A Light Into. to Scikit-Learn

Rina BUOY, PhD

ChatGPT 4.0

Scikit-Learn

- Scikit-Learn is characterized by a clean, uniform, and streamlined API, as well as by very useful and complete online documentation.

- A benefit of this uniformity is that once you understand the basic use and syntax of Scikit-Learn for one type of model, switching to a new model or algorithm is very straightforward.

ML Data as a Table

one sample

$$X = \begin{pmatrix} 1.1 & 2.2 & 3.4 & 5.6 & 1.0 \\ 6.7 & 0.5 & 0.4 & 2.6 & 1.6 \\ 2.4 & 9.3 & 7.3 & 6.4 & 2.8 \\ 1.5 & 0.0 & 4.3 & 8.3 & 3.4 \\ 0.5 & 3.5 & 8.1 & 3.6 & 4.6 \\ 5.1 & 9.7 & 3.5 & 7.9 & 5.1 \\ 3.7 & 7.8 & 2.6 & 3.2 & 6.3 \end{pmatrix} \qquad y = \begin{pmatrix} 1.6 \\ 2.7 \\ 4.4 \\ 0.5 \\ 0.2 \\ 5.6 \\ 6.7 \end{pmatrix}$$

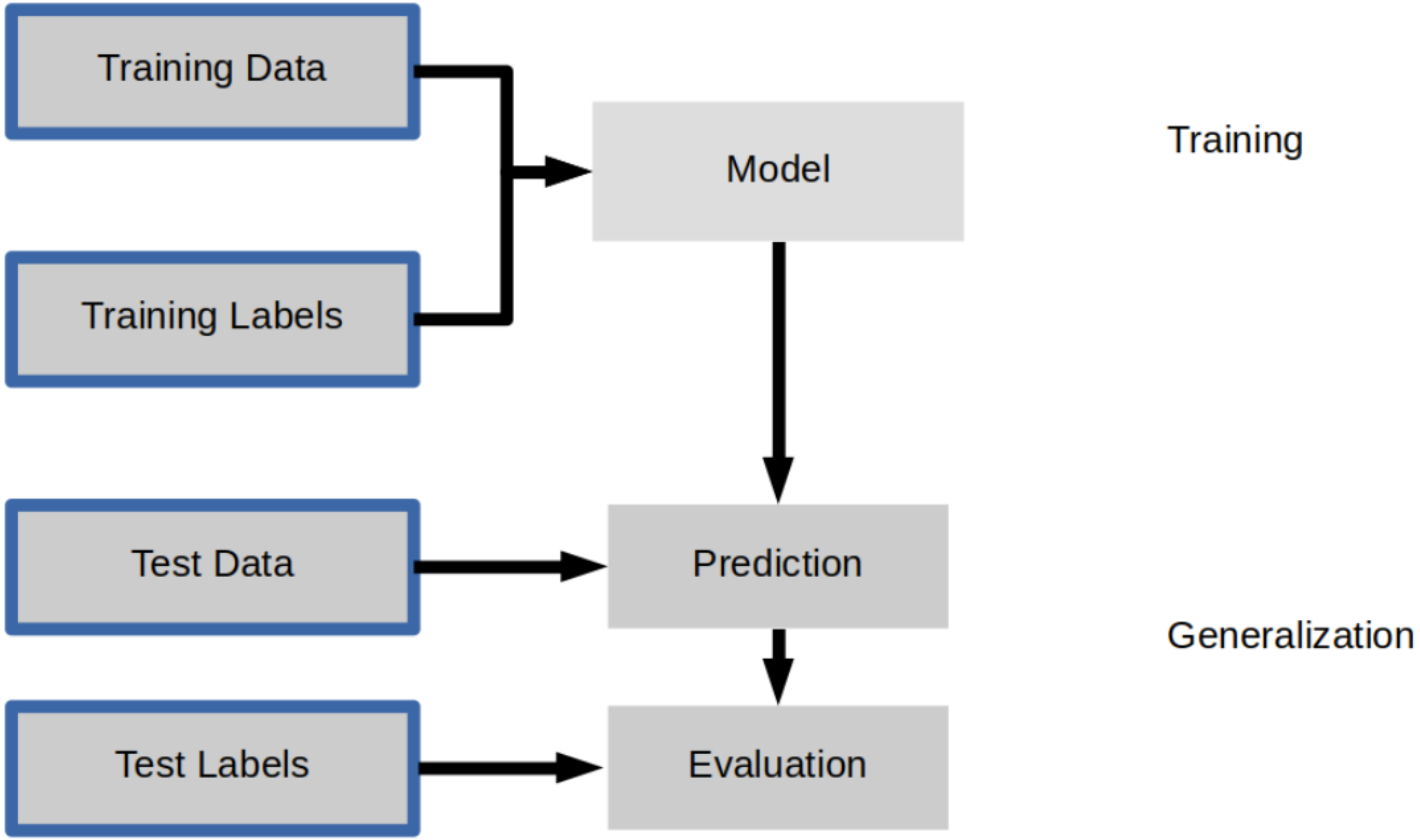
Features matrix

Target vector

one feature

outputs / labels

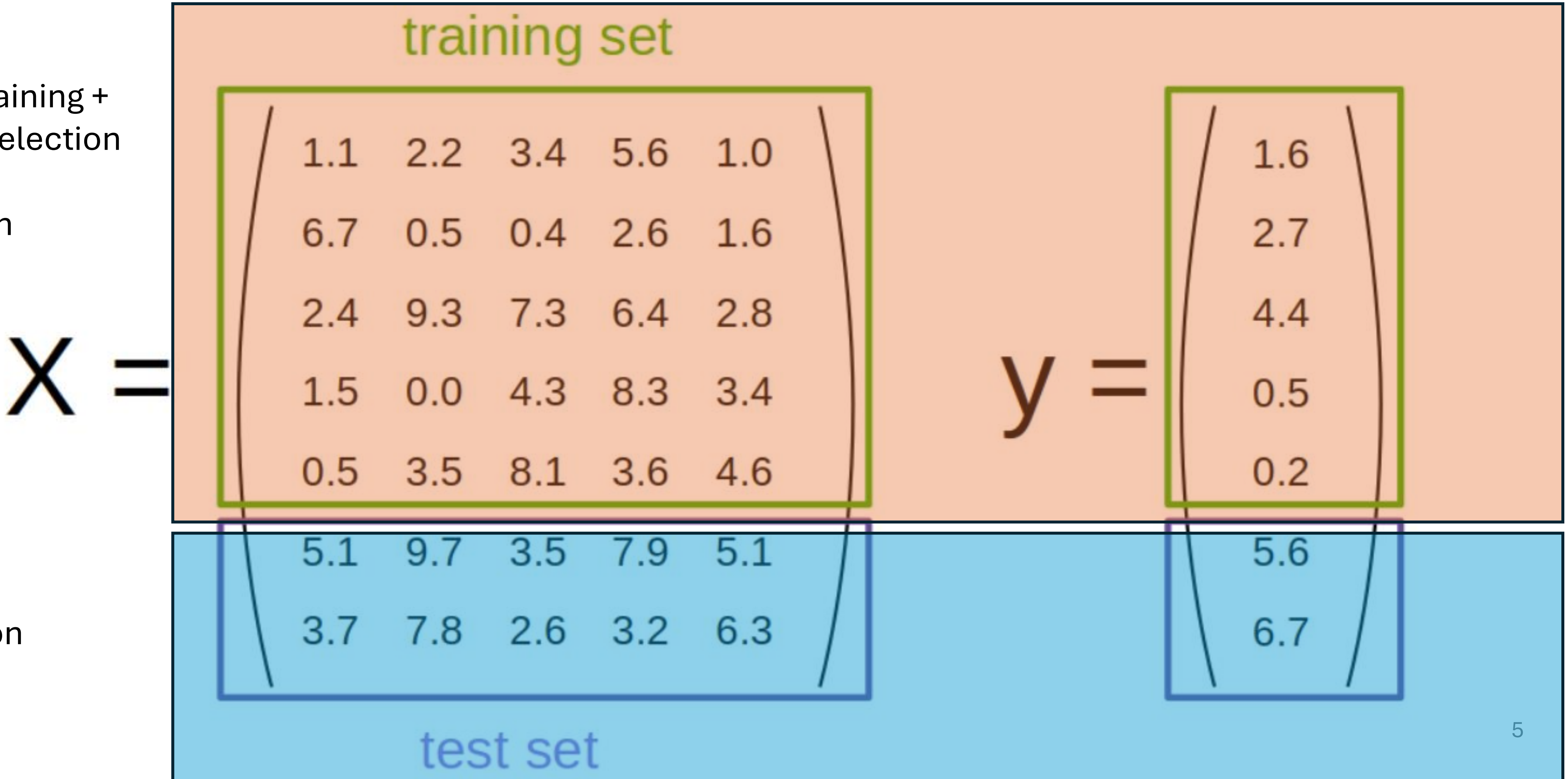
ML Workflow


Training Data
Training Labels
Model
Training
Test Data
Prediction
Generalization
Test Labels
Evaluation

Train-Test Sets

Model Training + Setting Selection (require validation dataset)

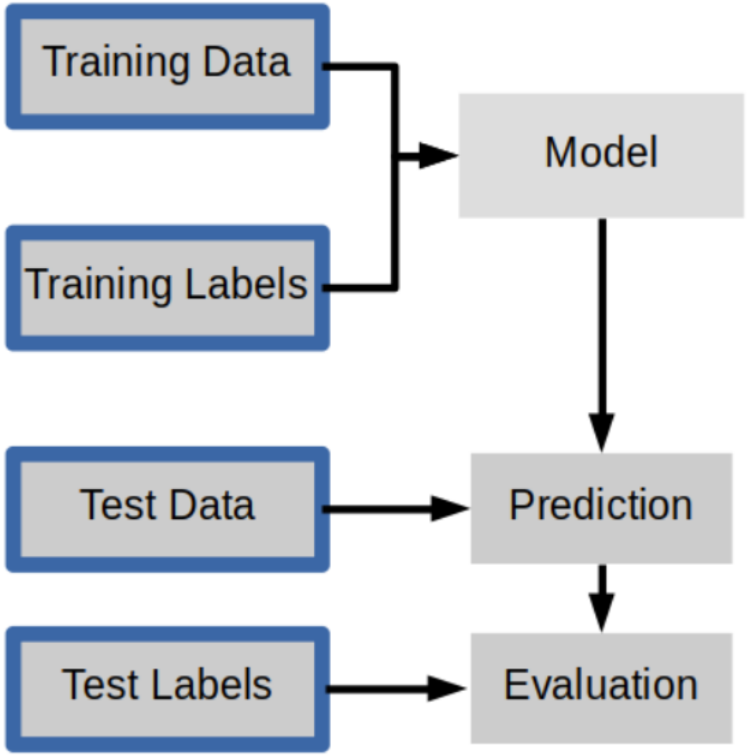
Only for evaluation



Scikit-Learn API

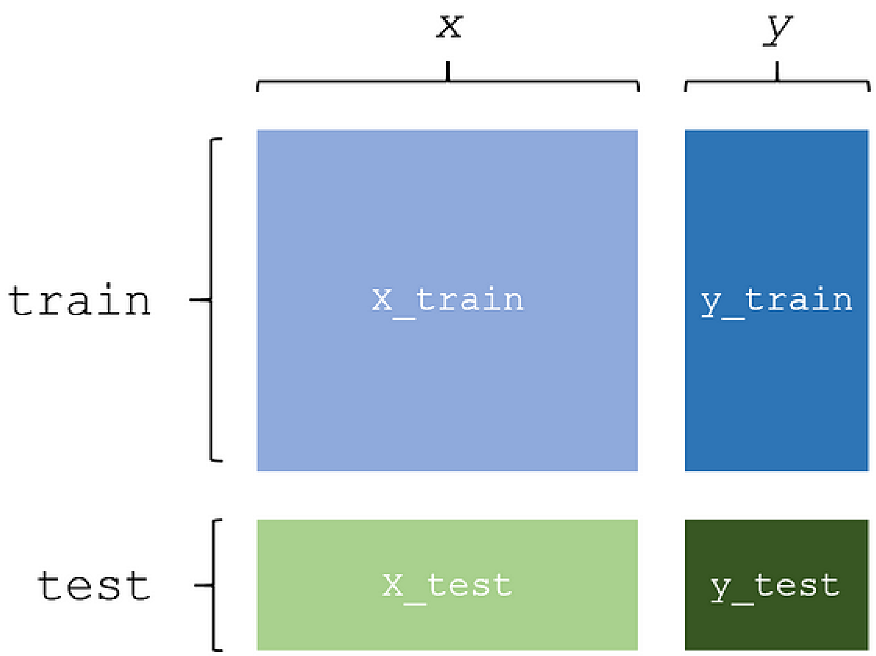
From scikitlearn import Model

```
clf = Model()

clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

clf.score(X_test, y_test)
```



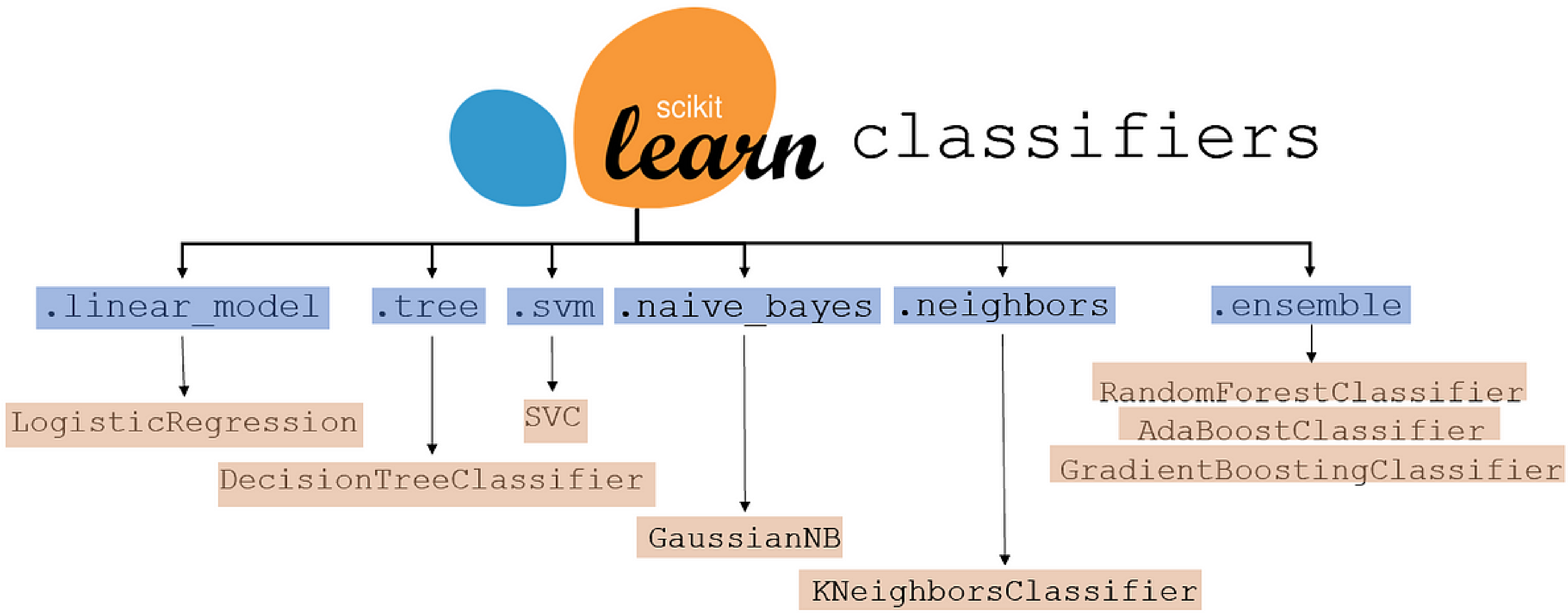
Train-test-split

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3)
```

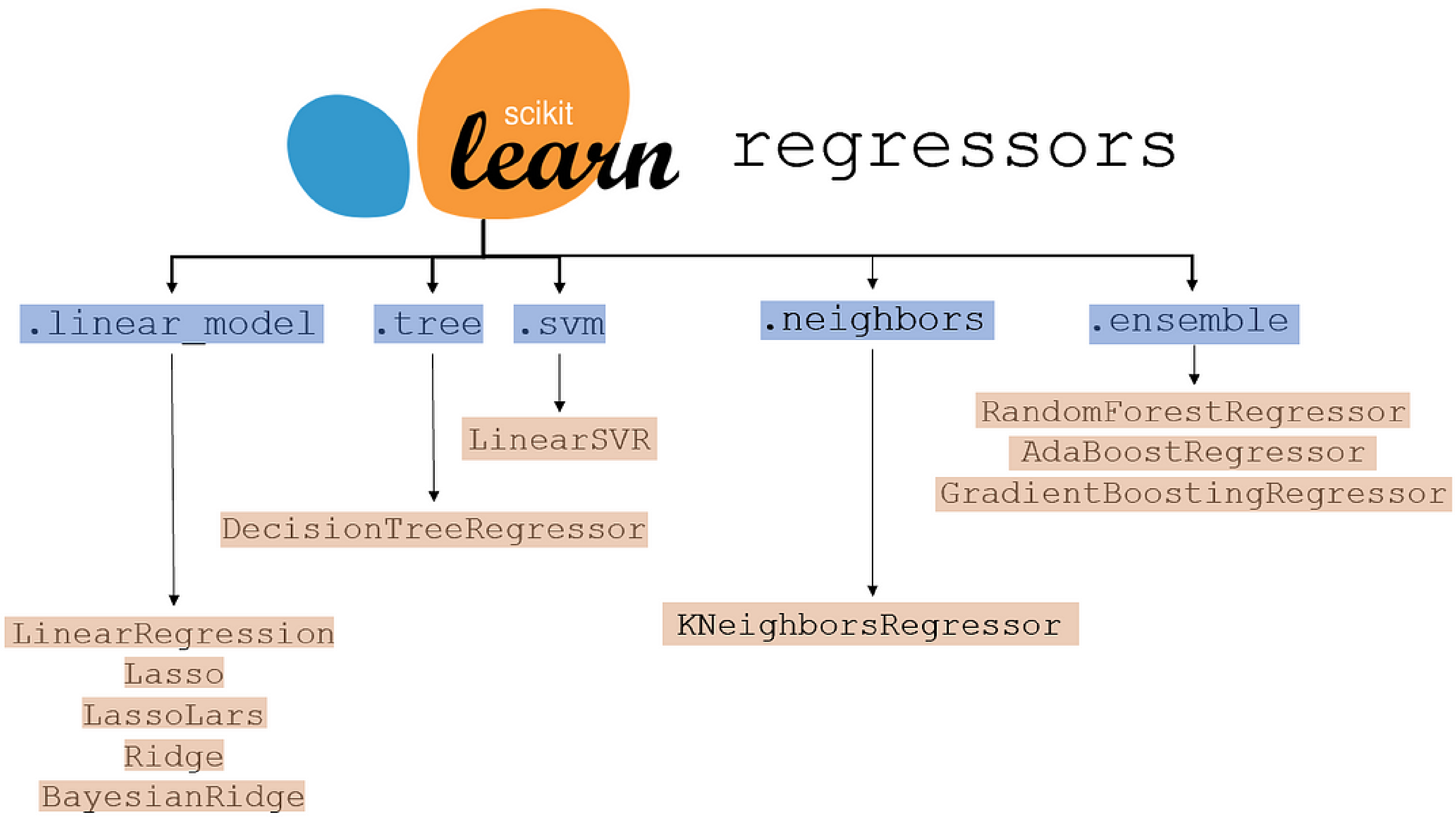


https://www.kdnuggets.com/2021/01/ultimate-scikit-learn-machine-learning-cheatsheet.html

Scikitlearn Classifiers



Scikitlearn Regressors

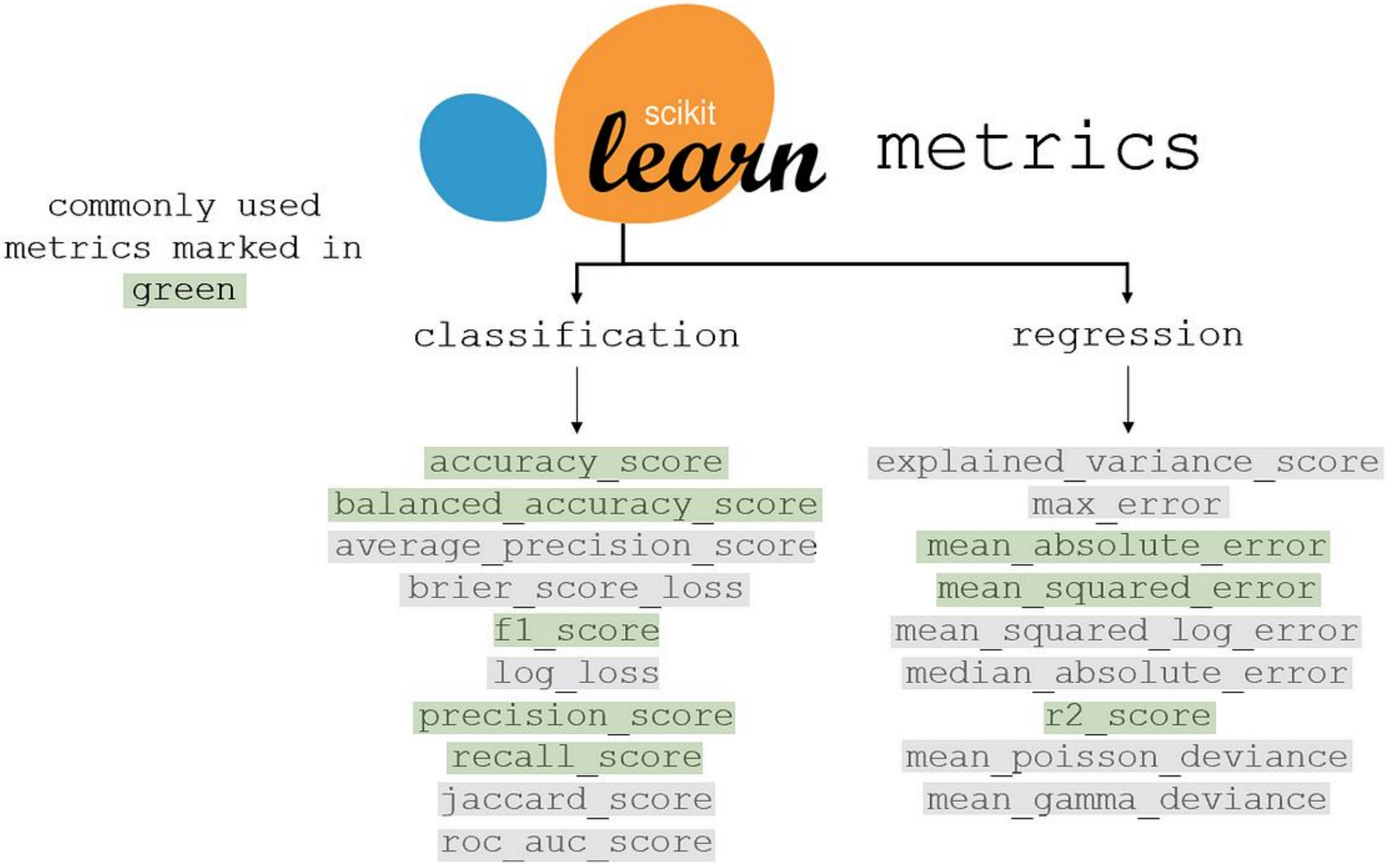


Evaluating Model Performance

```
#import
from sklearn.metrics import metric_name

#create instance
metric_name(y_test, model.predict(X_test))
```

real target   predicted target

Evaluating Model Performance



Data Transformation

- Standardizing or scaling is the process of ‘reshaping’ the data such that it contains the same information but has a mean of 0 and a variance of 1. By scaling the data, the mathematical nature of algorithms can usually handle data better.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(data)
transformed_data = scaler.transform(data)
```

```
data = scaler.inverse_transform(output_data)
```

Data Transformation

- Normalizing data puts it on a 0 to 1 scale, something that, similar to standardized data, makes the data mathematically easier to use for the model.

```
from sklearn.preprocessing import Normalizer
normalize = Normalizer()
transformed_data = normalize.fit_transform(data)
```

Data Transformation

- Box-cox transformations involve raising the data to various powers to transform it. Box-cox transformations can normalize data, make it more linear, or decrease the complexity.
- sklearn automatically determines the best series of box-cox transformations to apply to the data to make it better resemble a

```
from sklearn.preprocessing import PowerTransformer
transformer = PowerTransformer(method='box-cox')
transformed_data = transformer.fit_transform(data)
```

Dealing with Discreet Inputs

- Nominal variables: These variables represent categories without any inherent order or ranking. Examples include colors, types of fruits, or gender.
- Ordinal variables: These variables have a natural ordering or ranking among the categories. For example, ratings such as "low," "medium," and "high" represent ordinal variables.

One-Hot Encoding

id	color
1	red
2	blue
3	green
4	blue

One Hot Encoding

id	color_red	color_blue	color_green
1	1	0	0
2	0	1	0
3	0	0	1
4	0	1	0

```python
from sklearn.preprocessing import OneHotEncoder

# Sample data
data = [['red'], ['green'], ['blue']]

# Initialize the encoder
encoder = OneHotEncoder()

# Fit and transform the data
encoded_data = encoder.fit_transform(data).toarray()

# Print the encoded data
print(encoded_data)
```

Ordinal Encoding

Original Encoding	Ordinal Encoding
Poor	1
Good	2
Very Good	3
Excellent	4

```
from sklearn.preprocessing import OrdinalEncoder

# Sample data
data = [['low'], ['medium'], ['high']]

# Initialize the encoder
encoder = OrdinalEncoder(categories=[['low', 'medium', 'high']])

# Fit and transform the data
encoded_data = encoder.fit_transform(data)

# Print the encoded data
print(encoded_data)
```

Hyperparameter Selection

- Scikit-learn's GridSearchCV is an excellent tool for tuning hyperparameters of machine learning models. It exhaustively searches through a specified grid of hyperparameters to find the best ones based on cross-validated performance.

```
from sklearn.model_selection import GridSearchCV
parameters = {‘param': [0.1, 0.01, 0.001] }
model = Model()
gs = GridSearchCV(model, parameters, cv=5)
gs.fit(X_train, y_train)

best_model = gs.best_estimator_
best_model.fit(X_train, y_train)
```

Example of Complex Search

Parameters to search

```python
# number of trees
n_estimators=[500, 800, 1500, 2500, 5000]
# max number of features to consider at every split
max_features = ['auto', 'sqrt', 'log2']
# max number of levels in tree
max_depth = [10, 20, 30, 40, 50]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10, 15, 20]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 5, 10, 15]
```

```python
grid_param = {'n_estimators': n_estimators,
              'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf}
```

Pipeline

- Scikit-learn's Pipeline is a tool for chaining multiple estimators into one. This is useful when there are a series of steps in your machine learning workflow that need to be executed in a particular sequence, such as data preprocessing, feature engineering, and model training.

```python
# Define the steps in the pipeline
steps = [
    ('scaler', StandardScaler()),   # Step 1: Data preprocessing (scaling)
    ('pca', PCA()),                 # Step 2: Feature reduction (PCA)
    ('svm', SVC())                  # Step 3: Model training (Support Vector Machine)
]

# Create the pipeline
pipeline = Pipeline(steps)

# Now you can use the pipeline as a single estimator
pipeline.fit(X_train, y_train)
```

For more

https://scikit-learn.org/stable/index.html