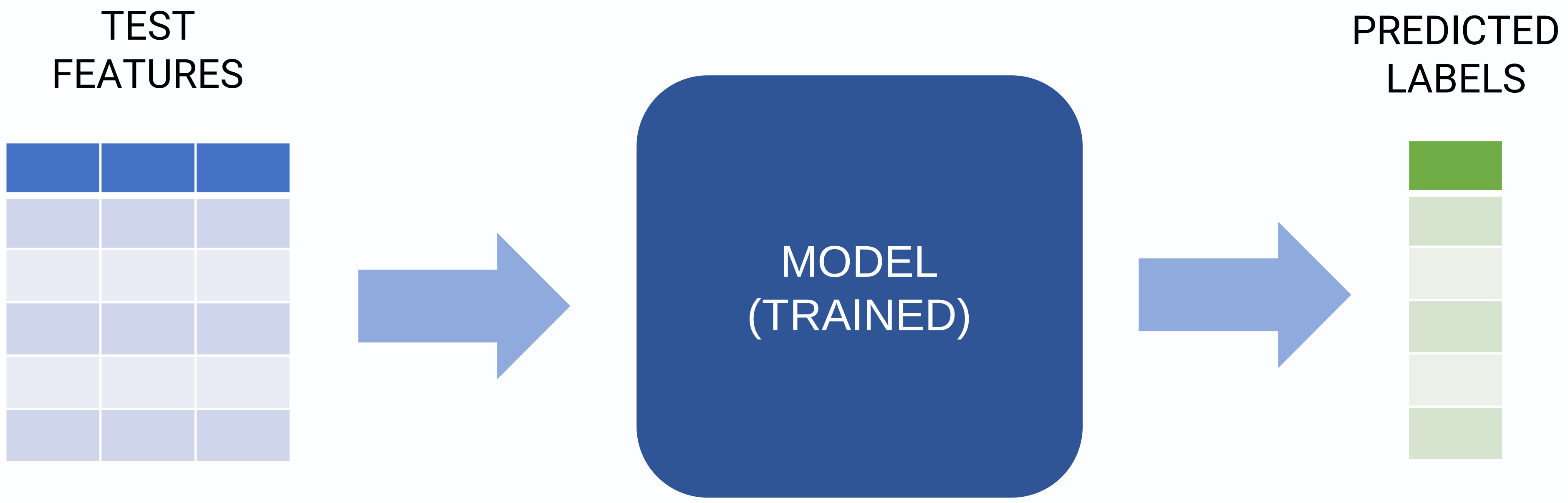
Overview

MODEL TRAINING



Overview

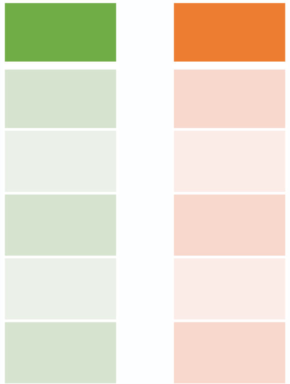
MODEL TESTING



Overview

RESULT EVALUATION

PREDICTED LABELS VS TRUE LABELS



$$\text{accuracy} = \frac{\#\ \text{correct predictions}}{\#\ \text{total predictions}}$$

predicted label

true label

	YES	NO
YES		
NO		

Conclusion

The Black Box

- We can follow the same steps, but using a different classifier from sklearn or other packages
- Can easily compare one method to another and evaluate what works best with the dataset

```
gnb = GaussianNB()
gnb.fit(iris_train_df, iris_train_y)
```

```
clf = tree.DecisionTreeClassifier()
clf = clf.fit(iris_train_df, iris_train_y)
```