

Outline

- What Is Machine Learning?
- Introducing Scikit-Learn
 - Supervised learning example
 - Simple linear regression
 - Iris classification
 - Unsupervised learning example
 - Iris dimensionality
 - Iris clustering
- Hyperparameters and Model Validation
 - Handout
 - Cross-Validation
- Selecting the Best Model
 - The bias–variance trade-off
 - Validation Curves
 - Learning Curves

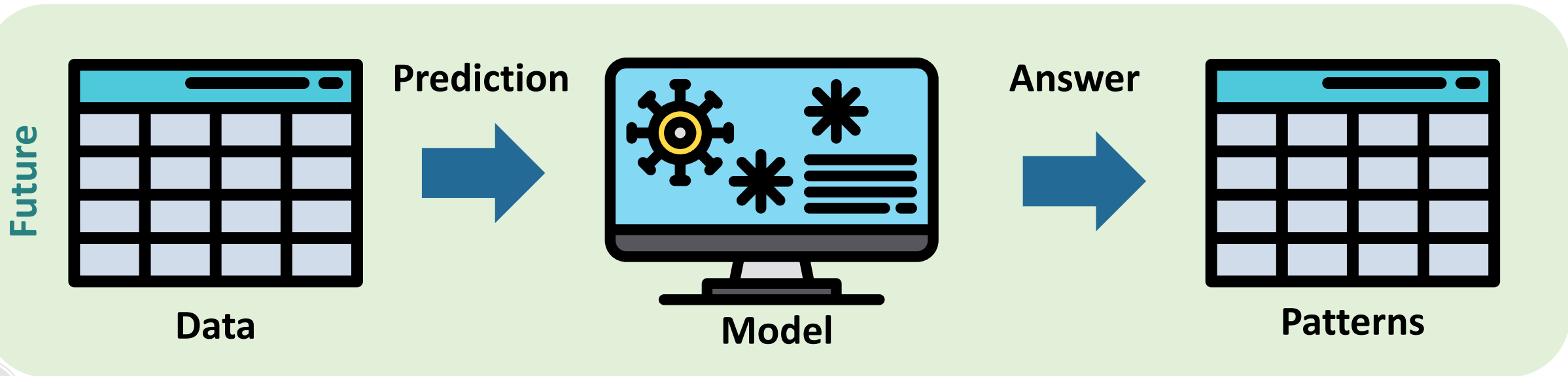
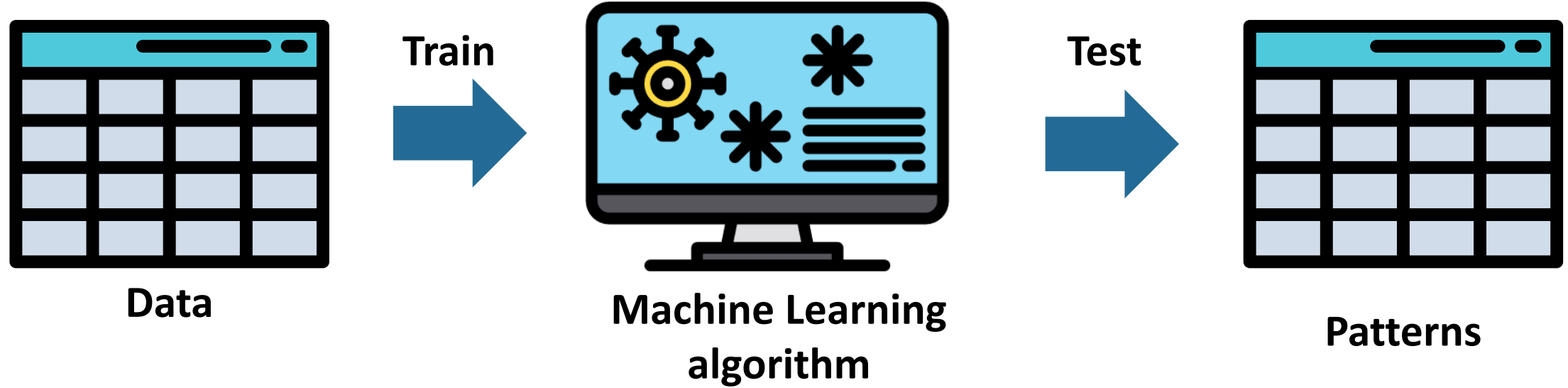


What Is Machine Learning?

What Is Machine Learning?

- Machine learning is a method of data analysis that automates analytical model building.
- It is a branch of artificial intelligence based on the idea that systems can learn from data, identify patterns and make decisions with minimal human intervention.

What Is Machine Learning?



Steps in a full ML project

Steps in a full machine learning project



Goals of this Session

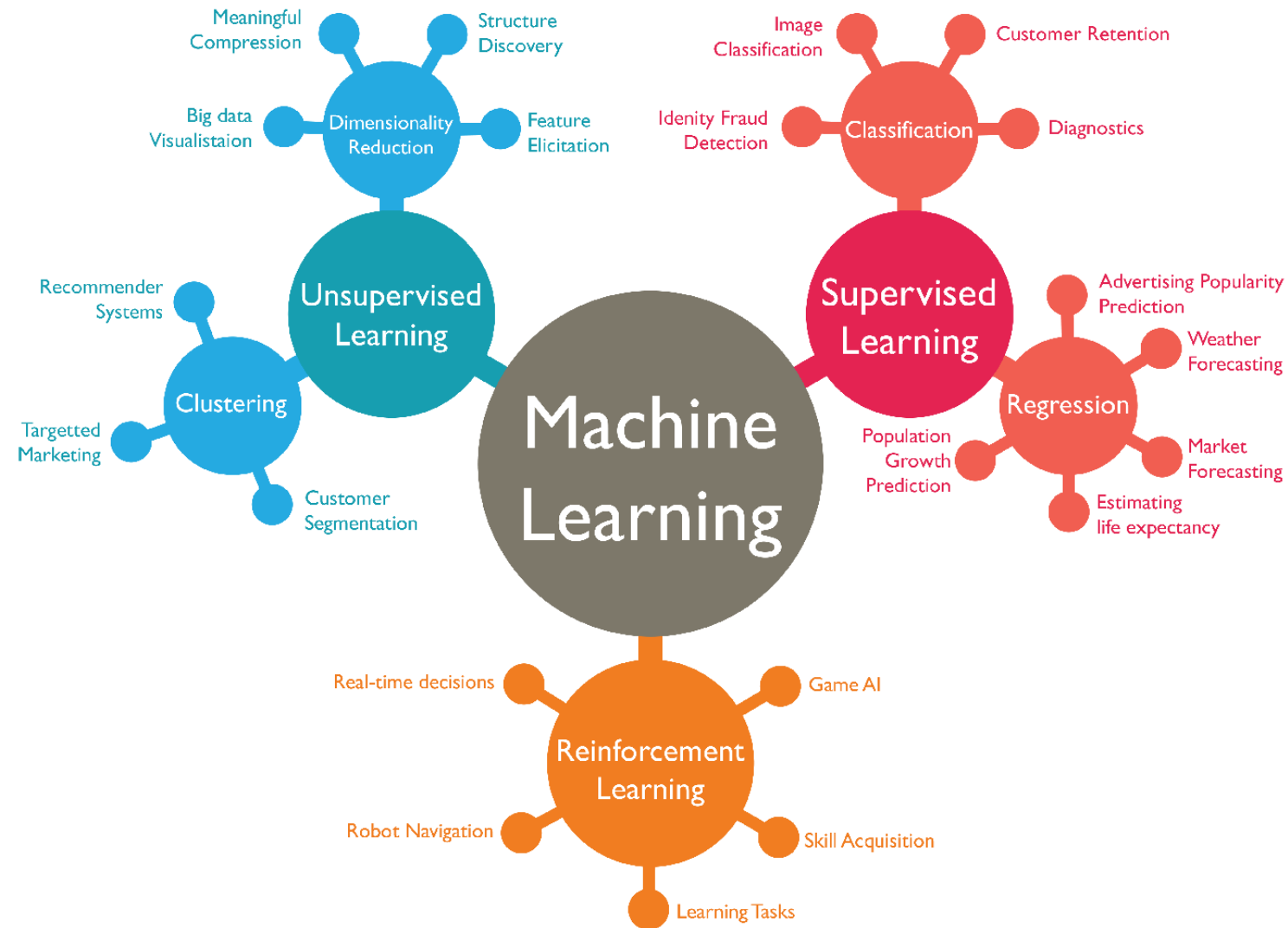


TO INTRODUCE THE **FUNDAMENTAL**
VOCABULARY AND CONCEPTS OF MACHINE
LEARNING.



TO INTRODUCE THE **SCIKIT-LEARN** API AND
SHOW SOME EXAMPLES OF ITS USE

Categories of Machine Learning



Our Focus



Supervised learning

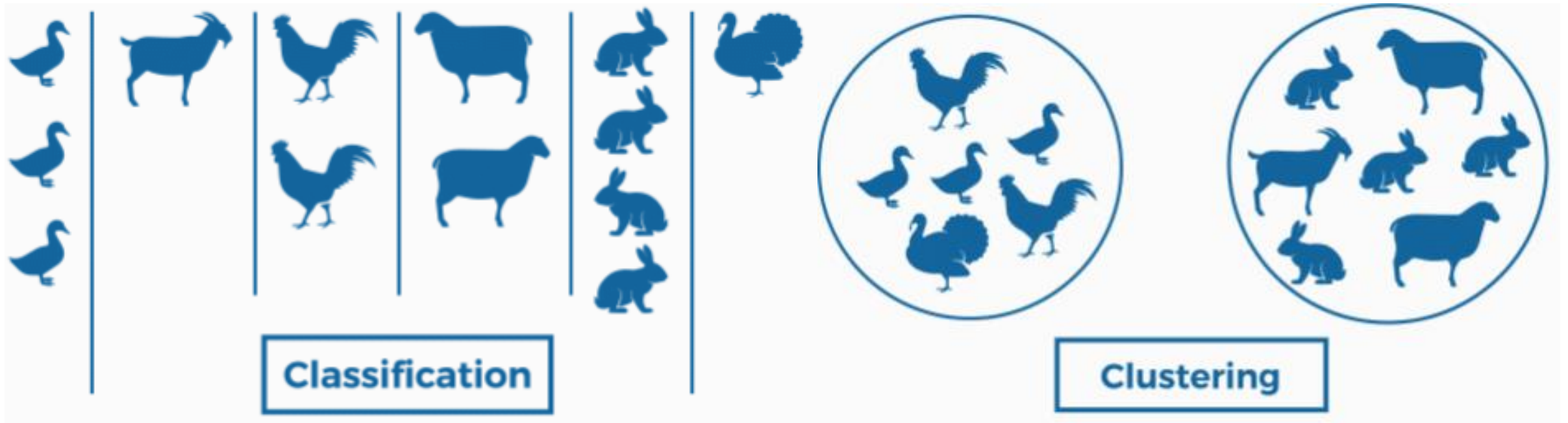


Unsupervised learning

How could we categorise these animals?

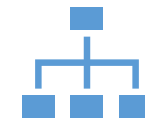


How could we categorise these animals?



Supervised learning

- Learning is guided by an instructor. The data set is **labelled**.
- Supervised learning involves somehow modelling the relationship between measured **features** of **data** and some label associated with the data.
- Subdivided into
 - *Classification*
 - the labels are discrete categories
 - *Regression*
 - the labels are continuous quantities.



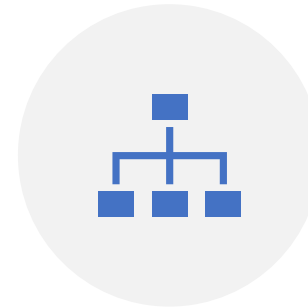
Classification



Regression

Unsupervised learning

- The model learns through perception and discovers structures in the information. The data set is **not labelled**.
- “letting the dataset speak for itself.”
- Subdivided into
 - *Clustering*
 - identify distinct groups of data
 - *Dimensionality reduction*
 - search for more succinct representations of the data.

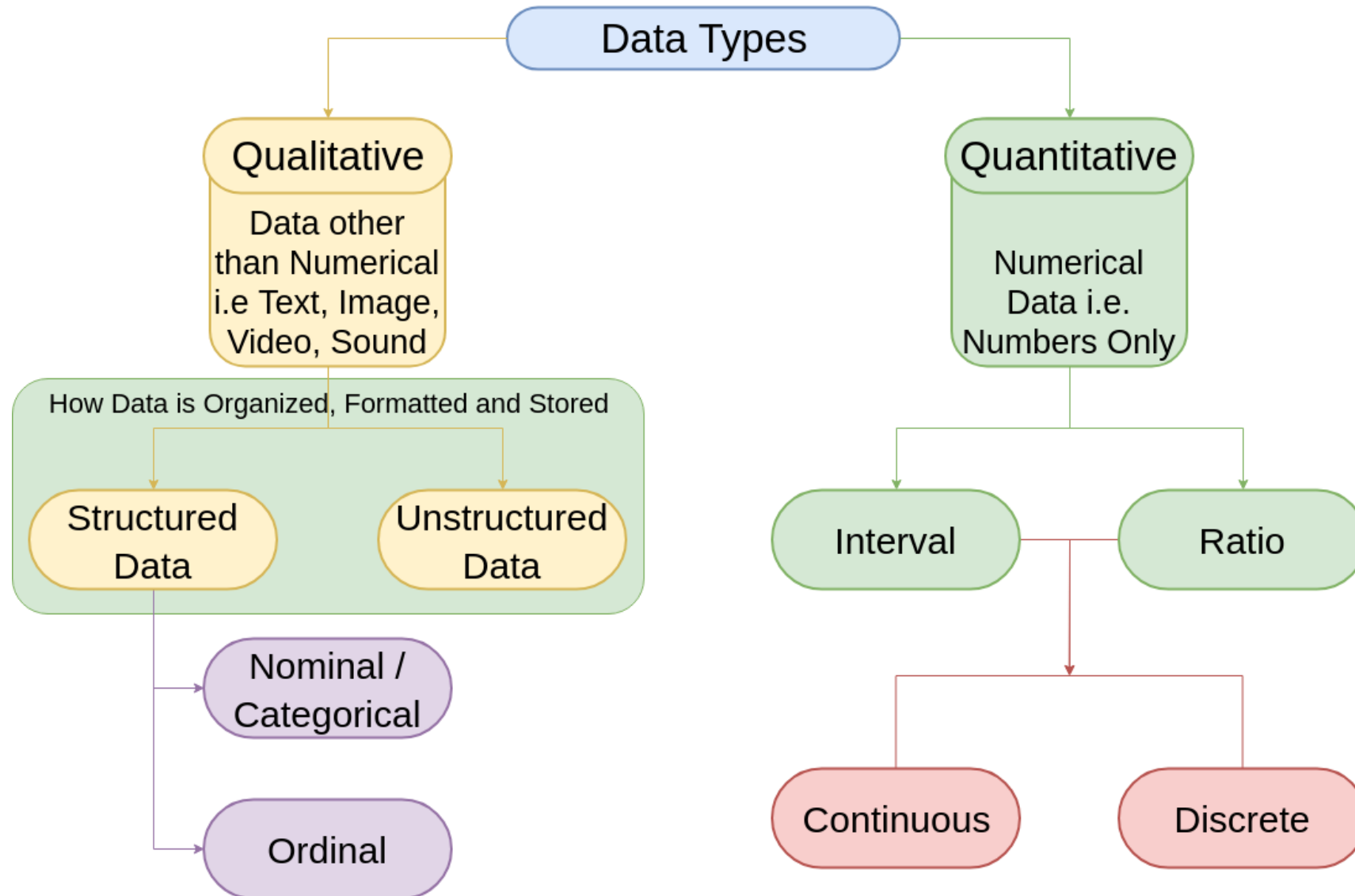


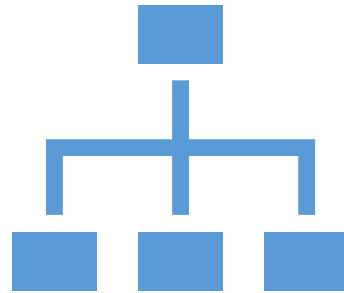
CLUSTERING



DIMENSIONALITY
REDUCTION

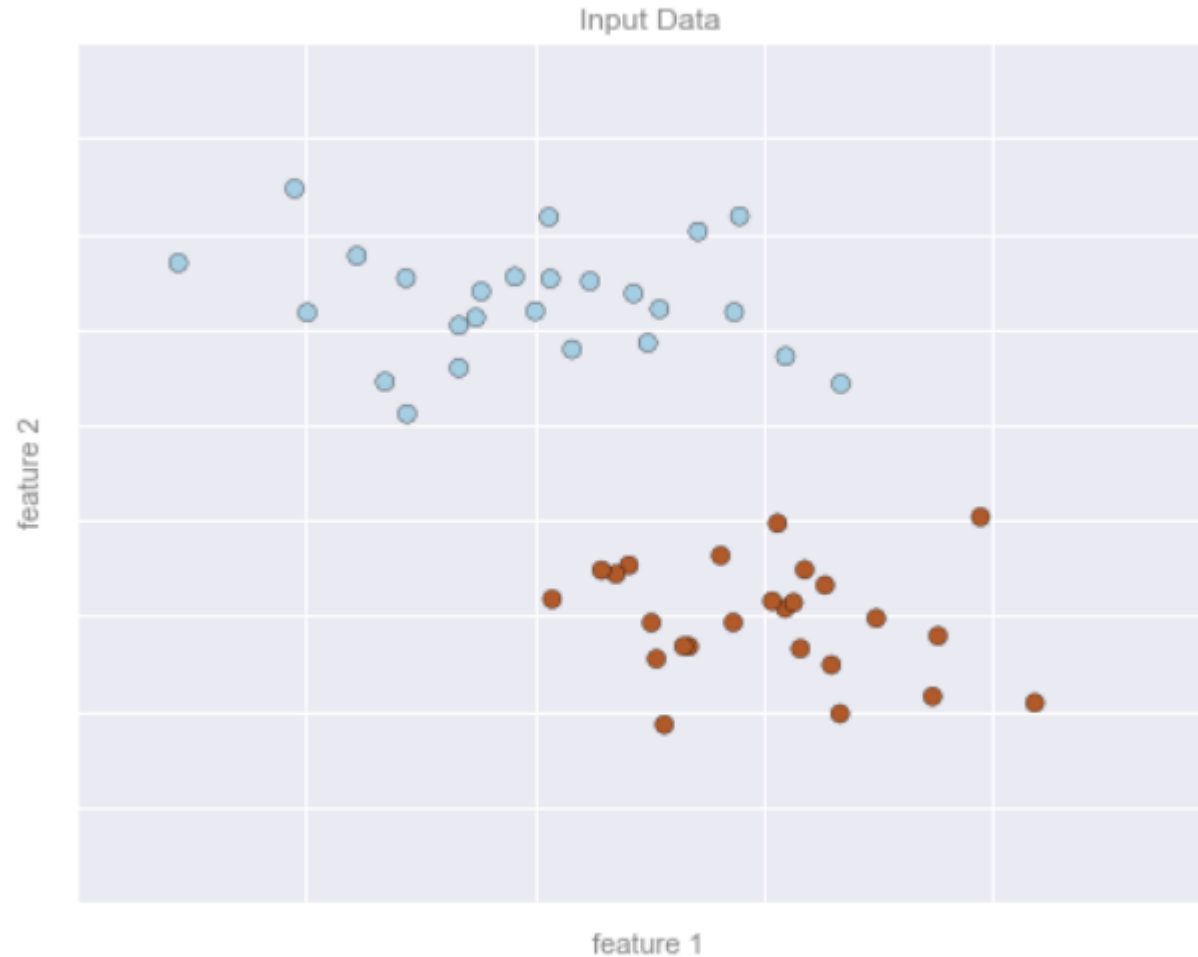
Machine Learning Data Types





Classification

Classification: Predicting discrete labels



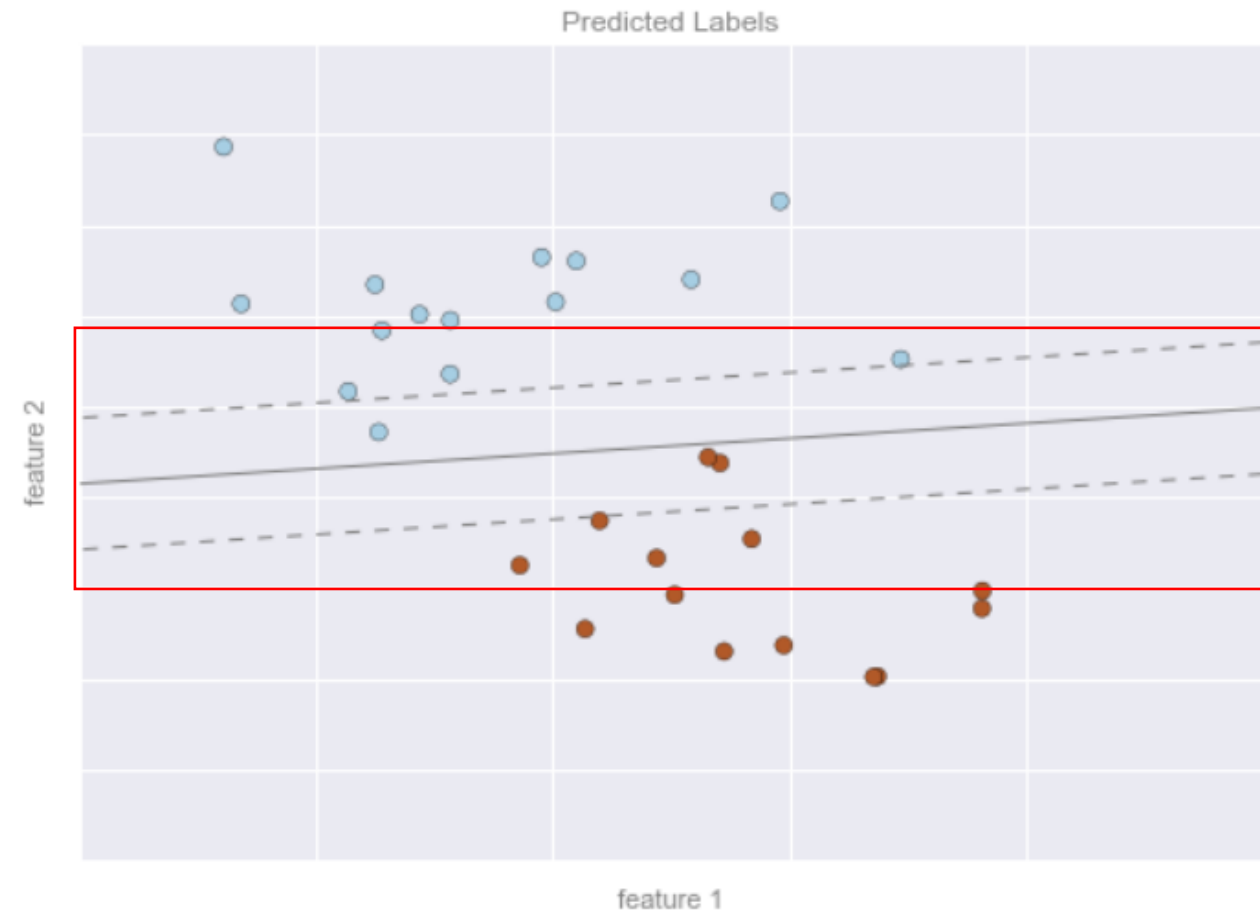
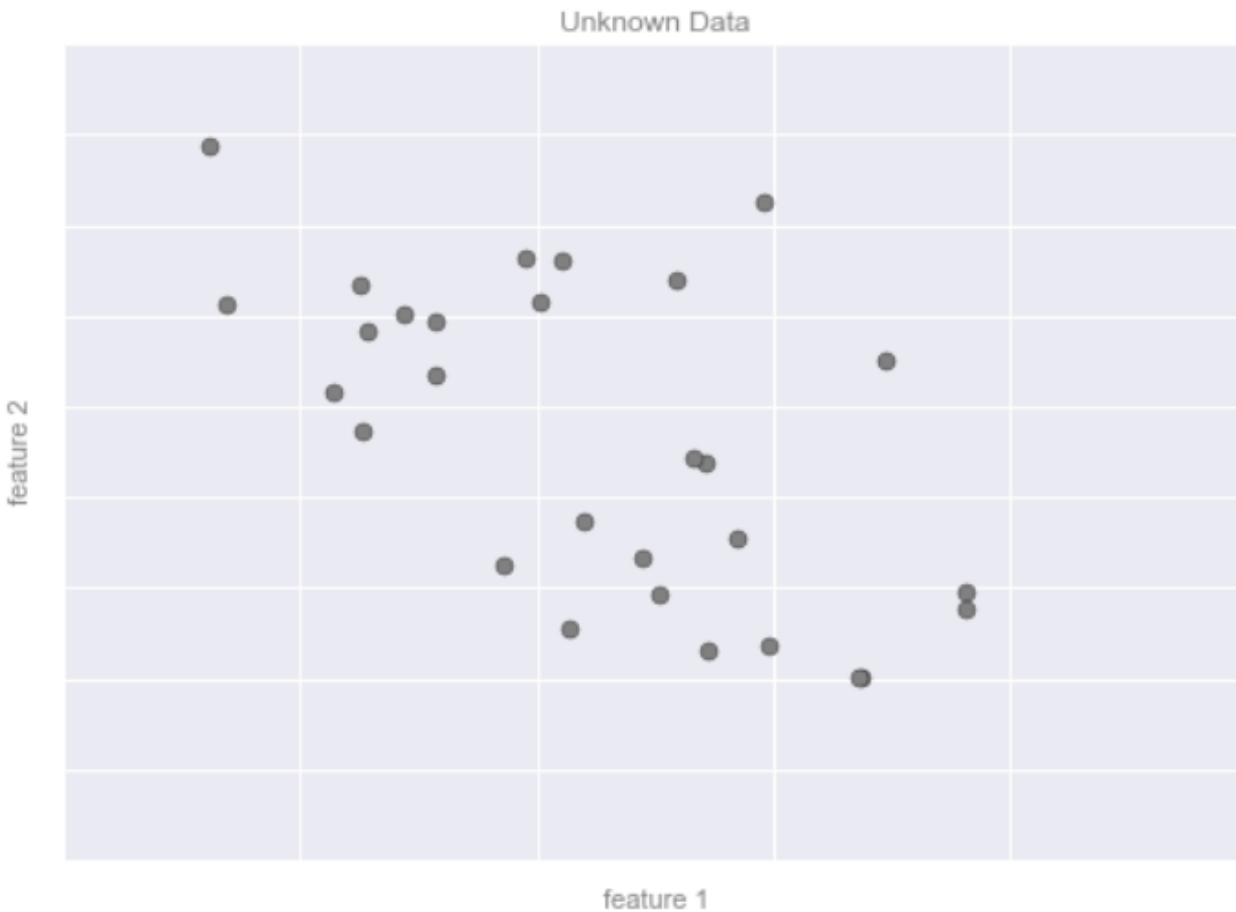
A simple data set for classification

Classification: Predicting discrete labels



A simple classification model

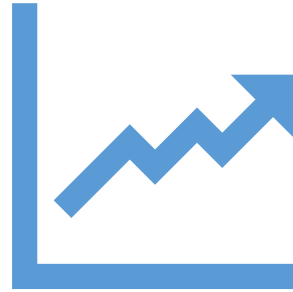
Classification: Predicting discrete labels



Applying a classification model to new data

Classification: Predicting discrete labels

- Example of Classification Algorithms
 - Naive Bayes Classification
 - Support Vector Machines
 - Decision Trees and Random Forests
- Example of Classification Applications
 - Automated spam detection for email
 - Identity Fraud Detection
 - Image Classification
 - Diagnostics



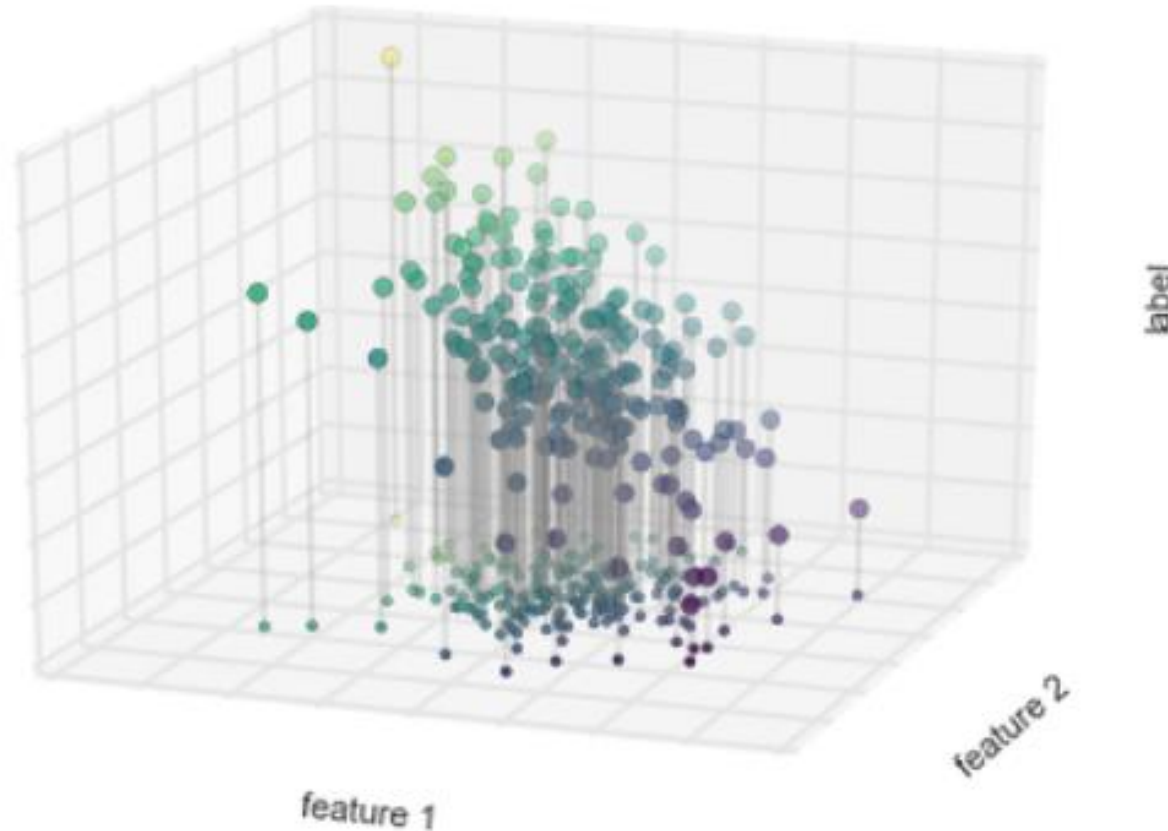
Regression

Regression: Predicting continuous labels



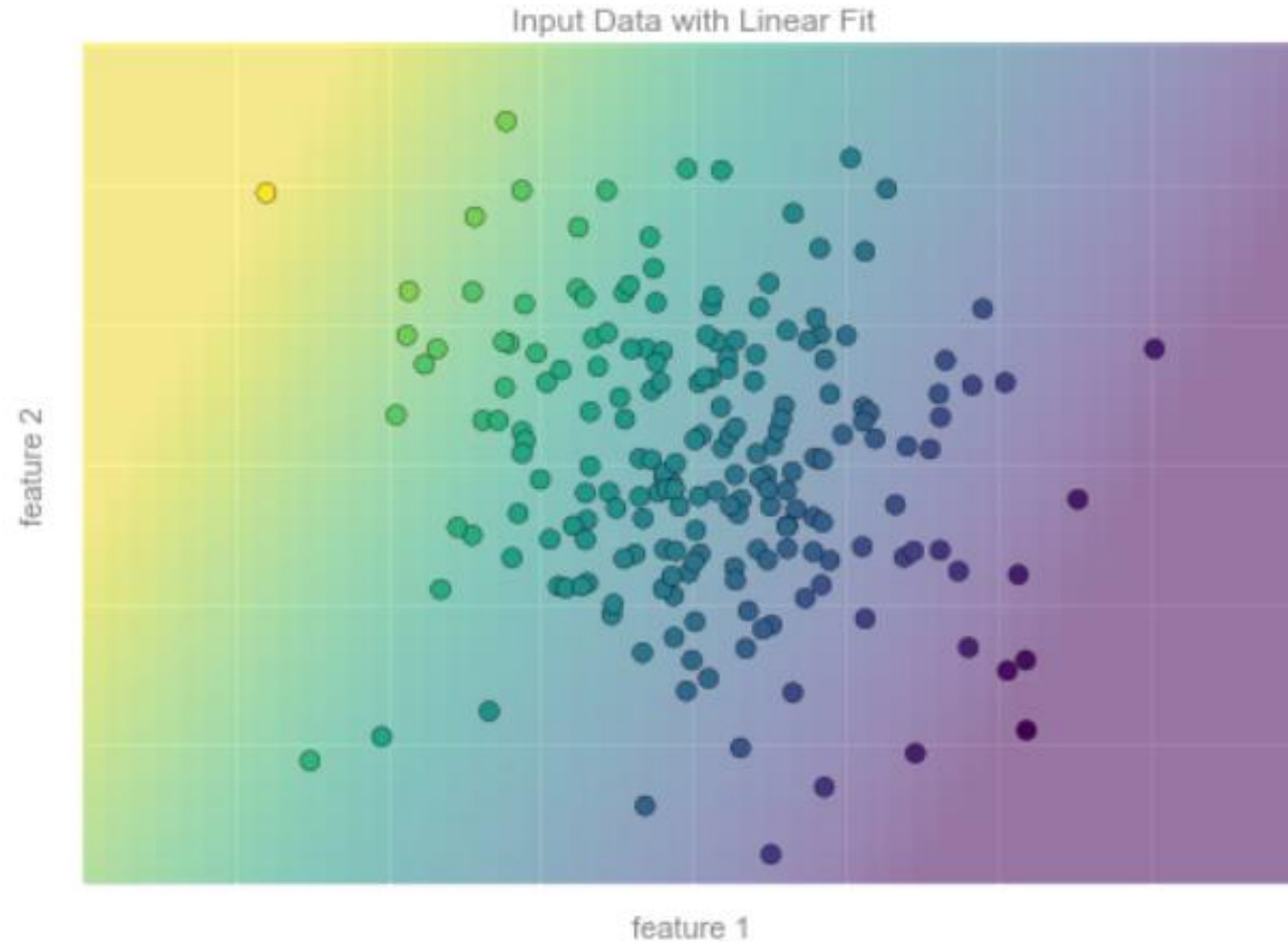
A simple dataset for regression

Regression: Predicting continuous labels



A three-dimensional view of the regression data

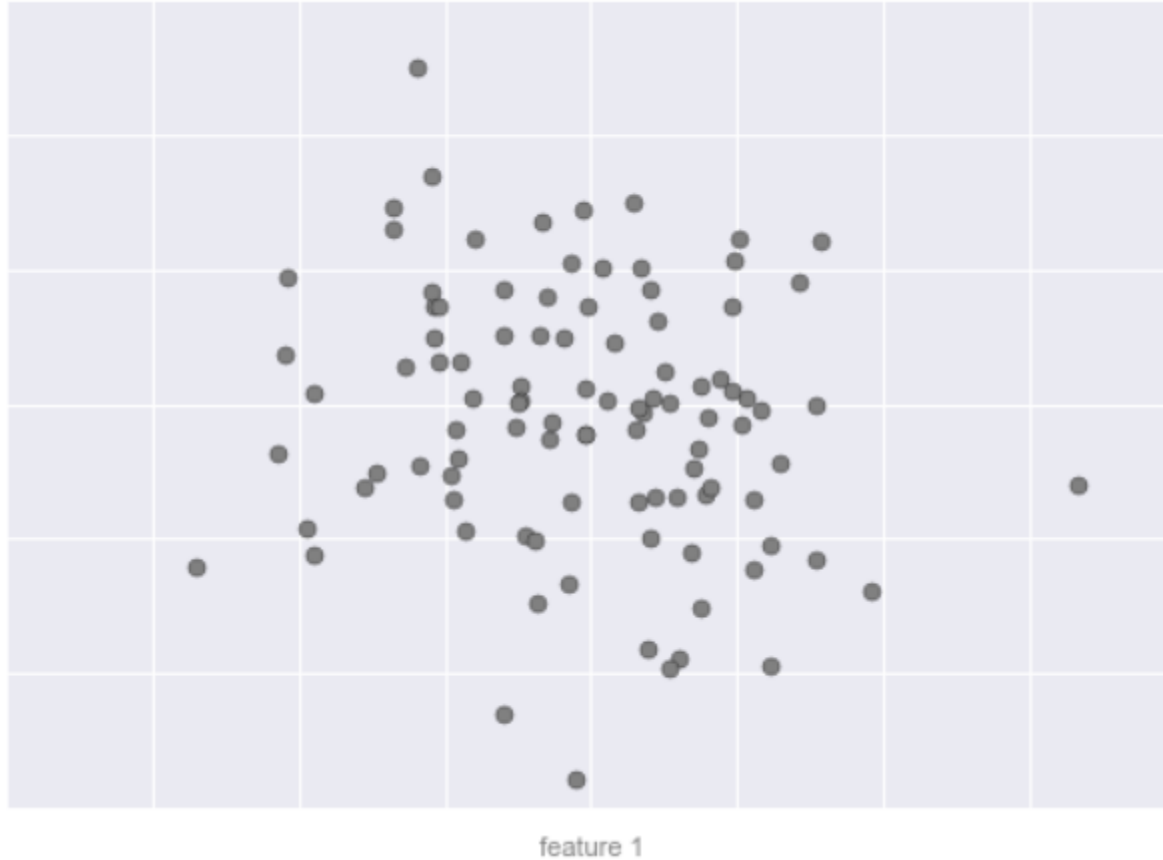
Regression: Predicting continuous labels



A representation of the regression model

Regression: Predicting continuous labels

Unknown Data



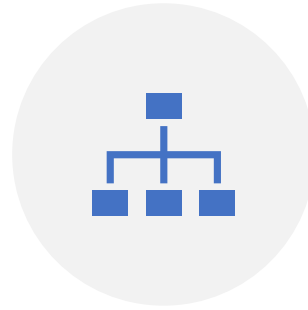
Predicted Labels



Applying the regression model to new data

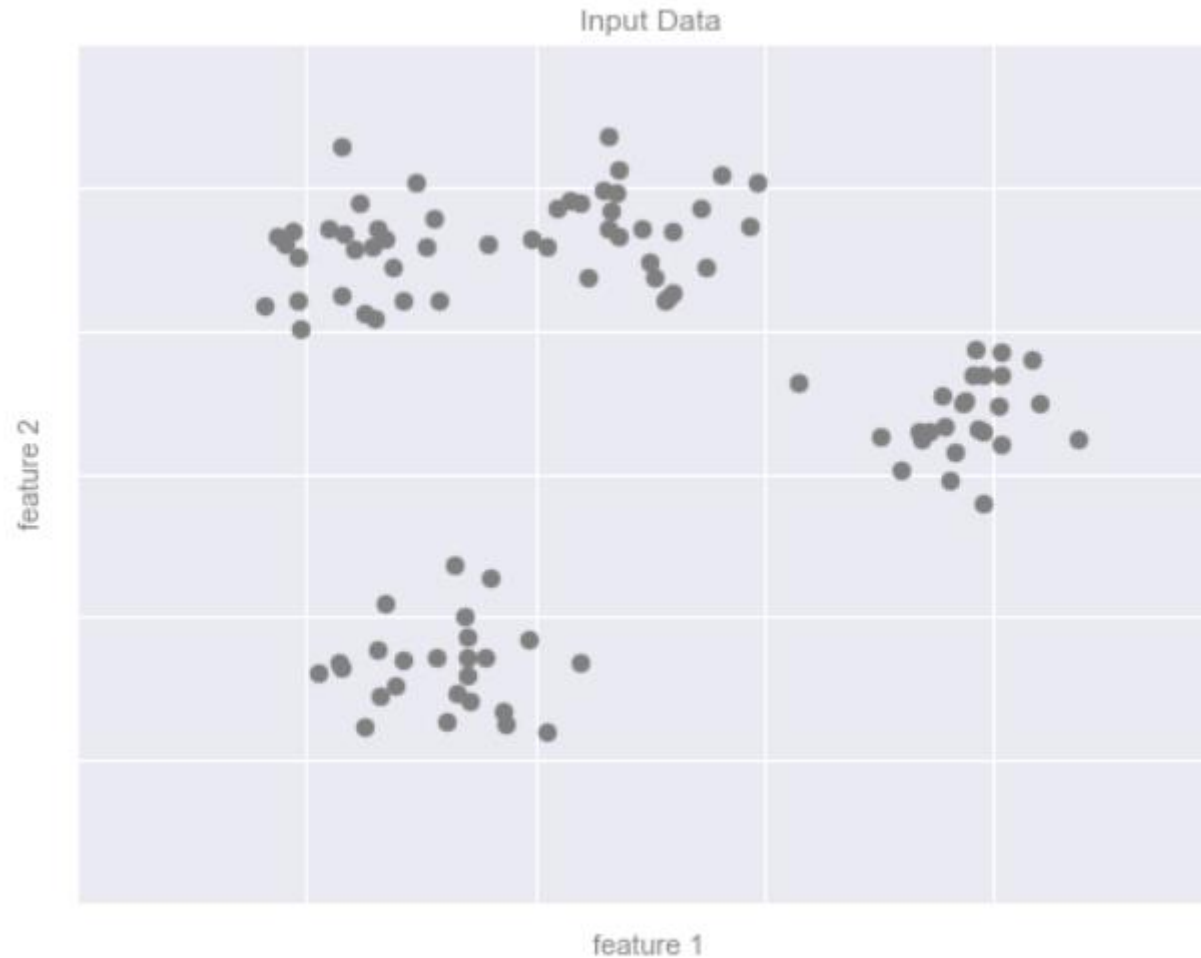
Regression: Predicting continuous labels

- Example of Regression Algorithms
 - Linear Regression
 - Support Vector Machines
 - Decision Trees and Random Forests
- Example of Regression Applications
 - Computing the distance to galaxies observed through a telescope
 - Population Growth Prediction
 - Advertising Popularity prediction
 - Weather Forecasting
 - Market Forecasting



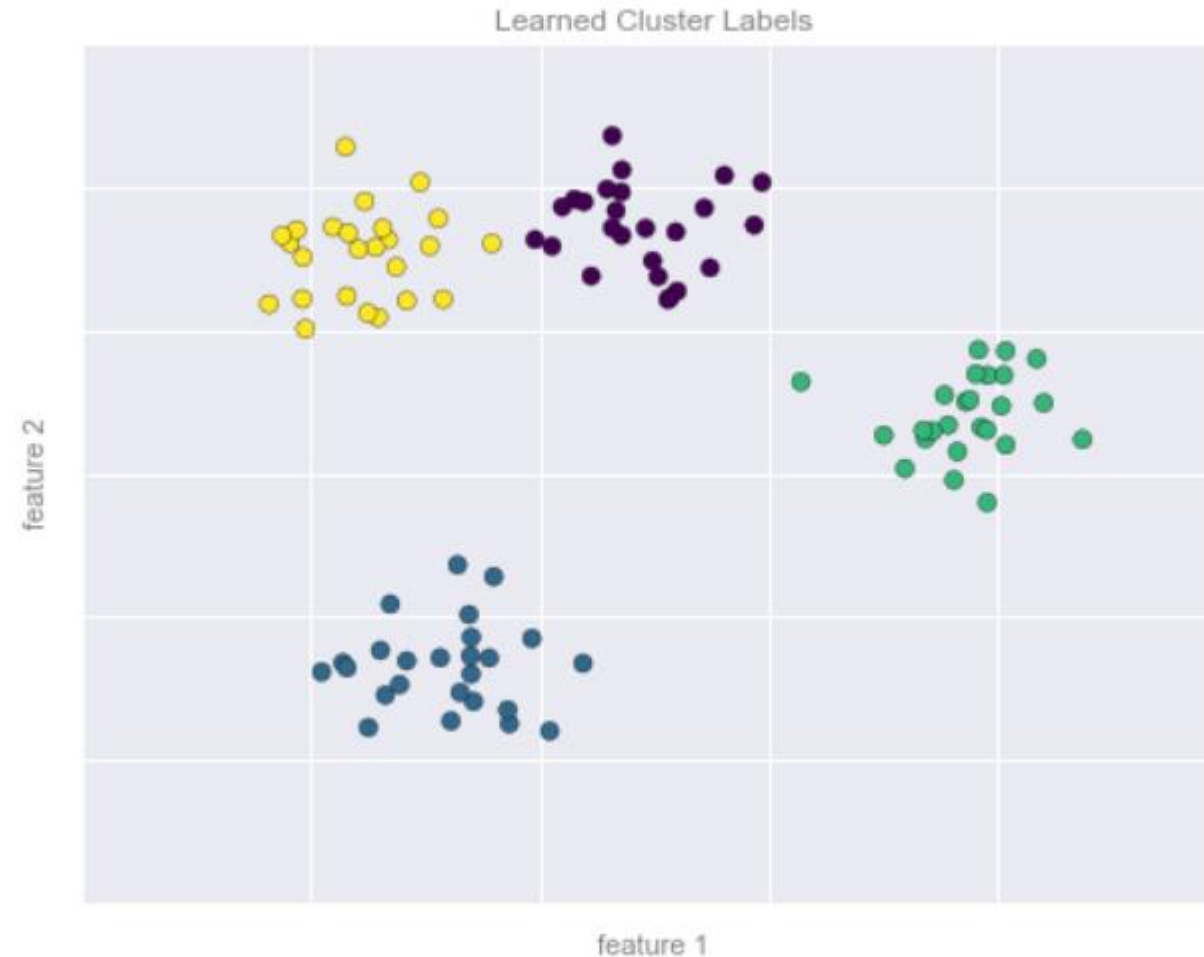
CLUSTERING

Clustering: Inferring labels on unlabelled data



Example data for clustering

Clustering: Inferring labels on unlabelled data



Data labelled with a k-means clustering model

Clustering: Inferring labels on unlabelled data

- Example of Clustering Algorithms
 - k-Means Clustering
 - Gaussian Mixture Models
 - Spectral clustering
- Example of Clustering Applications
 - Recommender Systems
 - Customer Segmentation
 - Targeted Marketing



DIMENSIONALITY REDUCTION

Dimensionality reduction: Inferring structure of unlabelled data

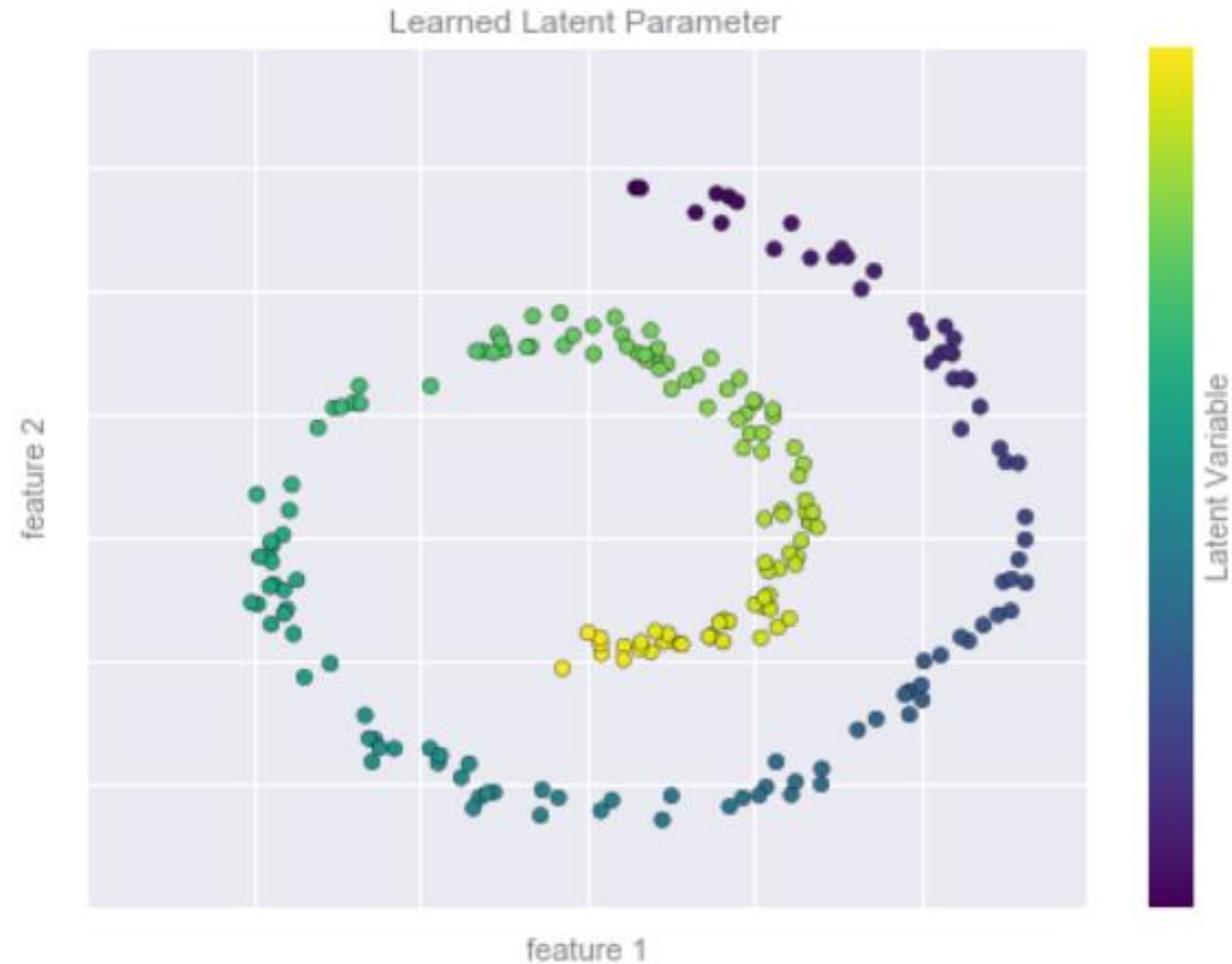
- it seeks to pull out some low-dimensional representation of data that in some way preserves relevant qualities of the full dataset.
- One dimensional line that is arranged in a spiral within this two-dimensional space.
- A suitable dimensionality reduction model in this case would be sensitive to this nonlinear embedded structure, and be able to pull out this lower-dimensionality representation.

Dimensionality reduction: Inferring structure of unlabelled data



Example data for dimensionality reduction

Dimensionality reduction: Inferring structure of unlabelled data



Data with a label learned via dimensionality reduction

Summary: What Is Machine Learning?

- Supervised learning
 - Models that can predict labels based on labelled training data
 - Classification
 - Models that predict labels as two or more discrete categories
 - Regression
 - Models that predict continuous labels
- Unsupervised learning
 - Models that identify structure in unlabelled data
 - Clustering
 - Models that detect and identify distinct groups in the data
 - Dimensionality reduction
 - Models that detect and identify lower-dimensional structure in higher dimensional data



Introducing Scikit-Learn

<https://scikit-learn.org/>

Introducing Scikit-Learn

- Scikit-Learn is the best known Python libraries for implementing machine learning algorithms.
- Data Representation in Scikit-Learn
 - Data as table
 - Features matrix
 - Target array

Data as table

- A basic table is a two-dimensional grid of data
 - Rows represent individual elements of the dataset
 - Columns represent quantities related to each of these elements.
- For example, consider the Iris dataset, famously analysed by Ronald Fisher in 1936.
 - We can download this dataset in the form of a Pandas **DataFrame** using the **Seaborn library**

Data as table

```
In[1]: import seaborn as sns
iris = sns.load_dataset('iris')
iris.head()
```

		sepal_length	sepal_width	petal_length	petal_width	species
5 samples	0	5.1	3.5	1.4	0.2	setosa
	1	4.9	3.0	1.4	0.2	setosa
	2	4.7	3.2	1.3	0.2	setosa
	3	4.6	3.1	1.5	0.2	setosa
	4	5.0	3.6	1.4	0.2	setosa

- Each row of the data refers to a single observed flower.
- The number of rows is the total number of flowers in the dataset.
- In general, we will refer to the rows of the matrix as *samples*, and the number of rows as *n_samples*.

Data as table

```
In[1]: import seaborn as sns  
       iris = sns.load_dataset('iris')  
       iris.head()
```

5 features

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

- Each column of the data refers to a particular quantitative piece of information that describes each sample.
- In general, we will refer to the columns of the matrix as *features*, and the number of columns as *n_features*.

Features matrix

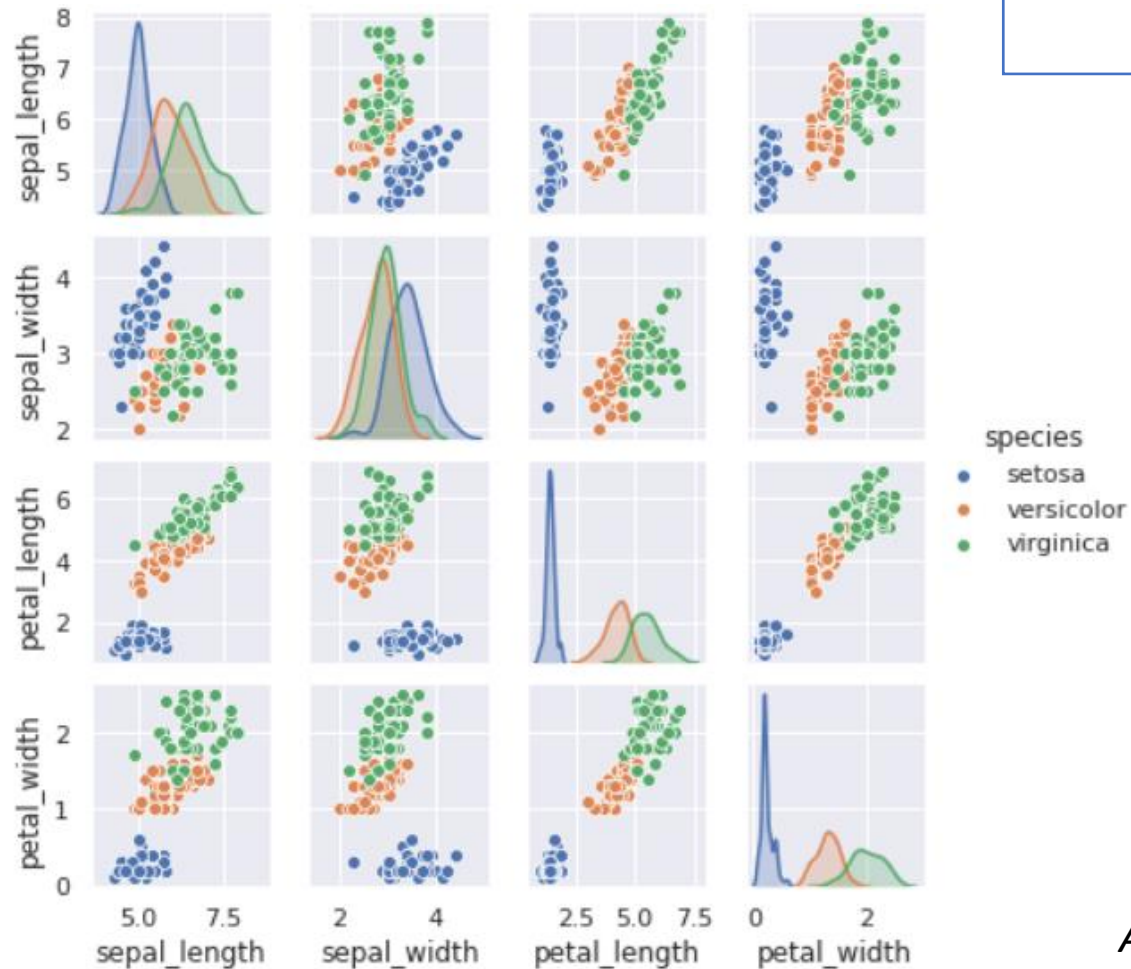
- This table layout makes clear that the information can be thought of as a two dimensional numerical array or matrix, which we will call the *features matrix*.
- This features matrix is often stored in a variable named **X**.
- The features matrix is assumed to be two-dimensional, with shape **[n_samples, n_features]**
 - The samples (i.e., rows) always refer to the individual objects described by the dataset.
 - The features (i.e., columns) always refer to the distinct observations that describe each sample in a quantitative manner.

Target array

- In addition to the feature matrix X , we also generally work with a label or target array, which by convention we will usually call y .
- The target array is usually *one dimensional*, with length $n_samples$.
- For example, in the preceding data we may wish to construct a model that can predict the species of flower based on the other measurements.
- In this case, the species column would be considered the target feature.

Target array

```
In[2]: %matplotlib inline
import seaborn as sns; sns.set()
sns.pairplot(iris, hue='species', size=1.5);
```



A visualization of the Iris dataset

Target array

- For use in Scikit-Learn, we will extract the features matrix and target array from the **DataFrame**.

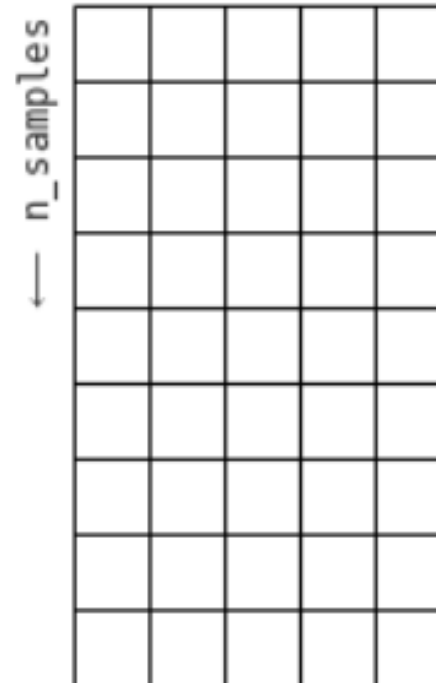
```
In[3]: X_iris = iris.drop('species', axis=1)
      X_iris.shape
```

```
Out[3]: (150, 4)
```

```
In[4]: y_iris = iris['species']
      y_iris.shape
```

```
Out[4]: (150,)
```

Feature Matrix (X)
n_features →



Target Vector (y)



Scikit-Learn's Estimator API

- Most commonly, the steps in using the Scikit-Learn estimator API are as follows
 1. Choose a class of model by importing the appropriate estimator class from ScikitLearn.
 2. Choose model hyperparameters by instantiating this class with desired values.
 3. Arrange data into a features matrix and target vector.
 4. Fit the model to your data by calling the `fit()` method of the model instance.
 5. Apply the model to new data:
 - For supervised learning, often we predict labels for unknown data using the `predict()` method.
 - For unsupervised learning, we often transform or infer properties of the data using the `transform()` or `predict()` method.

HandsOn-01

Simple Linear Regression



HandsOn-02

Iris Classification and Dimensionality



HandsOn-03Rainfall Prediction

Rainfall Prediction using Linear Regression



Summary

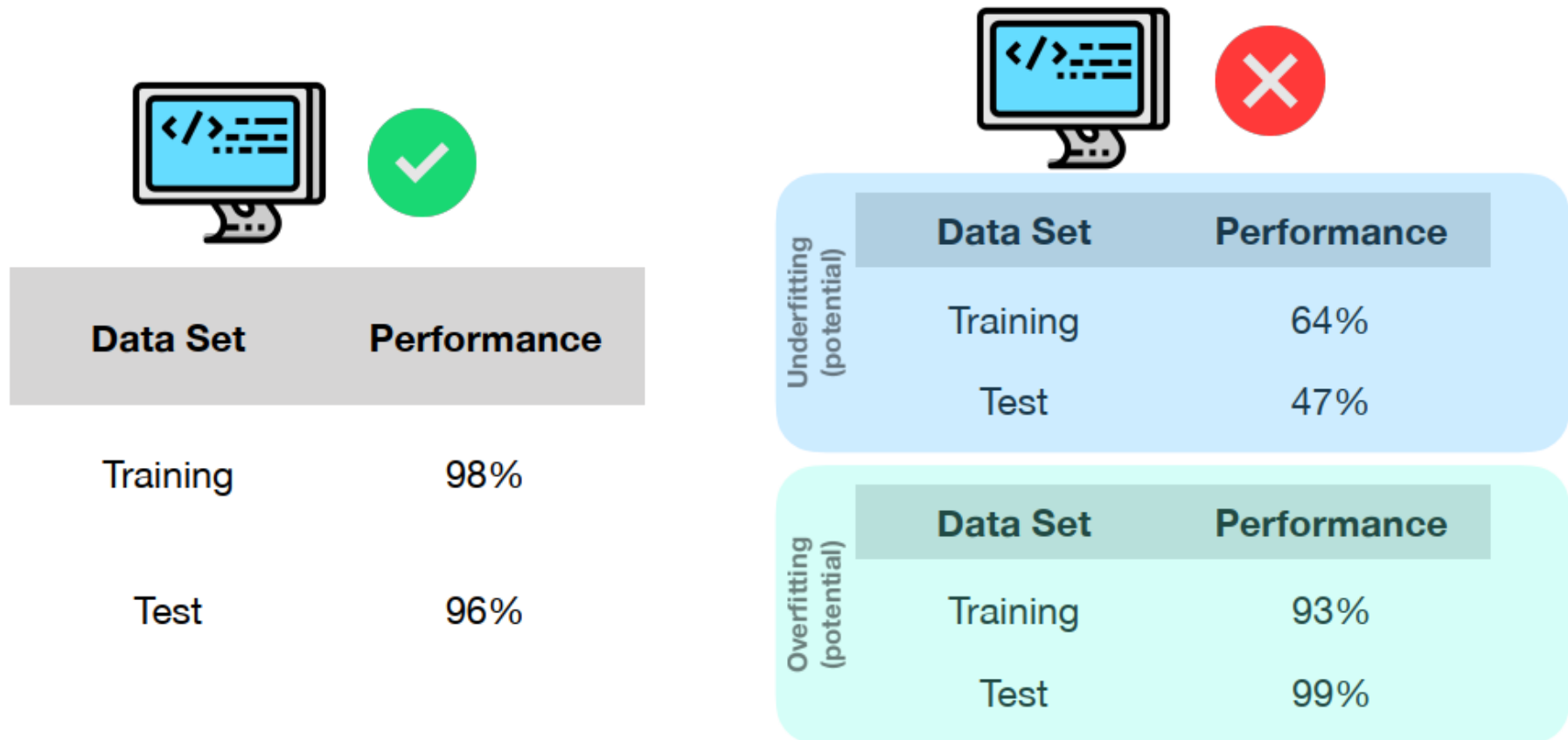
- The steps in using the Scikit-Learn estimator API
 1. Choose a class of model
 2. Choose model hyperparameters
 3. Arrange data into a features matrix and target vector (supervised).
 4. Fit the model to the training data
 5. Use the model to predict labels for new data



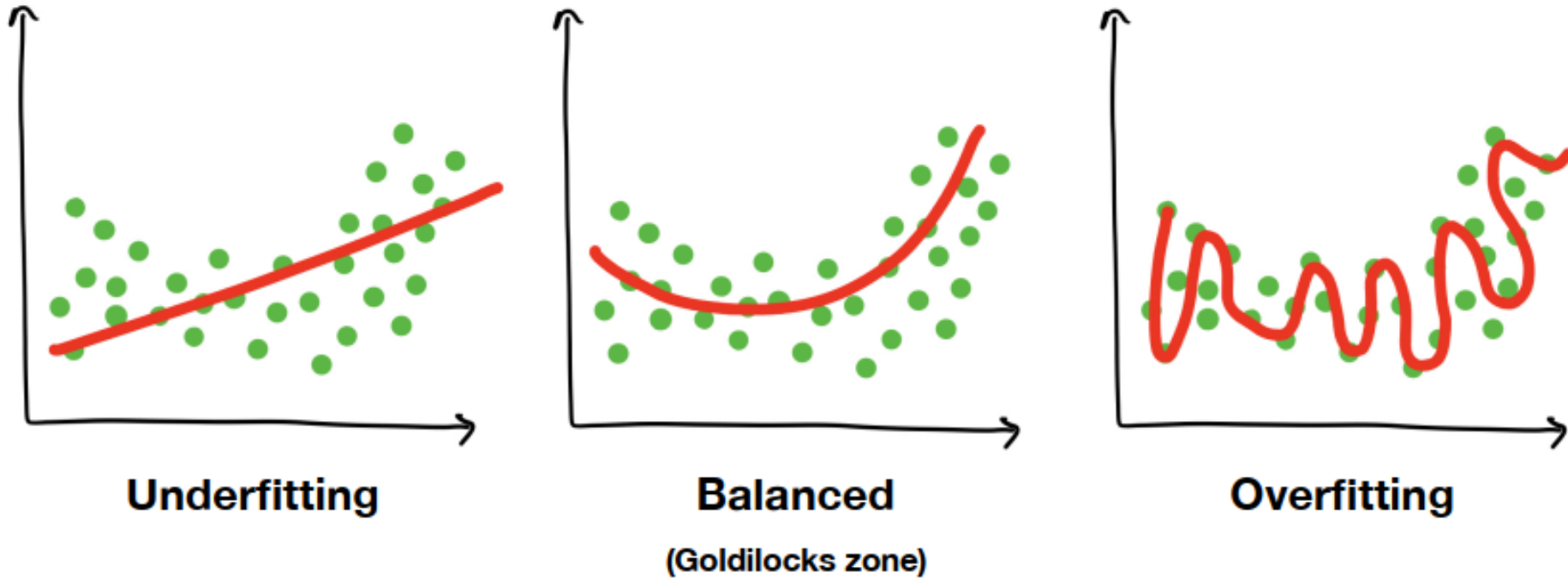
Hyperparameters and Model Validation

Hyperparameters and Model Validation

- Model Validation is a way to *validate* that our model and our hyperparameters are a good fit to the data.



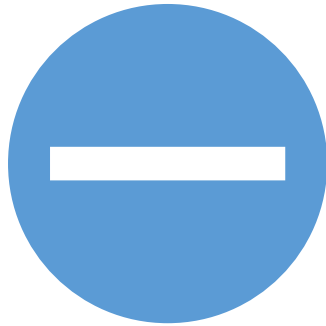
Hyperparameters and Model Validation



HandsOn-04

Model Validation Iris Data (Intro)

Model validation

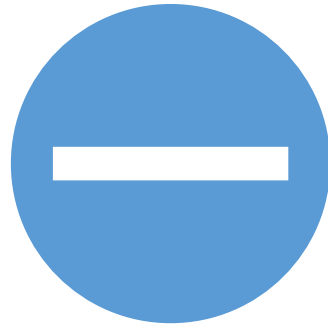


Holdout



Cross-validation

Model validation



Holdout

Model validation the right way: Holdout sets

- We hold back some subset of the data from the training of the model.
- Then use this holdout set to check the model performance.
- We can do this splitting using the `train_test_split` utility in Scikit-Learn

HandsOn-04

Model Validation Iris Data (Holdout)

Model validation



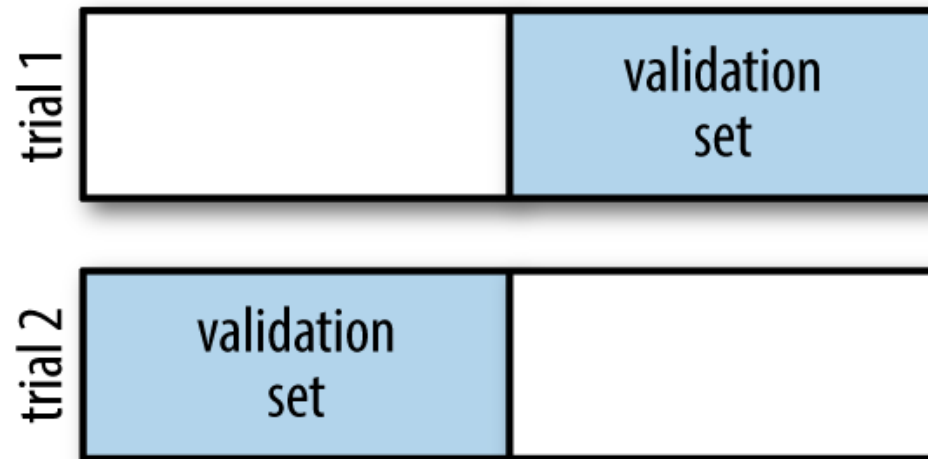
Cross-validation

Model validation via cross-validation

- One disadvantage of using a holdout set for model validation is that we have lost a portion of our data to the model training.
- In the previous case, half the dataset does not contribute to the training of the model!
- This is not optimal, and can cause problem especially if the initial set of training data is small.

Model validation via cross-validation

- One way to address this is to use *cross-validation*—that is, to do a sequence of fits where each subset of the data is used both as a training set and as a validation set.
- Visually, it might look something like



Model validation via cross-validation

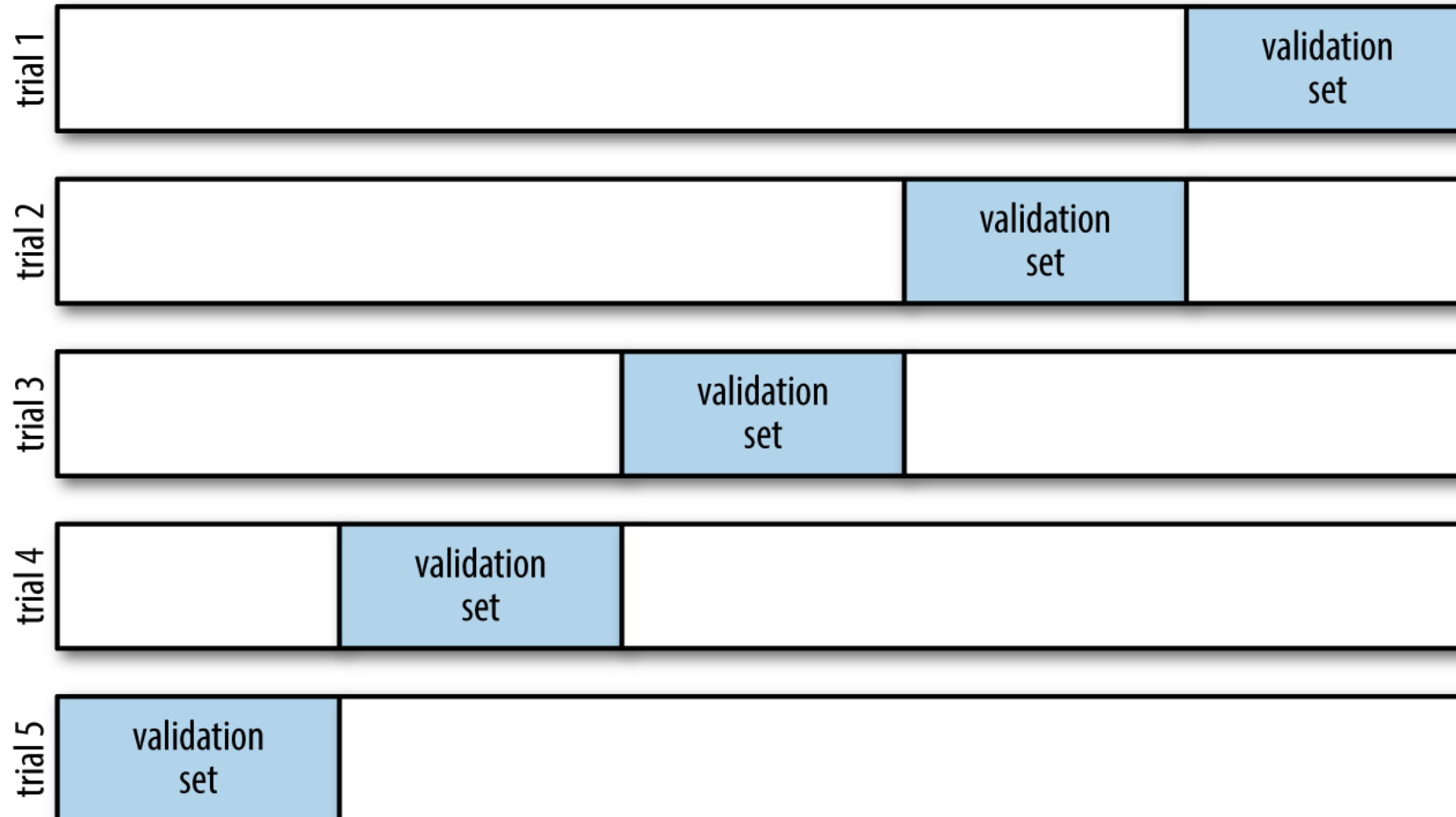
```
In[6]: y2_model = model.fit(X1, y1).predict(X2)
       y1_model = model.fit(X2, y2).predict(X1)
       accuracy_score(y1, y1_model), accuracy_score(y2, y2_model)
```

```
Out[6]: (0.95999999999999999996, 0.90666666666666666662)
```

- This particular form of cross-validation is a two-fold cross-validation—one in which we have split the data into two sets and used each in turn as a validation set.

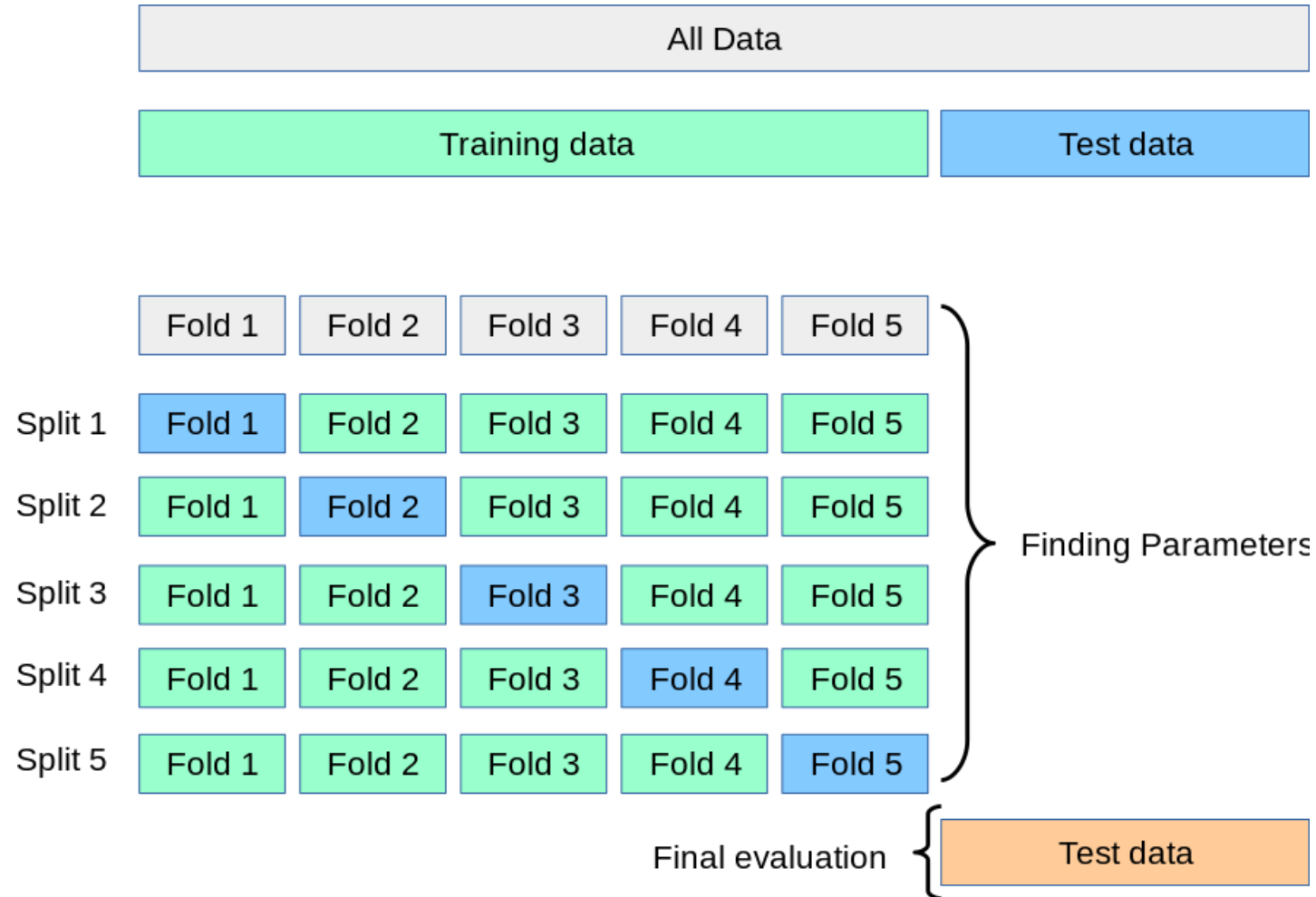
Model validation via cross-validation

- five-fold cross-validation



Model validation via cross-validation

- five-fold cross-validation



HandsOn-04

Model Validation Iris Data (Cross-validation)



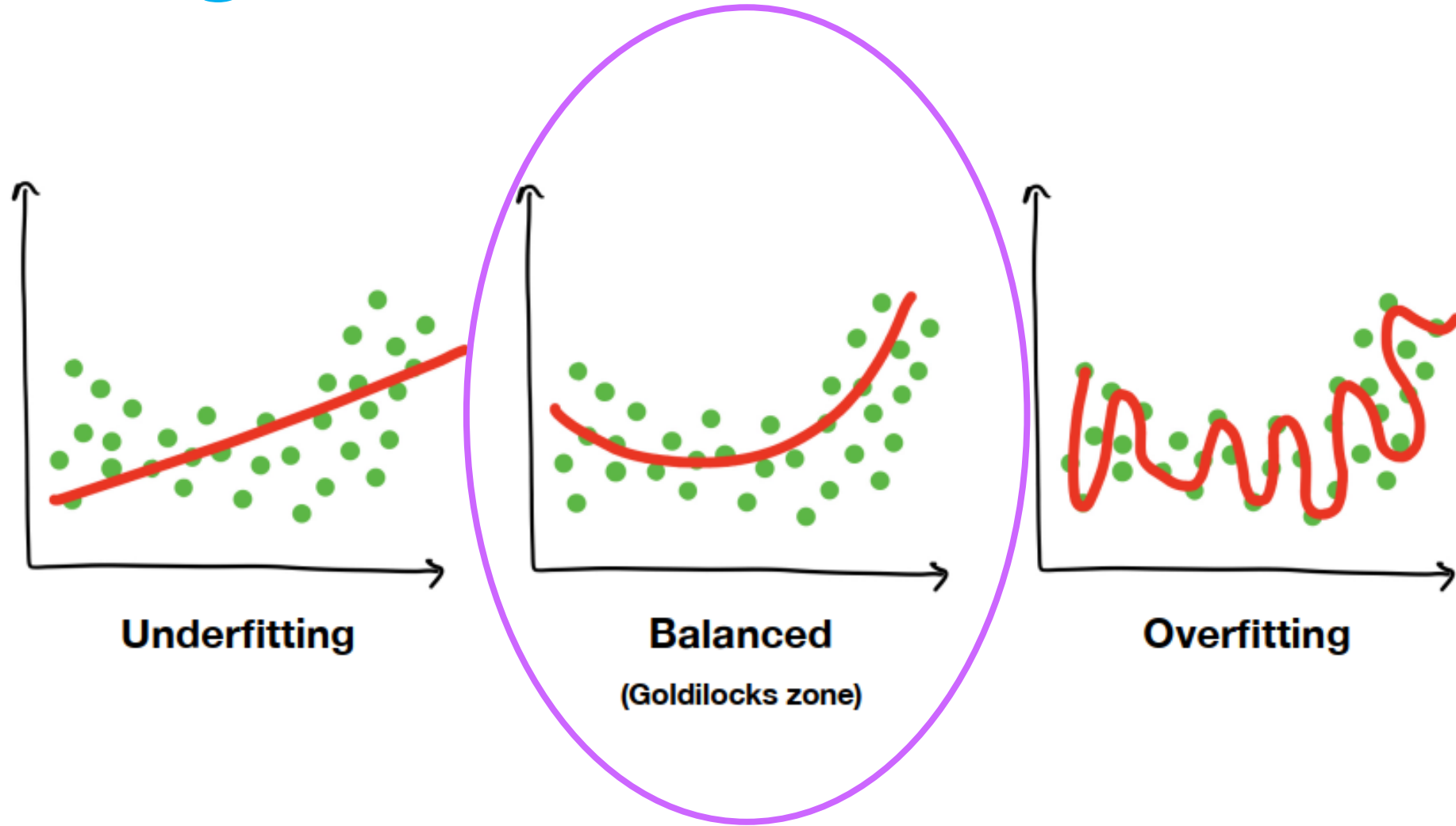


Selecting the Best Model

Selecting the Best Model

- Of core importance is the following question: *if our estimator is underperforming, how should we move forward?*
- There are several possible answers.
 - Use a more complicated/more flexible model
 - Use a less complicated/less flexible model
 - Gather more training samples
 - Gather more data to add features to each sample

Selecting the Best Model

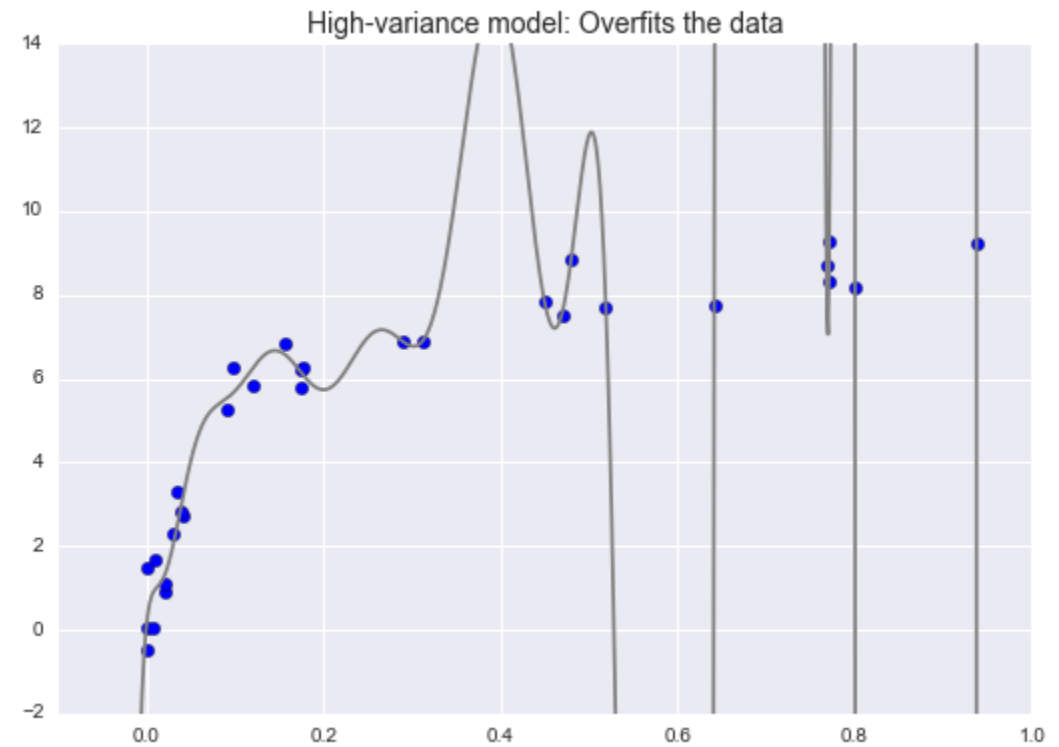
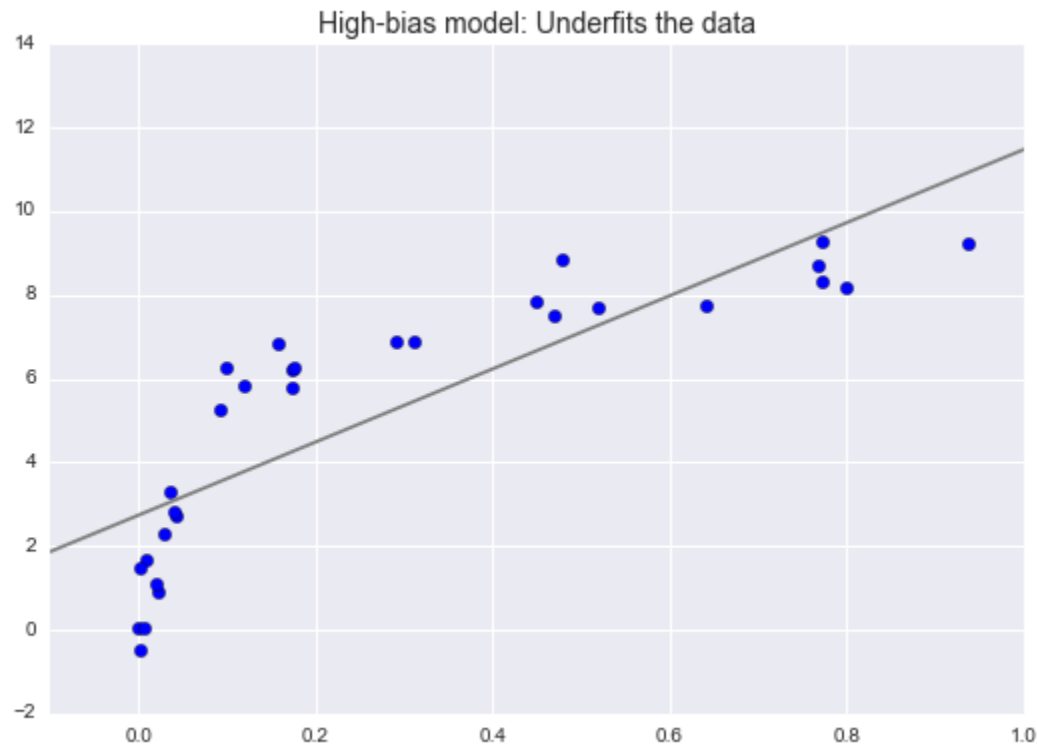


Selecting the Best Model

- Sometimes
 - Using a more complicated model will give worse results.
 - Adding more training samples may not improve your results.
- The ability to determine what steps will improve your model is what separates the successful machine learning practitioners from the unsuccessful.

The bias–variance trade-off

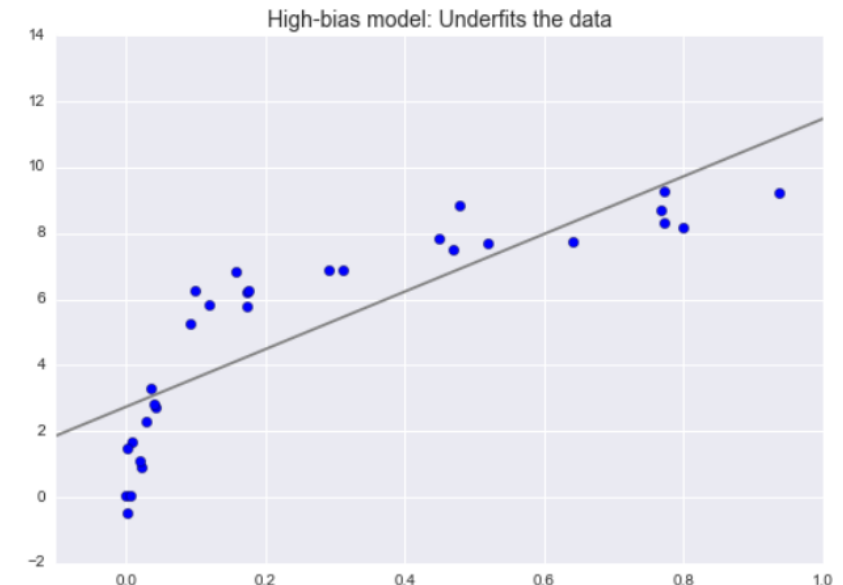
- Fundamentally, the question of “the best model” is about finding a sweet spot in the trade-off between *bias* and *variance*.



A high-bias and high-variance regression model

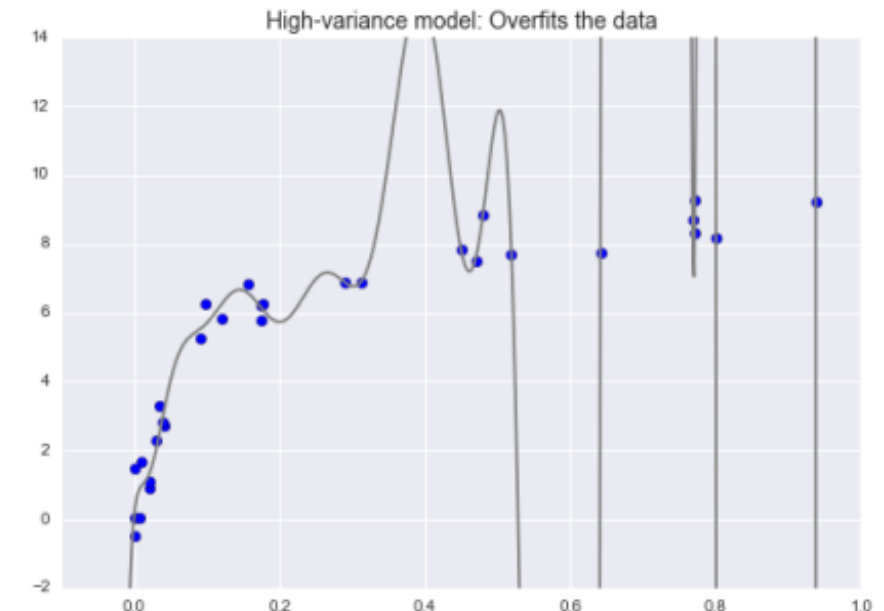
The bias–variance trade-off

- The model attempts to find a straight-line fit through the data.
- The data are intrinsically more complicated than a straight line, the straight-line model will never be able to describe this dataset well.
- Such a model is said to *underfit* the data; that is, it does not have enough model flexibility to suitably account for all the features in the data.
- Another way of saying this is that the model has *high bias*.



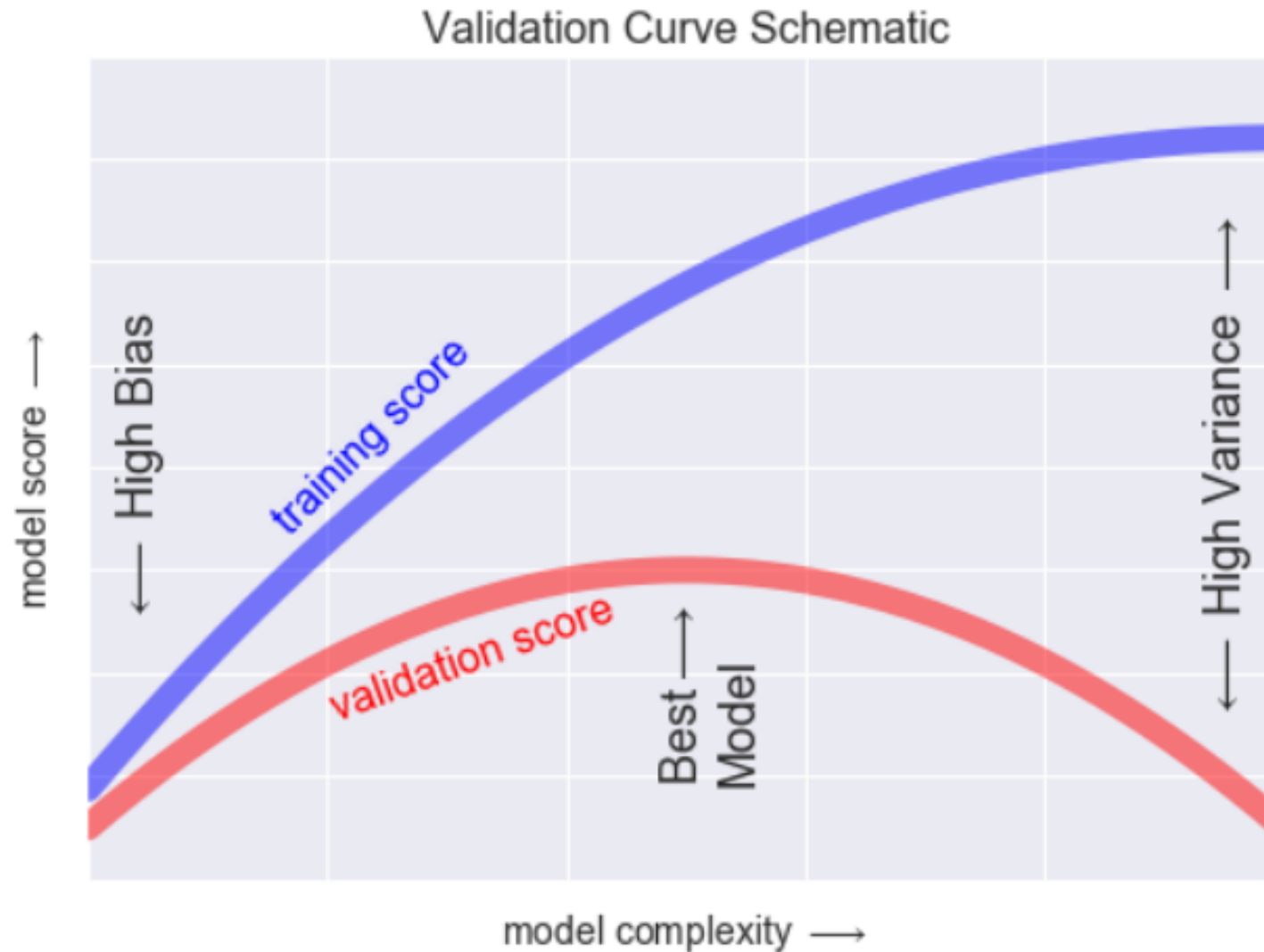
The bias–variance trade-off

- The model attempts to fit a high-order polynomial through the data.
- Even though it very accurately describes the training data, its precise form seems to be more reflective of the particular noise properties of the data rather than the intrinsic properties of whatever process generated that data.
- Such a model is said to *overfit* the data; that is, it has so much model flexibility that the model ends up accounting for random errors as well as the underlying data distribution.
- Another way of saying this is that the model has *high variance*.



Validation Curves

Validation Curves



Validation Curves

- The training score is everywhere higher than the validation score. This is generally the case: the model will be a better fit to data it has seen than to data it has not seen.
- For very low model complexity (a high-bias model), the training data is underfit, which means that the model is a poor predictor both for the training data and for any previously unseen data.
- For very high model complexity (a high-variance model), the training data is overfit, which means that the model predicts the training data very well, but fails for any previously unseen data.
- For some intermediate value, the validation curve has a maximum. This level of complexity indicates a suitable trade-off between bias and variance.

Validation curves in Scikit-Learn

- We will use a *polynomial regression* model: this is a generalized linear model in which the degree of the polynomial is a tuneable parameter.
- For example,
 - A degree-1 polynomial fits a straight line to the data; for model parameters a and b :
 - $y = ax + b$
 - A degree-3 polynomial fits a cubic curve to the data; for model parameters a, b, c, d :
 - $y = ax^3 + bx^2 + cx + d$

HandsOn-05

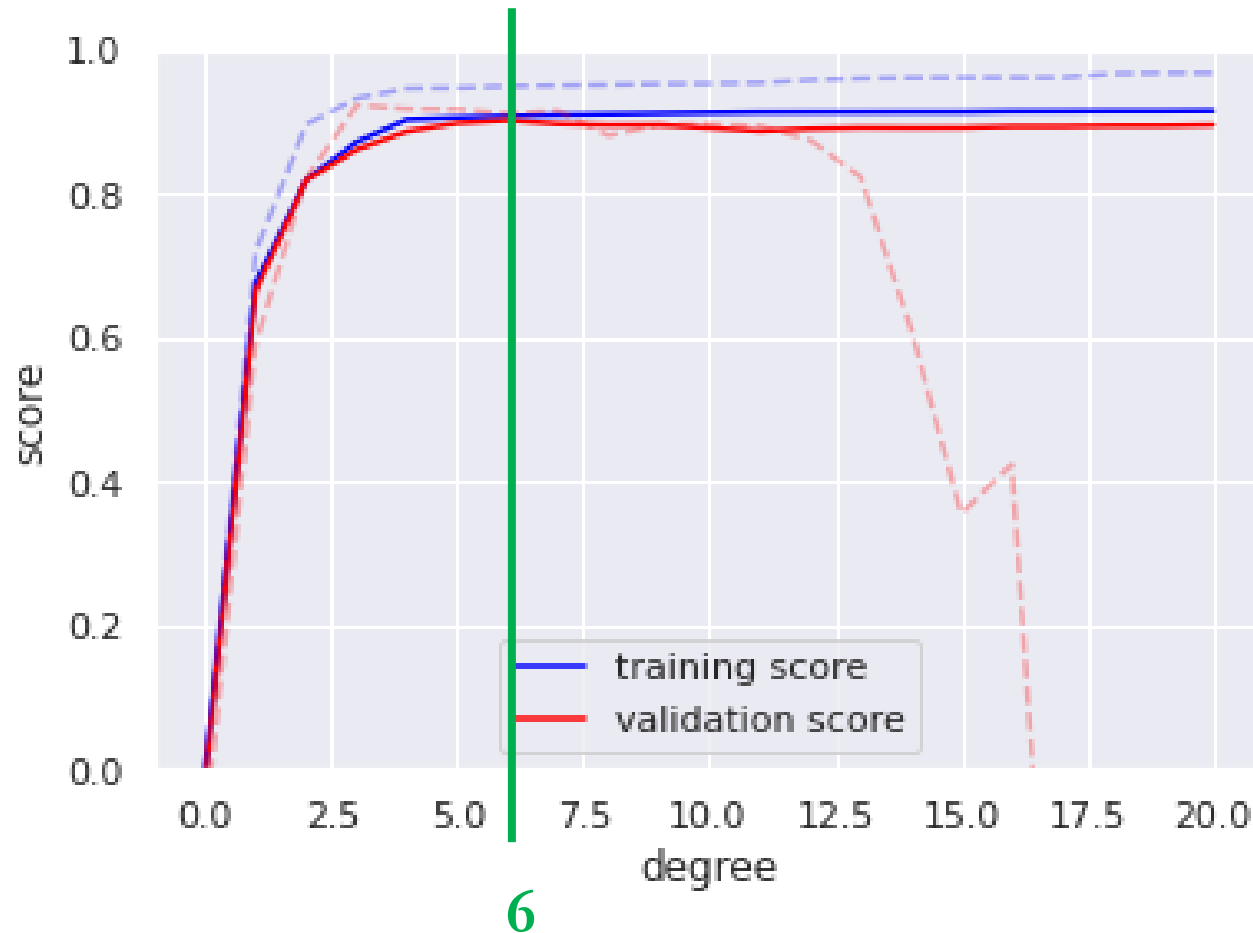
Validation Curves using Polynomial Regression (Validation Curves)



Learning Curves

Learning Curves

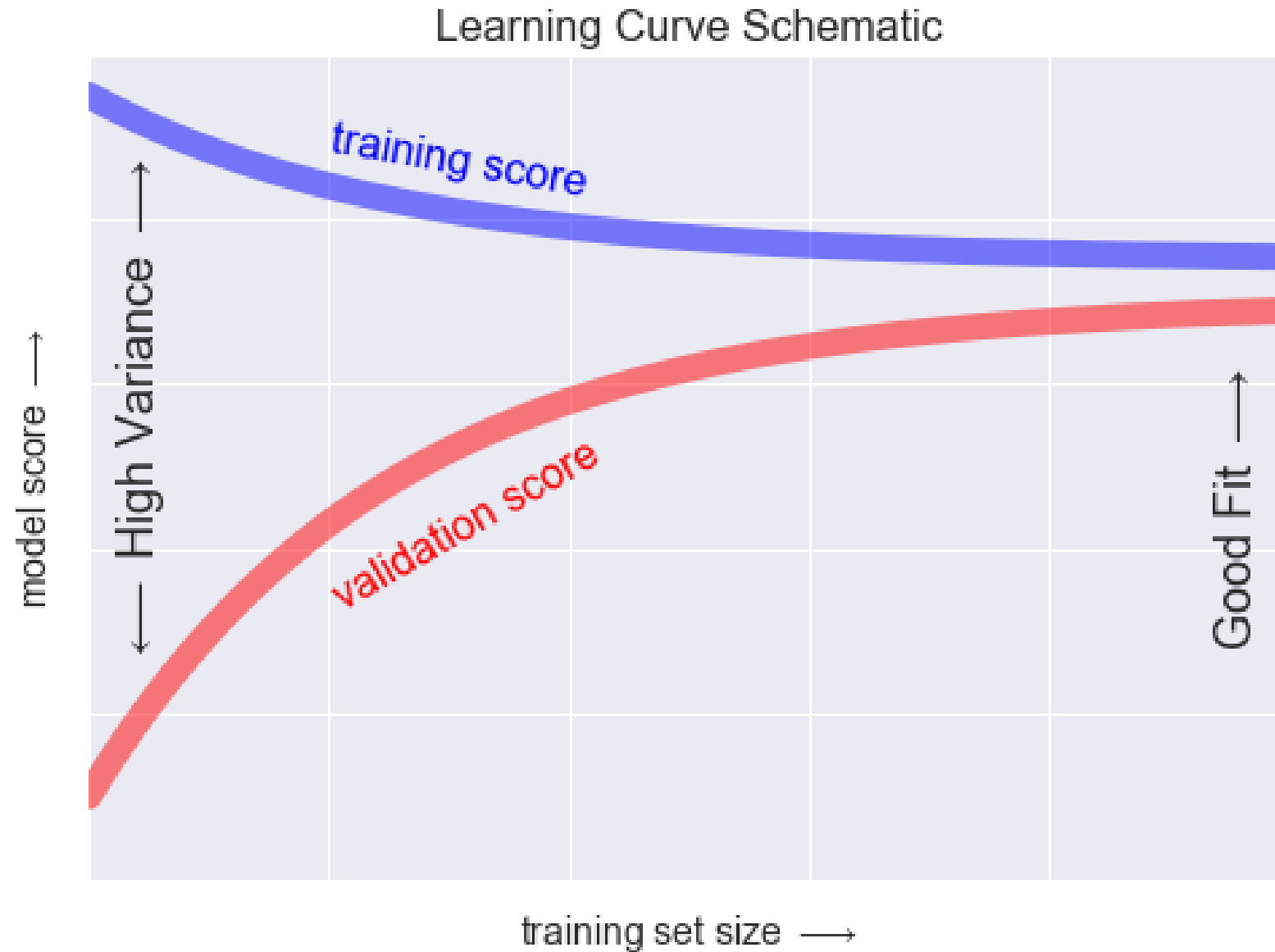
- One important aspect of model complexity is that the optimal model will generally depend on the size of your training data.



Learning Curves

- The behaviour of the validation curve has two important inputs:
 - the model complexity
 - the number of training points
- A plot of the training/validation score with respect to the size of the training set is known as a *learning curve*.

Learning Curves



Learning Curves

- The general behavior we would expect from a learning curve is this:
 - A model of a given complexity will *overfit* a small dataset: this means the **training score** will be relatively **high**, while the **validation score** will be relatively **low**.
 - A model of a given complexity will *underfit* a large dataset: this means that the **training score** will **decrease**, but the **validation score** will **increase**.
 - A model will never, except by chance, give a better score to the validation set than the training set: this means the **curves** should keep getting closer together but **never cross**.

HandsOn-05

Validation and Learning Curves (Learning Curves)



Validation in Practice: Grid Search

- Grid search is the process of performing hyper parameter tuning in order to determine the optimal values for a given model.
- Here is an example of using grid search to find the optimal polynomial model.

```
In[18]: from sklearn.model_selection import GridSearchCV
        param_grid = {'polynomialfeatures__degree': np.arange(21),
                       'linearregression__fit_intercept': [True, False],
                       'linearregression__normalize': [True, False]}

        grid = GridSearchCV(PolynomialRegression(), param_grid, cv=7)
```

Validation in Practice: Grid Search

- Calling the `fit()` method will fit the model at each grid point

```
In[19]: grid.fit(X, y)
```

- Display the best parameters

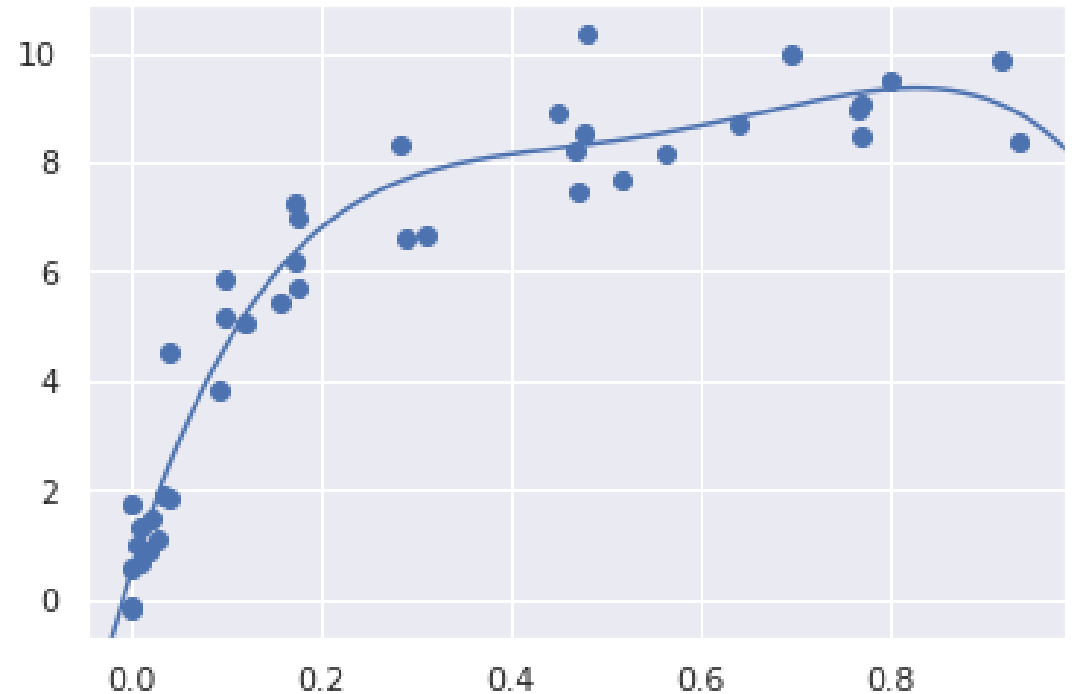
```
In[20]: grid.best_params_
```

```
Out[20]: {'linearregression__fit_intercept': False,  
          'linearregression__normalize': True,  
          'polynomialfeatures__degree': 4}
```

Validation in Practice: Grid Search

- Use the best model and show the fit to our data

```
In[21]: model = grid.best_estimator_  
        plt.scatter(X.ravel(), y)  
        lim = plt.axis()  
        y_test = model.fit(X, y).predict(X_test)  
        plt.plot(X_test.ravel(), y_test);  
        plt.axis(lim);
```



HandsOn-05

Validation and Learning Curves (GridSearchCV)



HandsOn-06

Social Adv with Decision Tree (Assignment)

