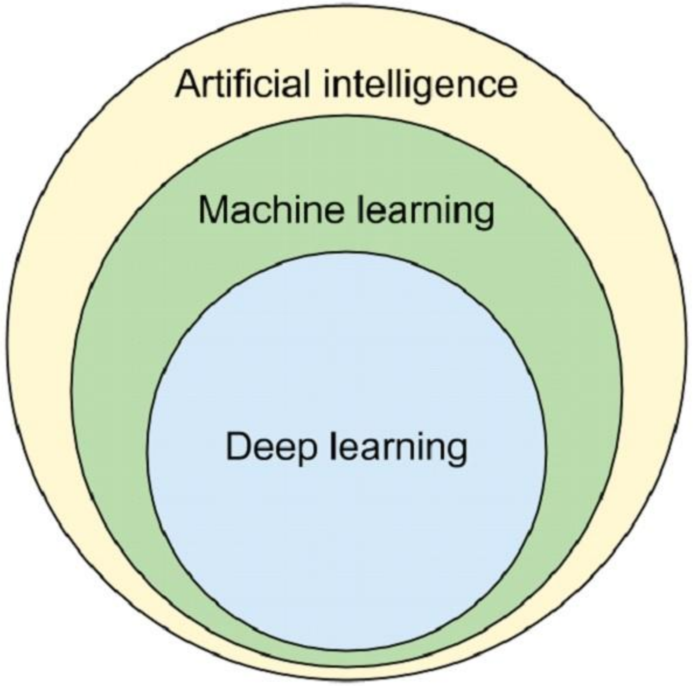
Note) Slides and example codes are available:
https://github.com/mint-lab/dl_tutorial

Machine Learning Tutorial with scikit-learn

Sunglok Choi, Assistant Professor, Ph.D.
Dept. of Computer Science and Engineering, SEOULTECH
sunglok@seoultech.ac.kr | https://mint-lab.github.io/

Machine Learning

- Machine learning (shortly *ML*)
  - A type of computer algorithms that achieve tasks (or build models) **automatically through the use of data**
  - **Data-driven approaches** vs. rule-based approaches
  - **Induction** (귀납법 in Korean) vs. deduction (연역법 in Korean)
  - Artificial intelligence vs. **machine learning** vs. deep (neural network) learning



vs.

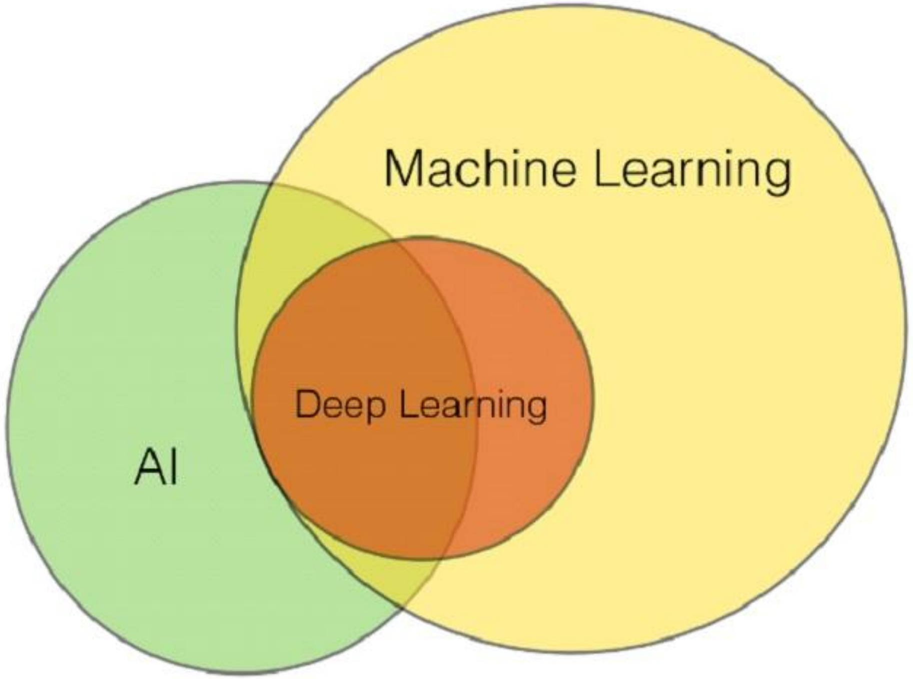
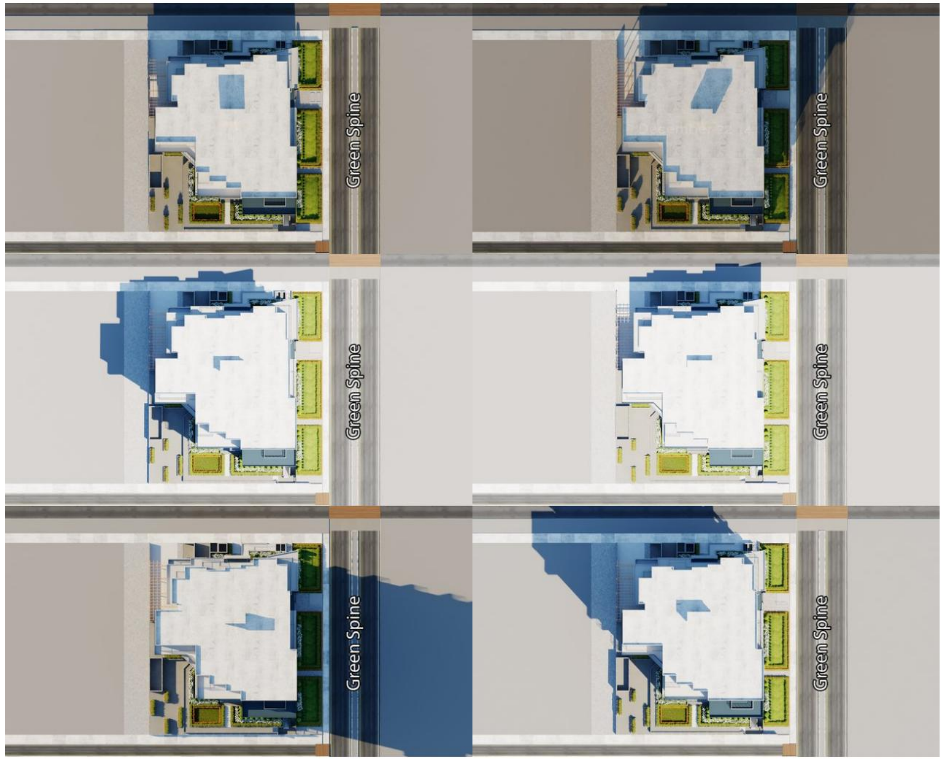


Image: Wikipedia

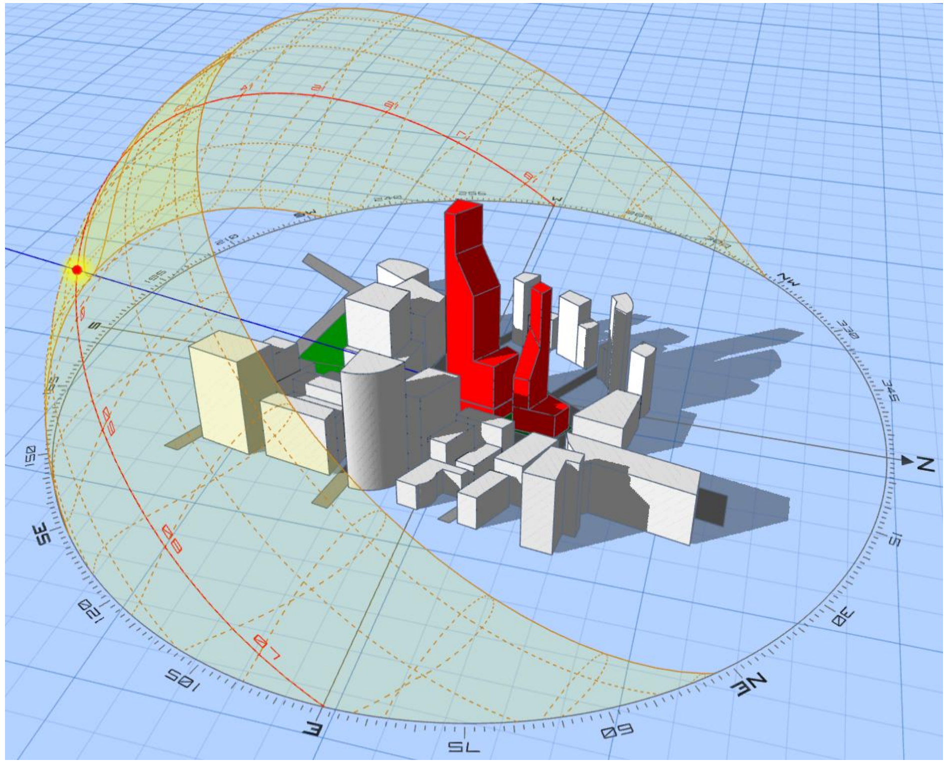
Machine Learning

- Machine learning (shortly *ML*)
  - **Data-driven approaches** vs. rule-based approaches
    - Example) Selecting a parking spot while avoiding strong sunshine



Data-driven approaches (shadow record)

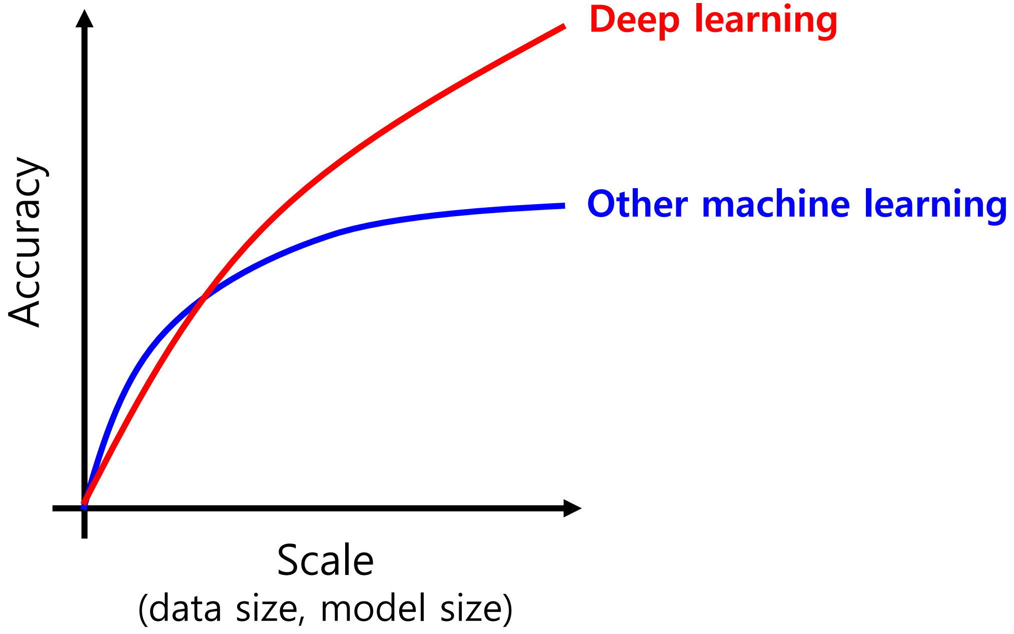
vs.



Rule-based approaches (sun path, ray casting, …)

Image: RealSpace, 3D Sun-Path

Why Machine Learning? (vs. Deep Learning)



Machine Learning

- **ML procedure** (~ trial and error; 시행착오 in Korean)
  1. **Data** acquisition
  2. Data preprocessing (e.g. labeling)
  3. **Feature** selection and extraction
  4. **Model** and **cost function** selection (or design)
  5. **Hyperparameter** selection (e.g. optimizer, learning rate)
  6. Model **training**
  7. Model **testing**
  8. Model **deployment**

80% of time spent for Machine Learning Projects is allocated to Data related tasks

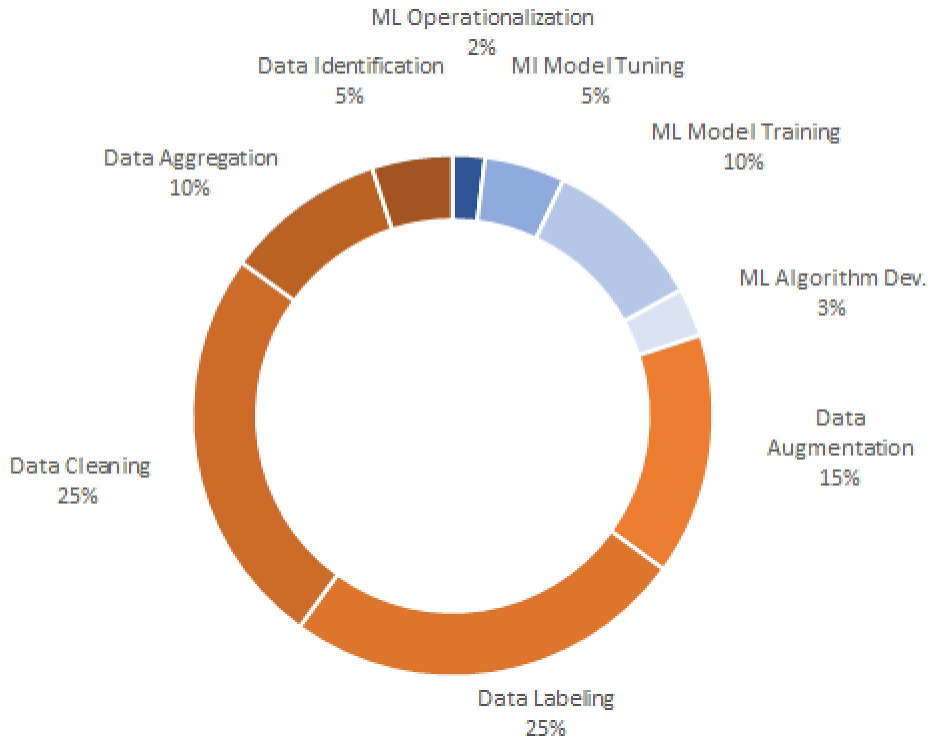


Image: Matthias Heller, [Data Labeling: AI's Human Bottleneck]

Machine Learning

- **ML approaches** (with respect to the given data)
  - Supervised learning (지도학습 in Korean): Given example inputs and their desired targets
    - Note) Self-supervised learning, weakly supervised learning, semi-supervised learning, few-shot learning, …
  - Unsupervised learning (비지도학습 in Korean): Given only example inputs (without their targets)
    - Note) It is sometimes used for feature learning (a.k.a. representation learning).
  - Reinforcement learning (강화학습 in Korean): Given current input and intermediate feedback (reward/penalty) after executing the current action

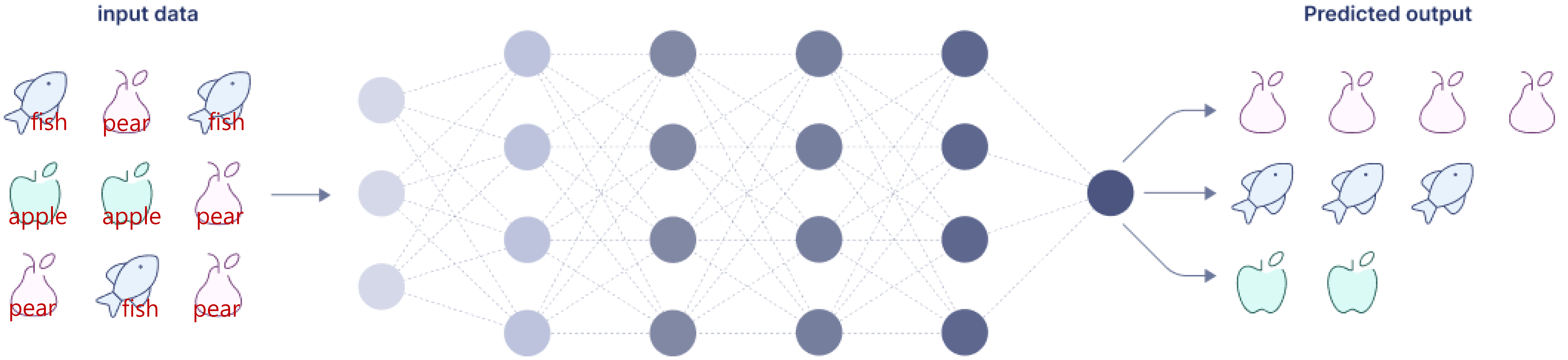
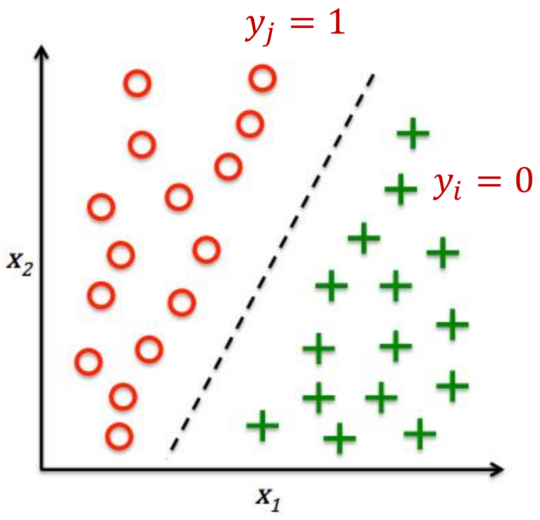


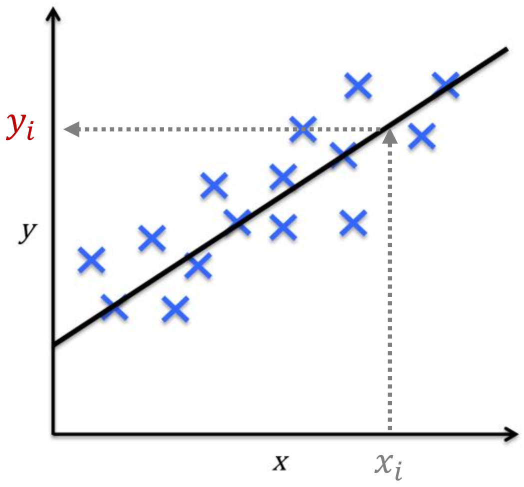
Image: Scribbr

Machine Learning

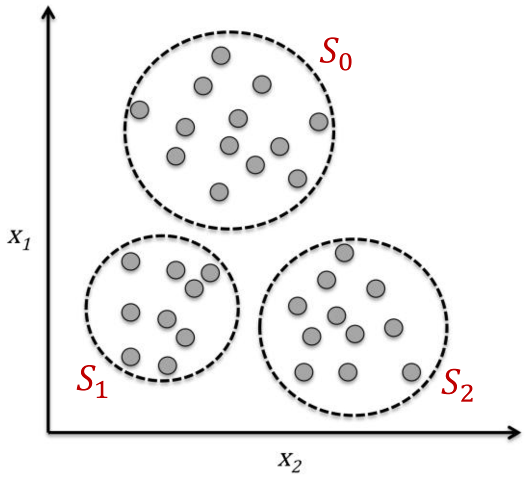
- **ML problem formulations** (with respect to the targets)
  - **Classification**: Identifying which category (**discrete**) an observation belongs to
  - **Regression**: Predicting dependent variables (**continuous**) corresponding to independent variables
  - **Clustering**: Grouping similar data into sets (**discrete**)
  - Dimensionality reduction: Transforming the given high-dimensional data to their low-dimensional representation
  - Model selection: Choosing a model type and its parameters well-associated with the given data
  - Reinforcement learning: Finding an optimal policy for achieving a goal (more cumulative reward)



Classification



Regression

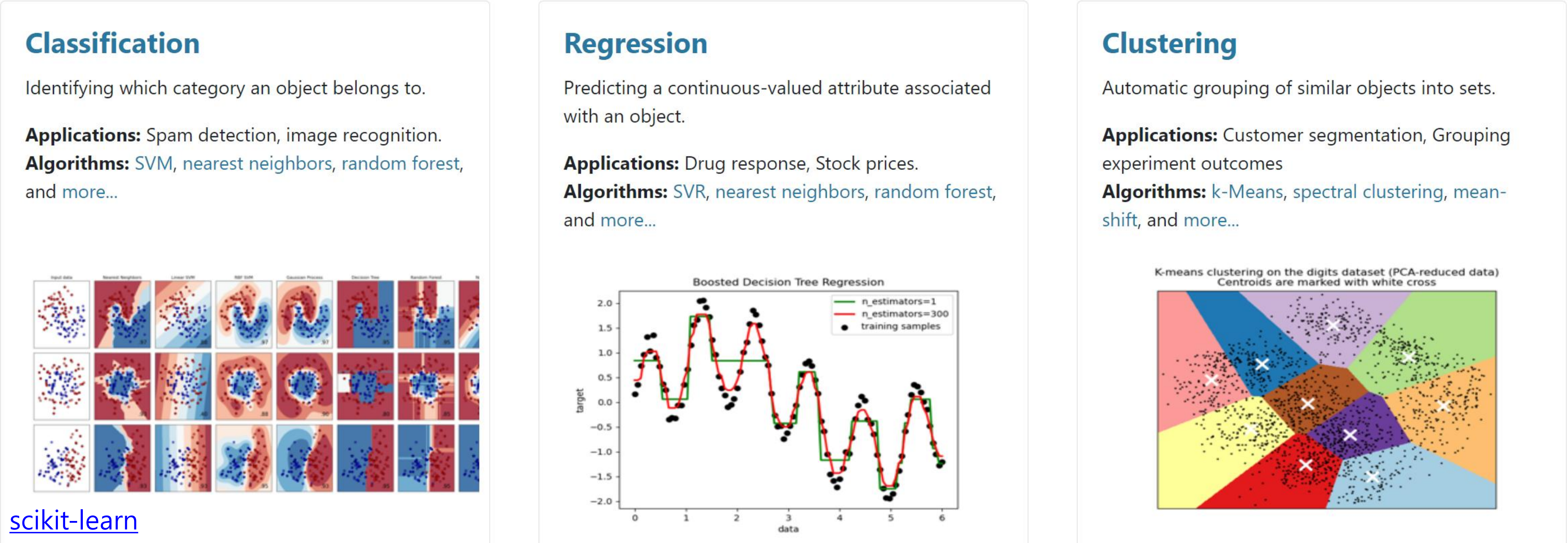


Clustering

Image: Raschka's book

scikit-learn: A Machine Learning Library in Python

- scikit-learn (a.k.a. *sklearn*) is a Python-based open-source machine learning library.
  - It is built on NumPy, SciPy, and matplotlib (and included in Anaconda by default).
  - Its name *sci* stands for *science*.
- References (Documentation): User guide (~ ML book), API reference, tutorials, and examples
  - Example datasets for education and algorithm evaluation.
    - Note) Machine Learning Repository, UCI



Classification

Identifying which category an object belongs to.

Applications: Spam detection, image recognition.
Algorithms: SVM, nearest neighbors, random forest, and more...

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.
Algorithms: SVR, nearest neighbors, random forest, and more...

Clustering

Automatic grouping of similar objects into sets.

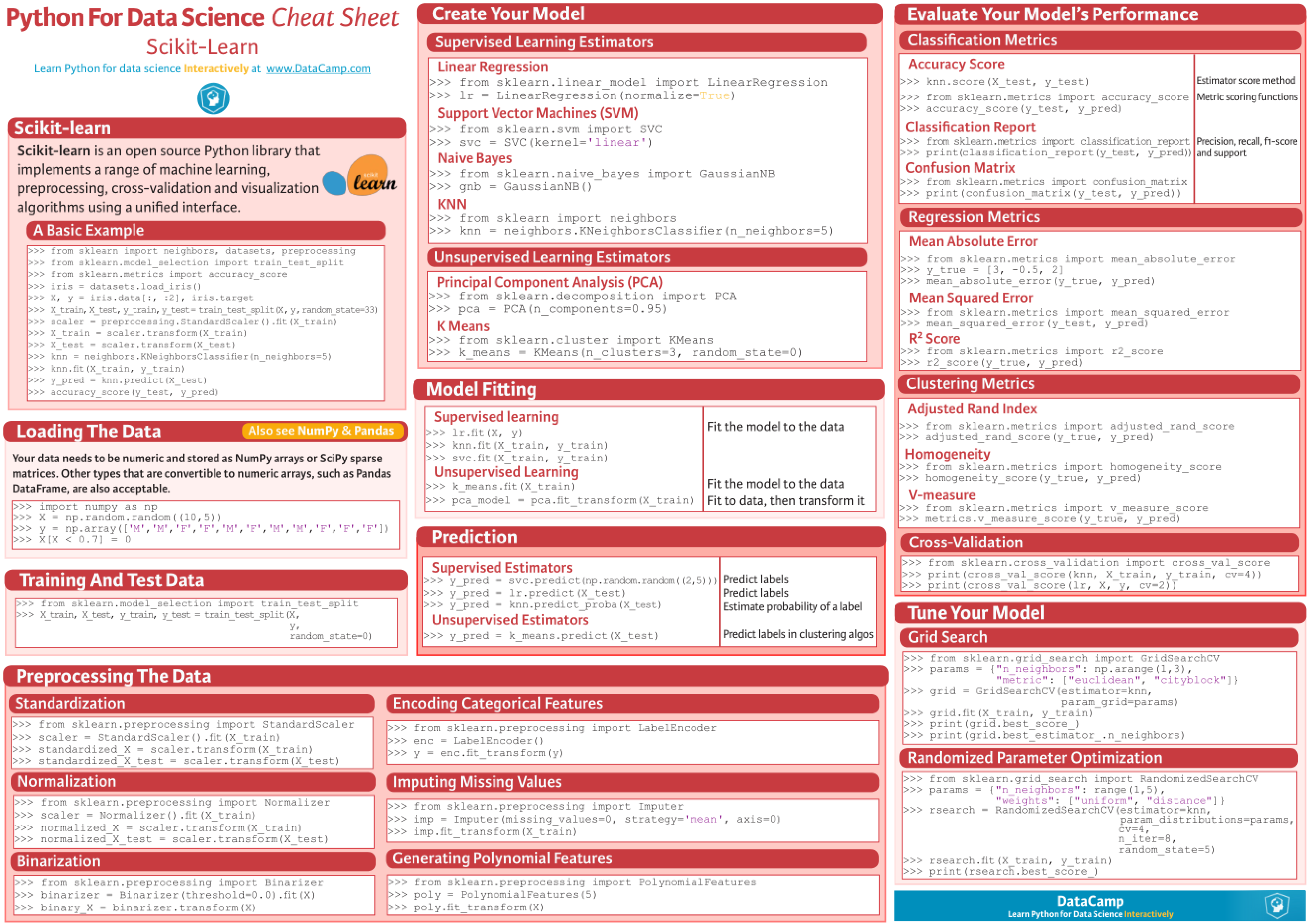
Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: k-Means, spectral clustering, mean-shift, and more...

K-means clustering on the digits dataset (PCA-reduced data)
Centroids are marked with white cross

Image: scikit-learn

scikit-learn:

- scikit-learn
  - It is buil
  - Its name
- References
  - Example
    - Not



Python For Data Science *Cheat Sheet*

Scikit-Learn

Learn Python for data science Interactively at www.DataCamp.com

Scikit-learn

Scikit-learn is an open source Python library that implements a range of machine learning, preprocessing, cross-validation and visualization algorithms using a unified interface.

A Basic Example

```
>>> from sklearn import neighbors, datasets, preprocessing
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.metrics import accuracy_score
>>> iris = datasets.load_iris()
>>> X, y = iris.data[:, :2], iris.target
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=33)
>>> scaler = preprocessing.StandardScaler().fit(X_train)
>>> X_train = scaler.transform(X_train)
>>> X_test = scaler.transform(X_test)
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
>>> knn.fit(X_train, y_train)
>>> y_pred = knn.predict(X_test)
>>> accuracy_score(y_test, y_pred)
```

Loading The Data

Also see NumPy & Pandas

Your data needs to be numeric and stored as NumPy arrays or SciPy sparse matrices. Other types that are convertible to numeric arrays, such as Pandas DataFrame, are also acceptable.

```
>>> import numpy as np
>>> X = np.random.random((10,5))
>>> y = np.array(['M','M','F','F','M','F','M','M','F','F','F'])
>>> X[X < 0.7] = 0
```

Training And Test Data

```
>>> from sklearn.model_selection import train_test_split
>>> X_train, X_test, y_train, y_test = train_test_split(X,
                                                         y,
                                                         random_state=0)
```

Create Your Model

Supervised Learning Estimators

Linear Regression

```
>>> from sklearn.linear_model import LinearRegression
>>> lr = LinearRegression(normalize=True)
```

Support Vector Machines (SVM)

```
>>> from sklearn.svm import SVC
>>> svc = SVC(kernel='linear')
```

Naive Bayes

```
>>> from sklearn.naive_bayes import GaussianNB
>>> gnb = GaussianNB()
```

KNN

```
>>> from sklearn import neighbors
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
```

Unsupervised Learning Estimators

Principal Component Analysis (PCA)

```
>>> from sklearn.decomposition import PCA
>>> pca = PCA(n_components=0.95)
```

K Means

```
>>> from sklearn.cluster import KMeans
>>> k_means = KMeans(n_clusters=3, random_state=0)
```

Model Fitting

Supervised learning

```
>>> lr.fit(X, y)
>>> knn.fit(X_train, y_train)
>>> svc.fit(X_train, y_train)
```

Fit the model to the data

Unsupervised Learning

```
>>> k_means.fit(X_train)
>>> pca_model = pca.fit_transform(X_train)
```

Fit the model to the data

Fit to data, then transform it

Prediction

Supervised Estimators

```
>>> y_pred = svc.predict(np.random.random((2,5)))
>>> y_pred = lr.predict(X_test)
>>> y_pred = knn.predict_proba(X_test)
```

Predict labels

Predict labels

Estimate probability of a label

Unsupervised Estimators

```
>>> y_pred = k_means.predict(X_test)
```

Predict labels in clustering algos

Preprocessing The Data

Standardization

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(X_train)
>>> standardized_X = scaler.transform(X_train)
>>> standardized_X_test = scaler.transform(X_test)
```

Normalization

```
>>> from sklearn.preprocessing import Normalizer
>>> scaler = Normalizer().fit(X_train)
>>> normalized_X = scaler.transform(X_train)
>>> normalized_X_test = scaler.transform(X_test)
```

Binarization

```
>>> from sklearn.preprocessing import Binarizer
>>> binarizer = Binarizer(threshold=0.0).fit(X)
>>> binary_X = binarizer.transform(X)
```

Encoding Categorical Features

```
>>> from sklearn.preprocessing import LabelEncoder
>>> enc = LabelEncoder()
>>> y = enc.fit_transform(y)
```

Imputing Missing Values

```
>>> from sklearn.preprocessing import Imputer
>>> imp = Imputer(missing_values=0, strategy='mean', axis=0)
>>> imp.fit_transform(X_train)
```

Generating Polynomial Features

```
>>> from sklearn.preprocessing import PolynomialFeatures
>>> poly = PolynomialFeatures(5)
>>> poly.fit_transform(X)
```

Evaluate Your Model's Performance

Classification Metrics

Accuracy Score

```
>>> knn.score(X_test, y_test)
```

Estimator score method

```
>>> from sklearn.metrics import accuracy_score
>>> accuracy_score(y_test, y_pred)
```

Metric scoring functions

Classification Report

```
>>> from sklearn.metrics import classification_report
>>> print(classification_report(y_test, y_pred))
```

Precision, recall, f1-score and support

Confusion Matrix

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

Regression Metrics

Mean Absolute Error

```
>>> from sklearn.metrics import mean_absolute_error
>>> y_true = [3, -0.5, 2]
>>> mean_absolute_error(y_true, y_pred)
```

Mean Squared Error

```
>>> from sklearn.metrics import mean_squared_error
>>> mean_squared_error(y_test, y_pred)
```

R^2 Score

```
>>> from sklearn.metrics import r2_score
>>> r2_score(y_true, y_pred)
```

Clustering Metrics

Adjusted Rand Index

```
>>> from sklearn.metrics import adjusted_rand_score
>>> adjusted_rand_score(y_true, y_pred)
```

Homogeneity

```
>>> from sklearn.metrics import homogeneity_score
>>> homogeneity_score(y_true, y_pred)
```

V-measure

```
>>> from sklearn.metrics import v_measure_score
>>> metrics.v_measure_score(y_true, y_pred)
```

Cross-Validation

```
>>> from sklearn.cross_validation import cross_val_score
>>> print(cross_val_score(knn, X_train, y_train, cv=4))
>>> print(cross_val_score(lr, X, y, cv=2))
```

Tune Your Model

Grid Search

```
>>> from sklearn.grid_search import GridSearchCV
>>> params = {"n_neighbors": np.arange(1,3),
              "metric": ["euclidean", "cityblock"]}
>>> grid = GridSearchCV(estimator=knn,
                        param_grid=params)
>>> grid.fit(X_train, y_train)
>>> print(grid.best_score_)
>>> print(grid.best_estimator_.n_neighbors)
```

Randomized Parameter Optimization

```
>>> from sklearn.grid_search import RandomizedSearchCV
>>> params = {"n_neighbors": range(1,5),
              "weights": ["uniform", "distance"]}
>>> rsearch = RandomizedSearchCV(estimator=knn,
                                 param_distributions=params,
                                 cv=4,
                                 n_iter=8,
                                 random_state=5)
>>> rsearch.fit(X_train, y_train)
>>> print(rsearch.best_score_)
```

DataCamp

Learn Python for Data Science Interactively

Image: DataCamp

scikit-learn: A Machine Learning Library in Python

- ***Iris* flower dataset** [UCI ML Repository] [scikit-learn] [API]
  - Classes (#: **3**): *Iris setosa*, *Iris versicolor*, and *Iris virginica*
  - Attributes: **4** real numbers
    - Sepal length (unit: cm), sepal width (unit: cm)
    - Petal length (unit: cm), petal width (unit: cm)
    - Note) The values are quantized in the unit of 0.1 cm.
    - Note) In Korean, Iris - 붓꽃, Sepal - 꽃받침, and Petal - 꽃잎
  - The number of data: **150** (50 for each class)
  - Example) Loading the *Iris* flower dataset

```python
from sklearn import datasets

iris = datasets.load_iris()

print(iris.target_names)  # ['setosa' 'versicolor' 'virginica']
print(iris.feature_names) # ['sepal length (cm)', ...]
print(iris.data.shape)    # (150, 4)
print(iris.target.shape)  # (150,)
```

Image: Wikipedia

scikit-learn: A Machine Learning Library in Python

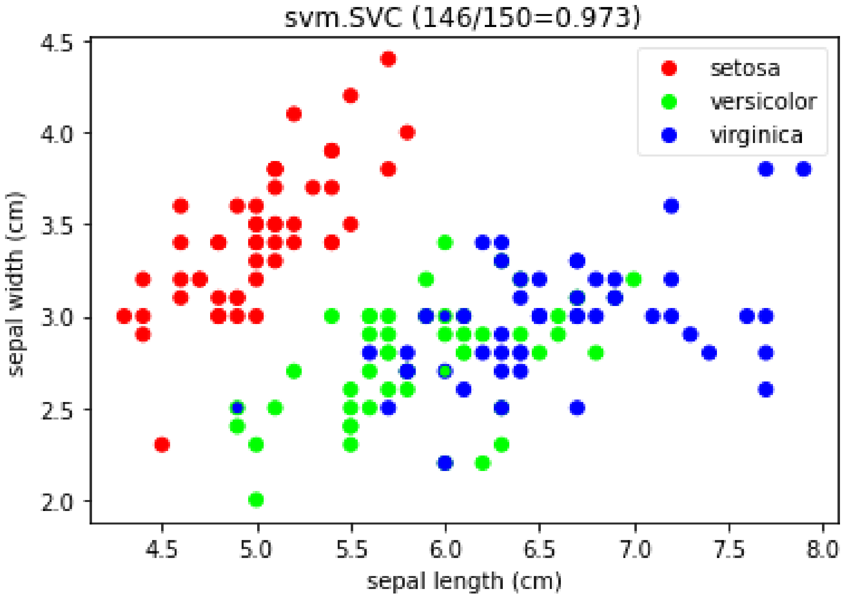
- **How to use (3 steps)**: Instantiation → training (fit) → inference/testing (predict)
- Example) Iris flower classification using support vector machine (SVM)
  - Accuracy with 4 attributes: **0.973** (146/150)
  - Accuracy with 2 attributes (sepal width and length): **0.820** (123/150)

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn import (datasets, svm)
from matplotlib.lines import Line2D # For the custom legend

# Load a dataset
iris = datasets.load_iris()

# Train a model
model = svm.SVC()                  # Accuracy: 0.973 (146/150)
model.fit(iris.data, iris.target)  # Try 'iris.data[:,0:2]' (Accuracy: 0.820)

# Test the model
predict = model.predict(iris.data) # Try 'iris.data[:,0:2]' (Accuracy: 0.820)
n_correct = sum(predict == iris.target)
accuracy = n_correct / len(iris.data)
```

scikit-learn: A Machine Learning Library in Python

- Example) Iris flower classification using support vector machine (SVM)

```python
# Visualize testing results
cmap = np.array([(1, 0, 0), (0, 1, 0), (0, 0, 1)])
clabel = [Line2D([0], [0], marker='o', lw=0, label=iris.target_names[i], color=cmap[i]) for i in range(len(cmap))]
for (x, y) in [(0, 1), (2, 3)]:
    plt.figure ()
    plt.title(f'svm.SVC ({n_correct}/{len(iris.data)}={accuracy:.3f})')
    plt.scatter(iris.data[:,x], iris.data[:,y], c=cmap[iris.target], edgecolors=cmap[predict])
    plt.xlabel(iris.feature_names[x])
    plt.ylabel(iris.feature_names[y])
    plt.legend(handles=clabel, framealpha=0.5)
plt.show()
```



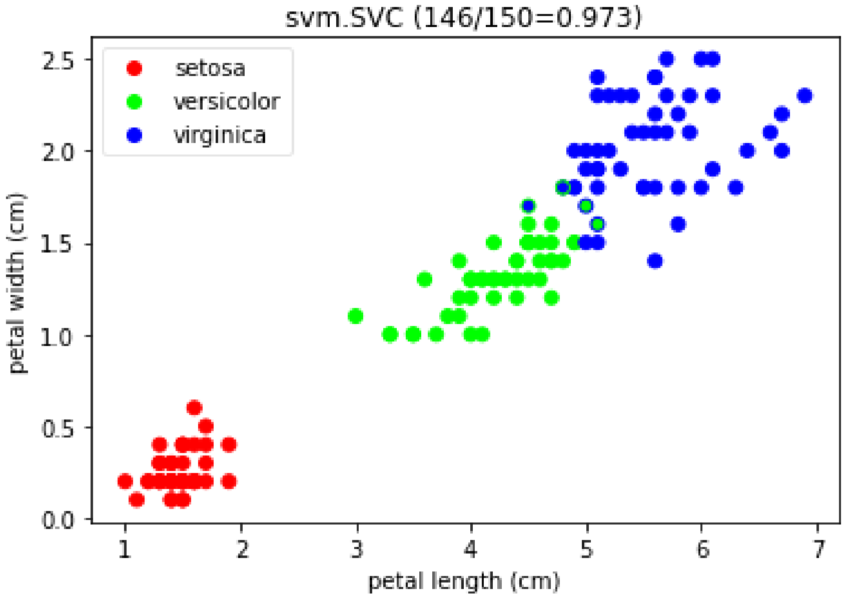


Table of Contents

- Machine Learning
- scikit-learn
- Classification
- Regression
- Clustering
- Machine Learning FAQ

Classification

- **Binary classification**
  - Classifying observations into one of two groups (1 or 0; `True` or `False`; positive or negative)
  - e.g. Detecting spam e-mails

    (Determining whether an e-mail is *spam* or *normal*)
  - **Accuracy**: # of correct answers / # of data
    - e.g. Accuracy of spam detection with 100 e-mails (spam: 10, normal: 90)

```python
def is_spam_always_true(text):
    return True  # Accuracy: 0.1 (10/100)

def is_spam_always_false(text):
    return False # Accuracy: 0.9 (90/100)
```

    - Accuracy is not a good measure for imbalanced data (T: 10, F: 90).

Classification

- **Binary classification**
  - Confusion matrix: 2D representation of correctness per classes

		Predicted class	
		+	−
Actual class	P	**TP** true positive	**FN** false negative (type II error)
	N	**FP** false positive (type I error)	**TN** true negative

Note) $\mathrm{P} = \mathrm{TP} + \mathrm{FN}$
$\mathrm{N} = \mathrm{FP} + \mathrm{TN}$

$$\mathrm{ACC} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{P} + \mathrm{N}}$$

Accuracy

$$\mathrm{BA} = \frac{1}{2}\left(\frac{\mathrm{TP}}{\mathrm{P}} + \frac{\mathrm{TN}}{\mathrm{N}}\right)$$

Balanced accuracy

$$\mathrm{RE} = \frac{\mathrm{TP}}{\mathrm{P}} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

Recall, hit rate, sensitivity,
true positive rate

$$\mathrm{PR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}$$

Precision,
positive predictive rate

$$\mathrm{F1} = 2 \cdot \frac{\mathrm{RE} \cdot \mathrm{PR}}{\mathrm{RE} + \mathrm{PR}}$$

F1-measure
harmonic mean of precision and recall

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{N}} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}}$$

Fall-out*,
false positive rate*

Note) higher* is worse.

Classification

- **Binary classification**
  - Confusion matrix: 2D representation of correctness per classes

		Predicted class	
		+	**−**
Actual class	**P**	**TP** true positive	**FN** false negative (type II error)
	N	**FP** false positive (type I error)	**TN** true negative

Note) $\mathrm{P} = \mathrm{TP} + \mathrm{FN}$
$\mathrm{N} = \mathrm{FP} + \mathrm{TN}$

e.g. Accuracy of spam detection
with 100 e-mails (spam: 10, normal: 90)

		Predicted	
		T	**F**
Actual	**S**	10	0
	N	90	0

Always `True`

		Predicted	
		T	**F**
Actual	**S**	0	10
	N	0	90

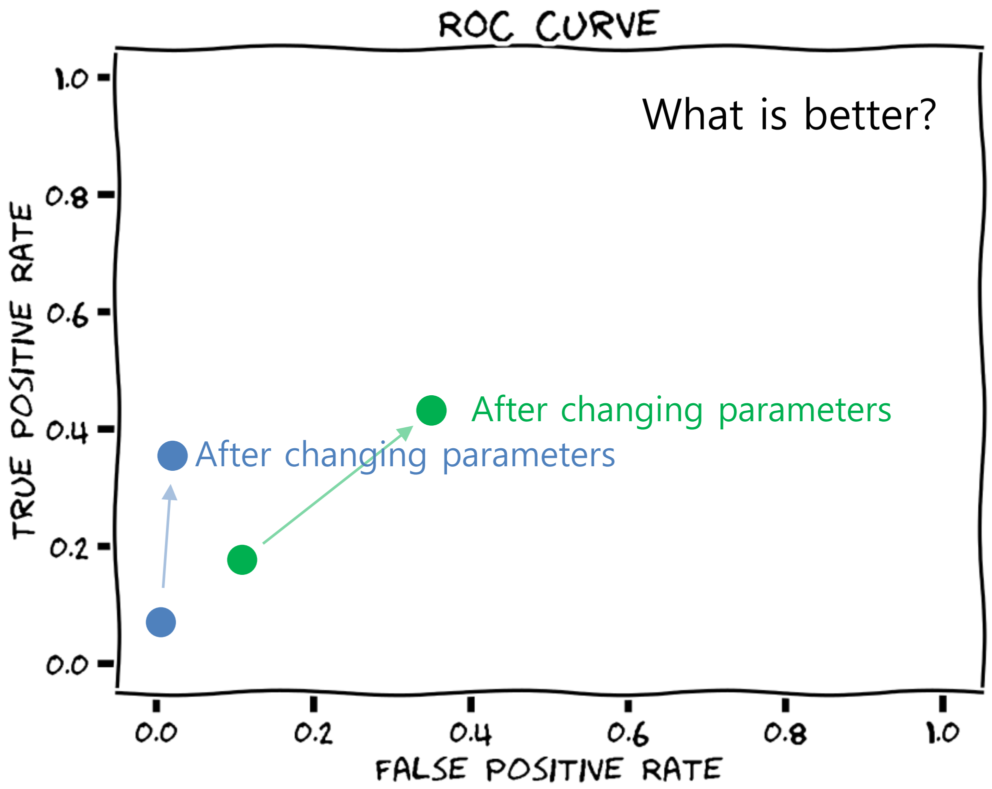
Always `False`

$\mathrm{ACC} = \dfrac{\mathrm{TP} + \mathrm{TN}}{\mathrm{P} + \mathrm{N}}$	**Accuracy**	0.1 vs. **0.9**
$\mathrm{BA} = \dfrac{1}{2}\left(\dfrac{\mathrm{TP}}{\mathrm{P}} + \dfrac{\mathrm{TN}}{\mathrm{N}}\right)$	**Balanced accuracy**	0.5 vs. 0.5
$\mathrm{RE} = \dfrac{\mathrm{TP}}{\mathrm{P}} = \dfrac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$	**Recall**, hit rate, sensitivity, true positive rate	**1.0** vs. 0.0
$\mathrm{PR} = \dfrac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}$	**Precision**, positive predictive rate	0.1 vs. 0.0
$\mathrm{F1} = 2 \cdot \dfrac{\mathrm{RE} \cdot \mathrm{PR}}{\mathrm{RE} + \mathrm{PR}}$	**F1-measure** harmonic mean of precision and recall	0.18 vs. 0.0
$\mathrm{FPR} = \dfrac{\mathrm{FP}}{\mathrm{N}} = \dfrac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}}$	Fall-out*, **false positive rate***	1.0 vs. **0.0**

Note) higher* is worse.

Classification

- **Binary classification**
  - **Precision and recall**
  - **ROC curve** (Receiver Operating Characteristic curve)
    - A 2D performance plot for varying internal parameters



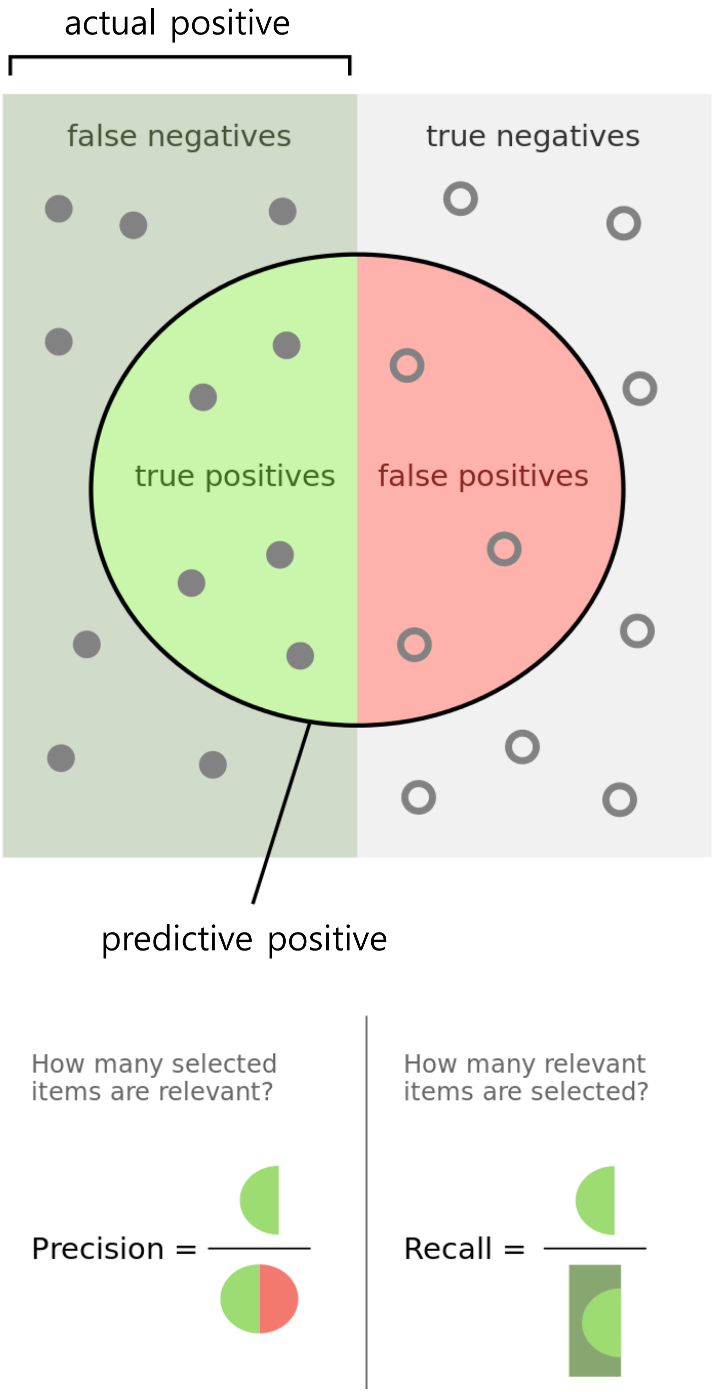
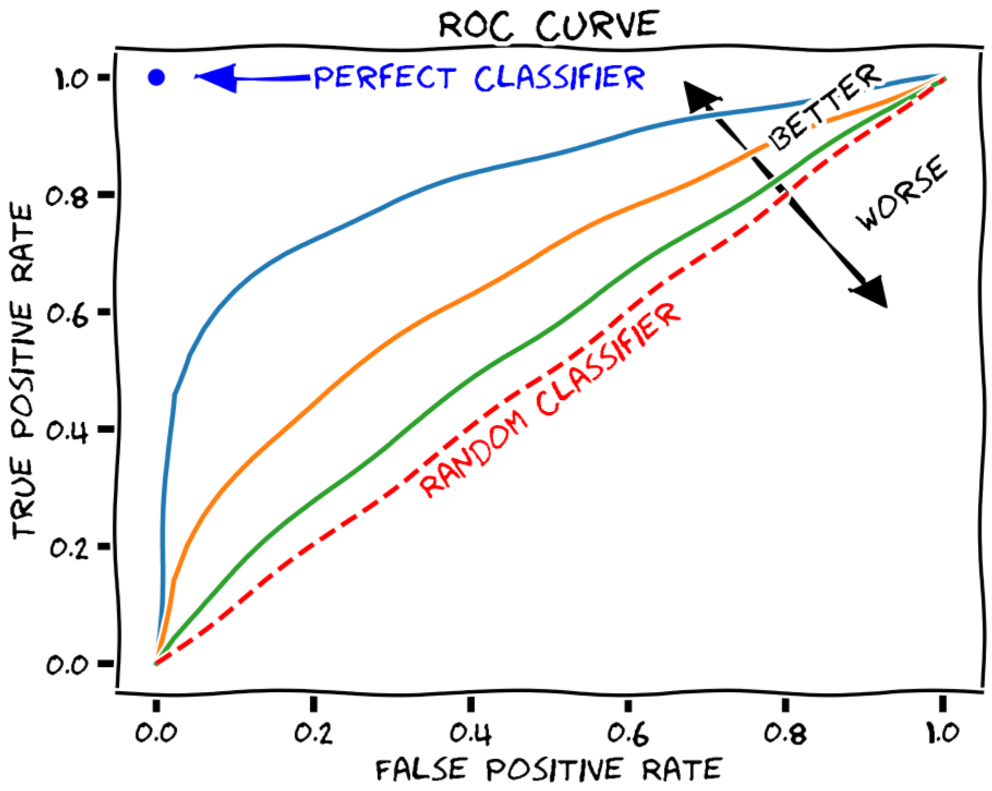


Image: Wikipedia (precision and recall and ROC)

Classification

- **Binary classification**
  - **Precision and recall**
  - **ROC curve** (Receiver Operating Characteristic curve)
    - A 2D performance plot for varying internal parameters



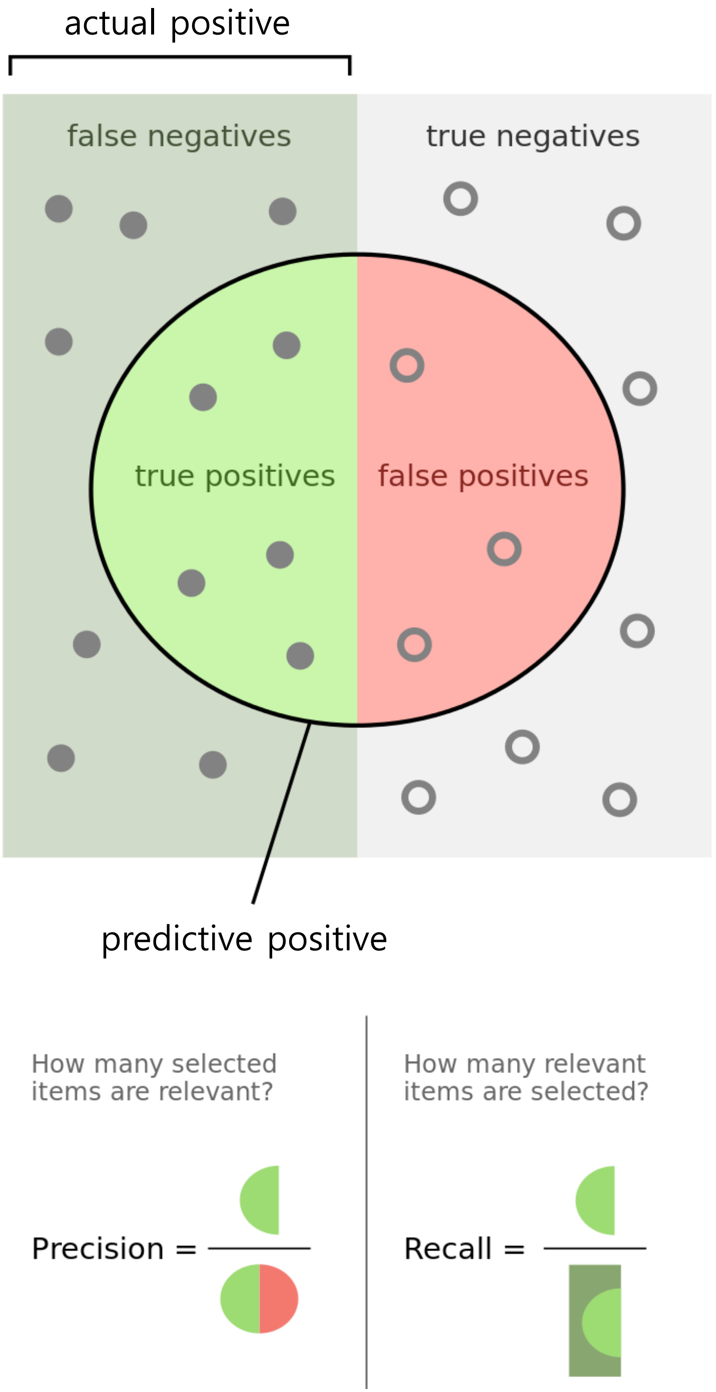
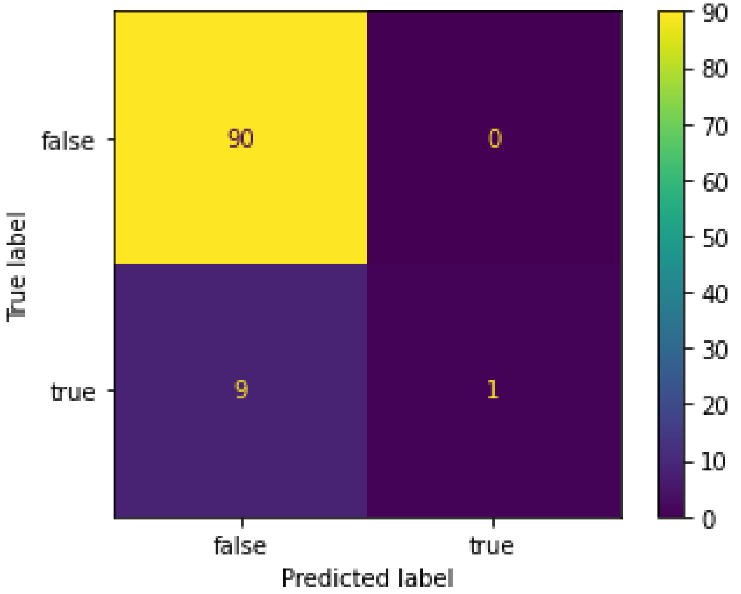


Image: Wikipedia (precision and recall and ROC)

Classification

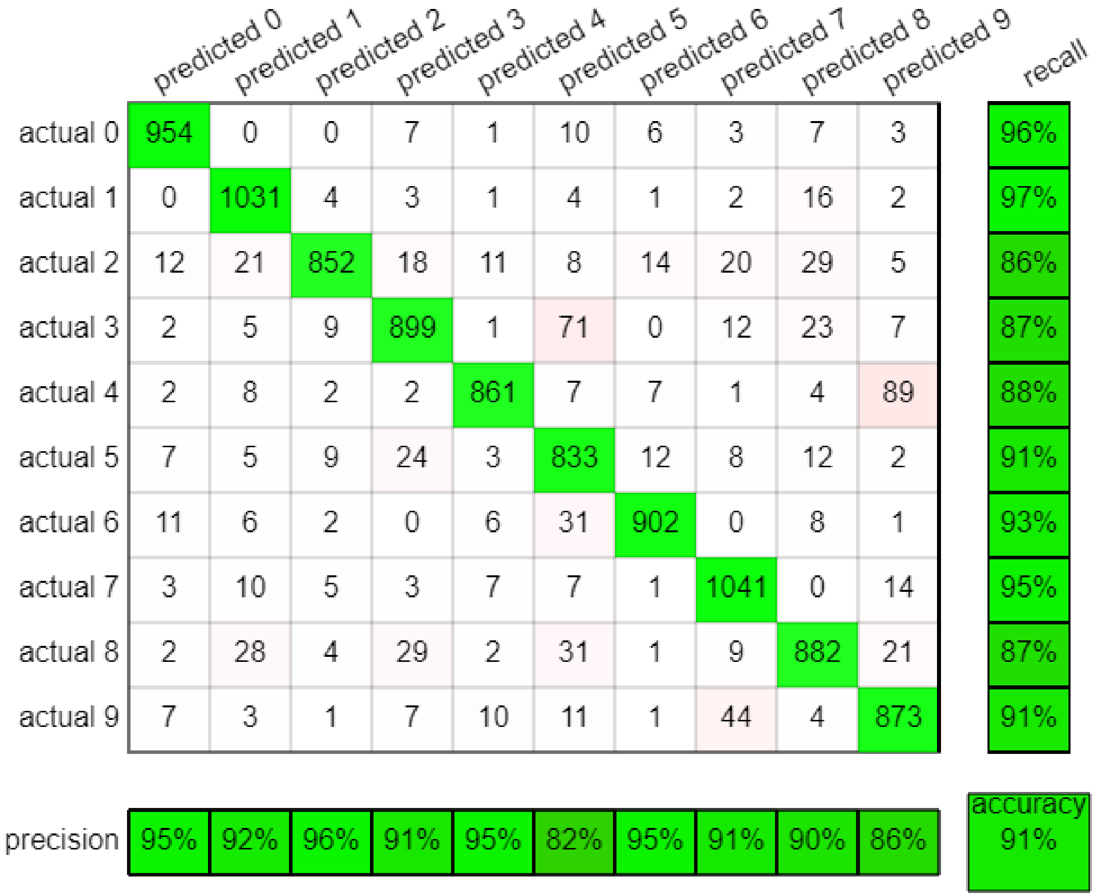
- **Binary classification**
  - Example) Evaluating binary classification

```python
from sklearn import metrics

# Example ground truth and its prediction
y_true = [False] * 90 + [True] * 10  # True labels
y_pred = [False] * 99 + [True] * 1   # Predicted labels

print(metrics.accuracy_score(y_true, y_pred))                  # 0.91
print(metrics.balanced_accuracy_score(y_true, y_pred))         # 0.55
print(metrics.precision_score(y_true, y_pred))                 # 1.0  = 1 / 1
print(metrics.recall_score(y_true, y_pred))                    # 0.1  = 1 / 10
print(metrics.f1_score(y_true, y_pred))                        # 0.18 = 2 * (1.0*0.1) / (1.0+0.1)
print(metrics.precision_recall_fscore_support(y_true, y_pred)) # ...
print(metrics.classification_report(y_true, y_pred))
#               precision    recall  f1-score   support
#        False       0.91      1.00      0.95        90
#         True       1.00      0.10      0.18        10
#     accuracy                           0.91       100
#    macro avg       0.95      0.55      0.57       100
# weighted avg       0.92      0.91      0.88       100
conf_matx = metrics.confusion_matrix(y_true, y_pred)           # np.array([[90, 0], [9, 1]], dtype=int64)
conf_disp = metrics.ConfusionMatrixDisplay(conf_matx, display_labels=['false', 'true'])
conf_disp.plot()
```



Classification

- **Multiclass classification**
  - Classifying observations into one of three or more classes
  - e.g. Confusion matrix of visual digit classification



	predicted 0	predicted 1	predicted 2	predicted 3	predicted 4	predicted 5	predicted 6	predicted 7	predicted 8	predicted 9	recall
actual 0	954	0	0	7	1	10	6	3	7	3	96%
actual 1	0	1031	4	3	1	4	1	2	16	2	97%
actual 2	12	21	852	18	11	8	14	20	29	5	86%
actual 3	2	5	9	899	1	71	0	12	23	7	87%
actual 4	2	8	2	2	861	7	7	1	4	89	88%
actual 5	7	5	9	24	3	833	12	8	12	2	91%
actual 6	11	6	2	0	6	31	902	0	8	1	93%
actual 7	3	10	5	3	7	7	1	1041	0	14	95%
actual 8	2	28	4	29	2	31	1	9	882	21	87%
actual 9	7	3	1	7	10	11	1	44	4	873	91%
precision	95%	92%	96%	91%	95%	82%	95%	91%	90%	86%	accuracy 91%

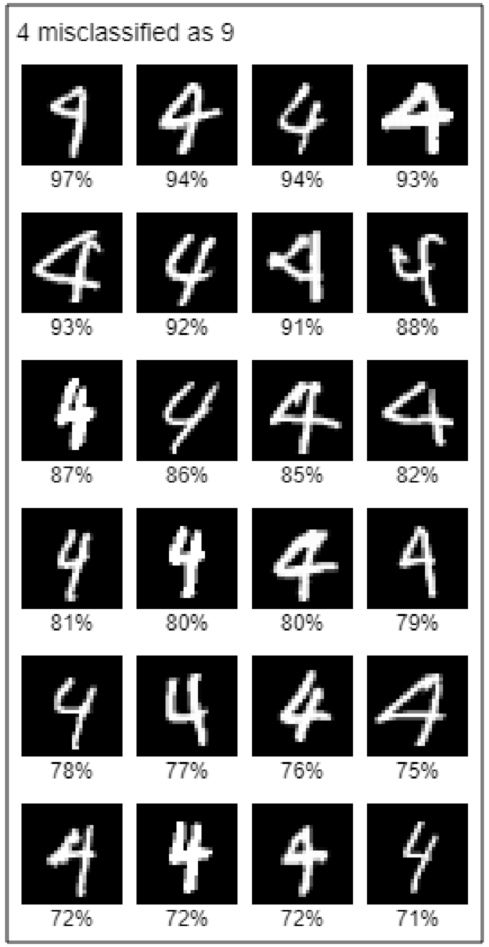
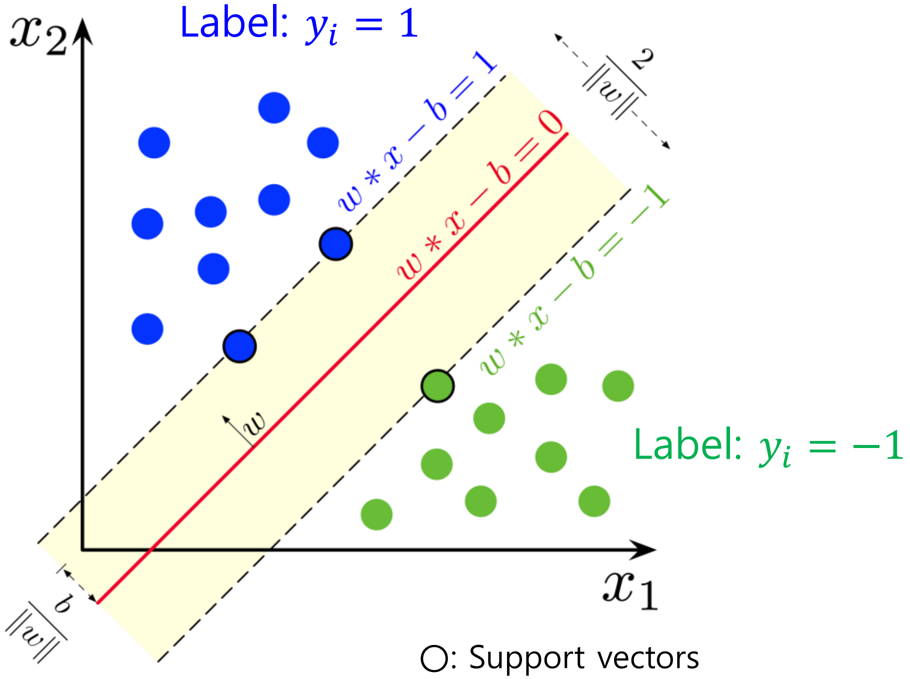


Image: Machine Learning for Artists

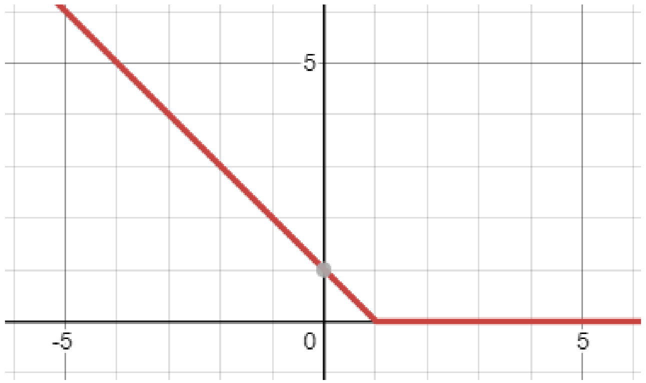
Classification

- **Support vector machine (SVM)** [scikit-learn] [API]
  - Supervised learning models (usually for classification and regression)
  - A non-probabilistic (deterministic) binary linear classifier
    - Finding the maximum-margin hyperplane (~ boundary; line in 2D, plane in 3D)

      $\mathbf{w}^\intercal \mathbf{x}_i - b = 0$ (signed distance) → $y_{\mathrm{i}}(\mathbf{w}^\intercal \mathbf{x}_i - b)$ (per-class signed distance)
    - Optimization) $\operatorname{argmin}_{\mathbf{w},b} \frac{1}{n} \sum_{i=0}^{n} h\big(y_i(\mathbf{w}^\intercal \mathbf{x}_i - b)\big) + \lambda \|\mathbf{w}\|^2$ where $h(x) = \max(0, 1 - x)$



○: Support vectors



Hinge loss: $h(x) = \max(0, 1 - x)$

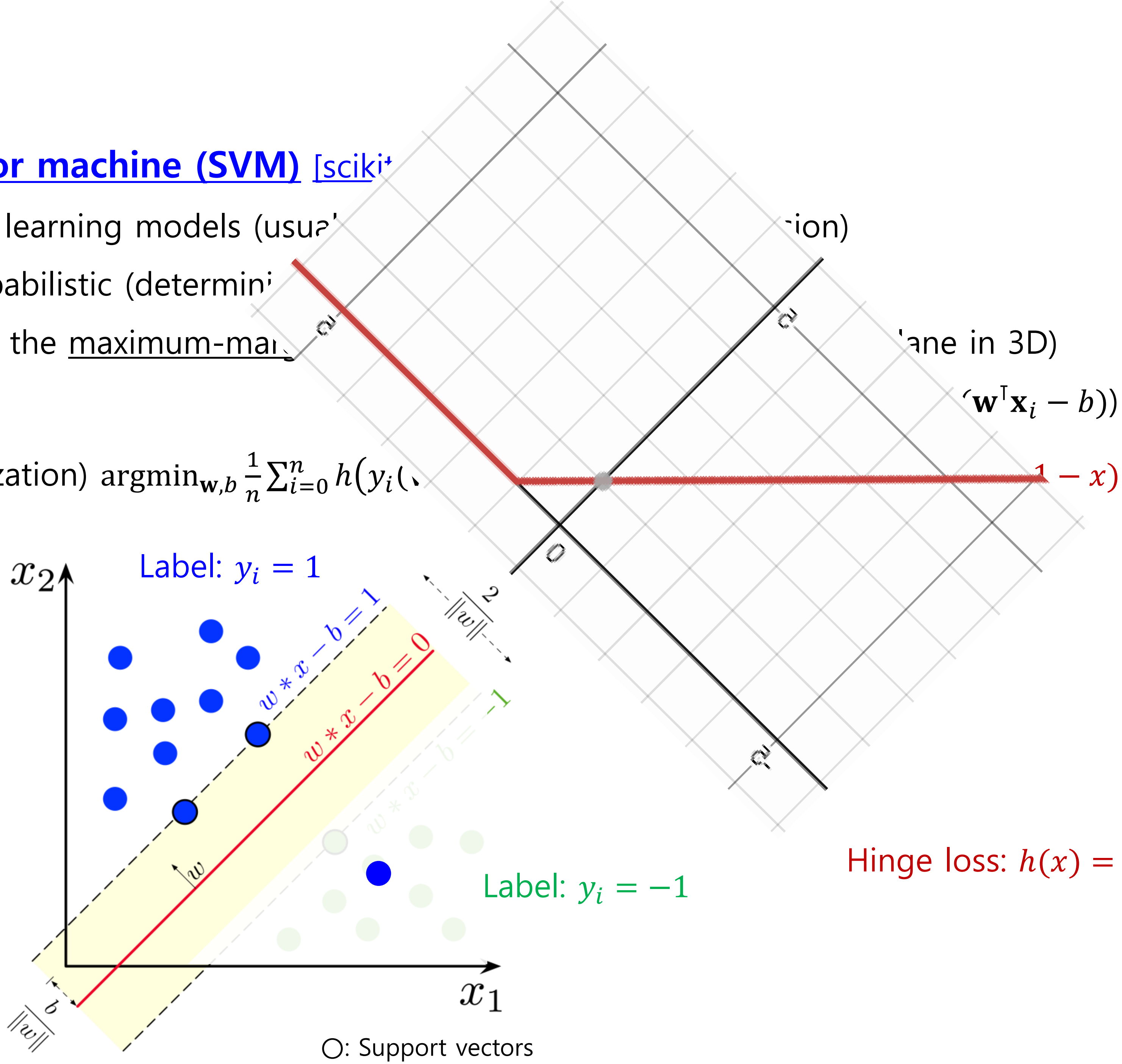
Image: Wikipedia

Classification

- **Support vector machine (SVM)** [sciki‘
  - Supervised learning models (usua’ ·ion)
  - A non-probabilistic (determini
    - Finding the maximum-mar ’ane in 3D)

      ‘$\mathbf{w}^\top \mathbf{x}_i - b$))
    - Optimization) $\operatorname{argmin}_{\mathbf{w},b} \frac{1}{n}\sum_{i=0}^{n} h\big(y_i($

      $1 - x)$



Hinge loss: $h(x) = \max(0, 1 - x)$

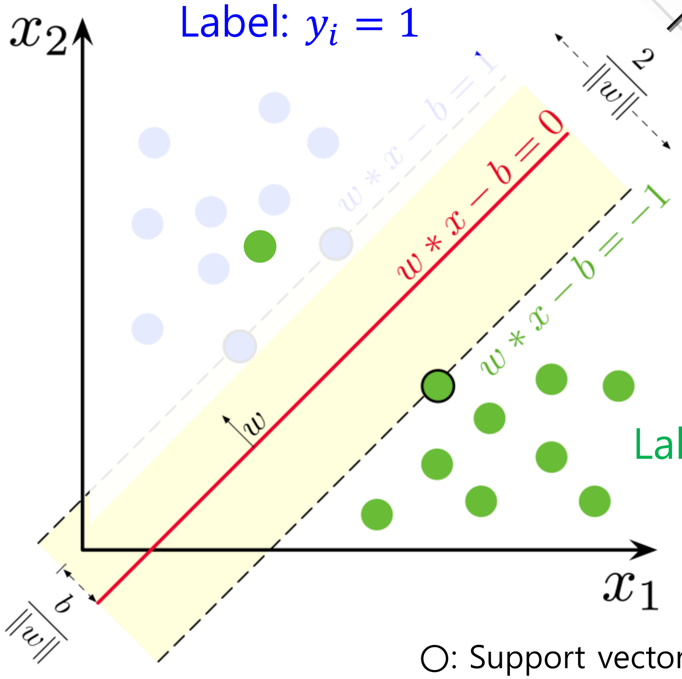
○: Support vectors

Image: Wikipedia

Classification

- **Support vector machine (SVM)** [scikit
  - Supervised learning models (usua ion)
  - A non-probabilistic (determin
    - Finding the maximum-marg lane in 3D)

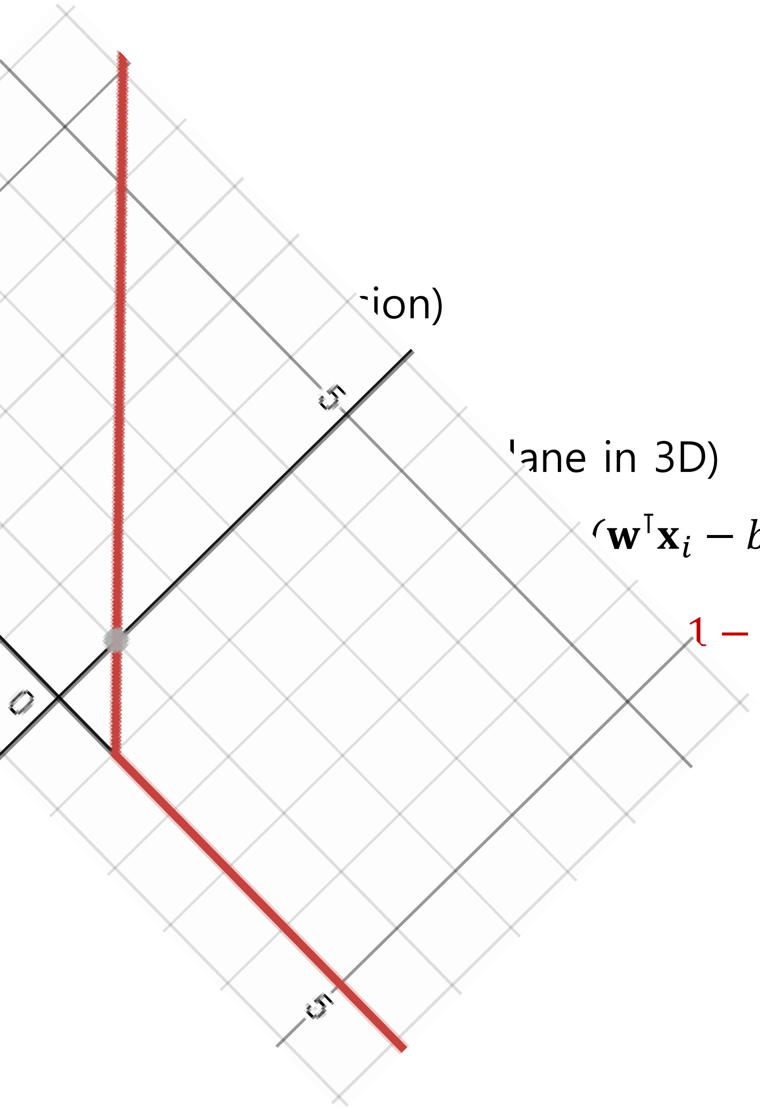
      $\mathbf{w}^\top \mathbf{x}_i - b))$
    - Optimization) $\text{argmin}_{\mathbf{w},b} \frac{1}{n}\sum_{i=0}^{n} h(y_i($ $1 - x)$



Label: $y_i = 1$

Label: $y_i = -1$

○: Support vectors

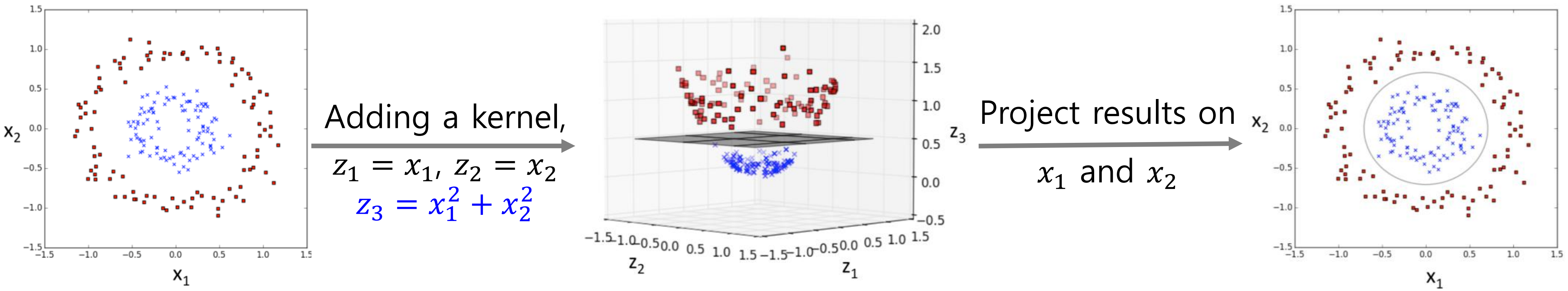


Hinge loss: $h(x) = \max(0, 1 - x)$

Image: Wikipedia

Classification

- **Support vector machine (SVM)** [scikit-learn] [API]
  - The kernel trick enables *non-linear* classification.



Adding a kernel,

$z_1 = x_1$, $z_2 = x_2$

$z_3 = x_1^2 + x_2^2$

Project results on

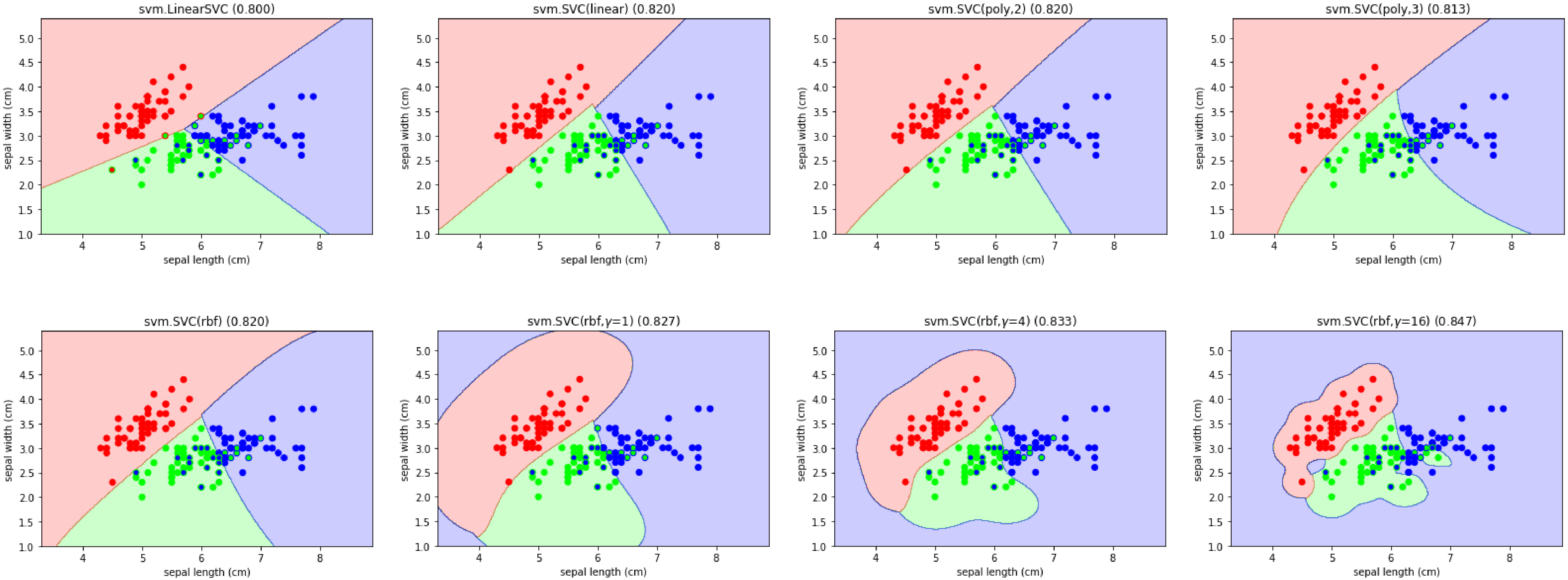
x_1 and x_2

  - Advantages
    - Effective in high-dimensional spaces
      - Still effective in cases where # of dimensions > # of samples
    - Memory-efficient because it uses a subset of data (called support vectors) in training
    - Customizable by specifying kernel functions
  - Disadvantages
    - Does not directly provide probabilistic estimates
    - Often fails in highly noisy data and large-scale data

Image: Raschka's book

Classification

- **Support vector machine (SVM)** [scikit-learn] [API]
  - – Example) SVM classifiers with various kernels (1/3)



svm.LinearSVC (0.800)

svm.SVC(linear) (0.820)

svm.SVC(poly,2) (0.820)

svm.SVC(poly,3) (0.813)

svm.SVC(rbf) (0.820)

svm.SVC(rbf,γ=1) (0.827)

svm.SVC(rbf,γ=4) (0.833)

svm.SVC(rbf,γ=16) (0.847)

- **Support vector machine (SVM)** [scikit-learn] [API]
  - Example) SVM classifiers with various kernels (2/3)

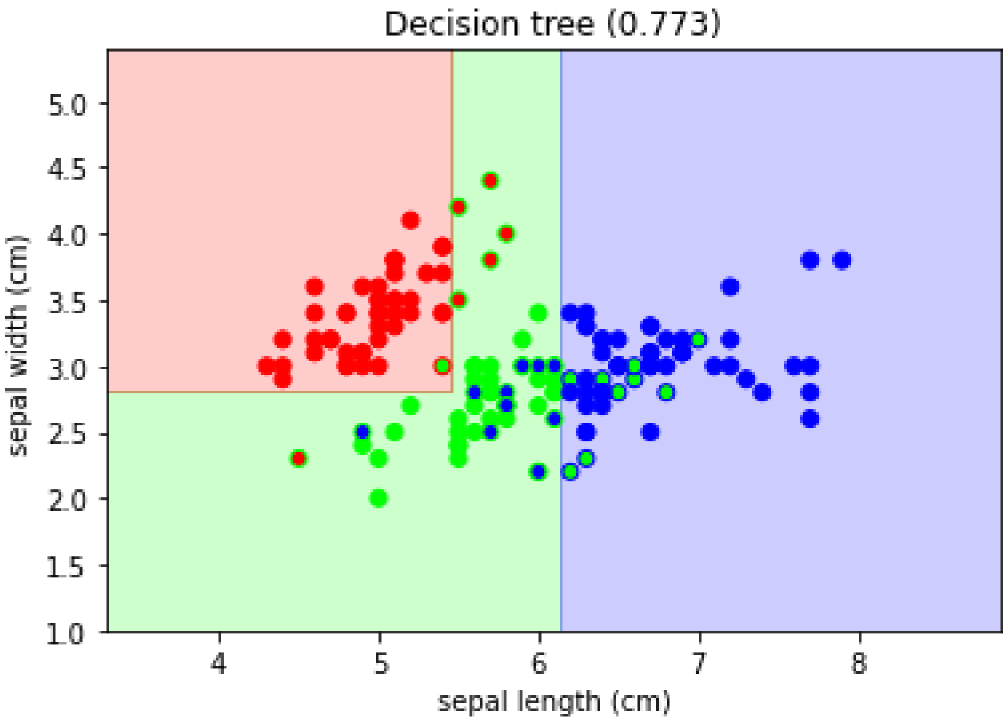
```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn import (datasets, svm, metrics)
from matplotlib.colors import ListedColormap

# Load a dataset partially
iris = datasets.load_iris()
iris.data = iris.data[:,0:2]
iris.feature_names = iris.feature_names[0:2]
iris.color = np.array([(1, 0, 0), (0, 1, 0), (0, 0, 1)])

# Instantiate classification models
models = [
    {'name': 'svm.LinearSVC',               'obj': svm.LinearSVC()},
    {'name': 'svm.SVC(linear)',             'obj': svm.SVC(kernel='linear')},
    {'name': 'svm.SVC(poly,2)',             'obj': svm.SVC(kernel='poly', degree=2)},
    {'name': 'svm.SVC(poly,3)',             'obj': svm.SVC(kernel='poly')},
    {'name': 'svm.SVC(poly,4)',             'obj': svm.SVC(kernel='poly', degree=4)},
    {'name': 'svm.SVC(rbf)',                'obj': svm.SVC(kernel='rbf')},
    {'name': 'svm.SVC(rbf,$\gamma$=1)',     'obj': svm.SVC(kernel='rbf', gamma=1)},
    {'name': 'svm.SVC(rbf,$\gamma$=4)',     'obj': svm.SVC(kernel='rbf', gamma=4)},
    {'name': 'svm.SVC(rbf,$\gamma$=16)',    'obj': svm.SVC(kernel='rbf', gamma=16)},
    {'name': 'svm.SVC(rbf,$\gamma$=64)',    'obj': svm.SVC(kernel='rbf', gamma=64)},
    {'name': 'svm.SVC(sigmoid)',            'obj': svm.SVC(kernel='sigmoid')},
]
```

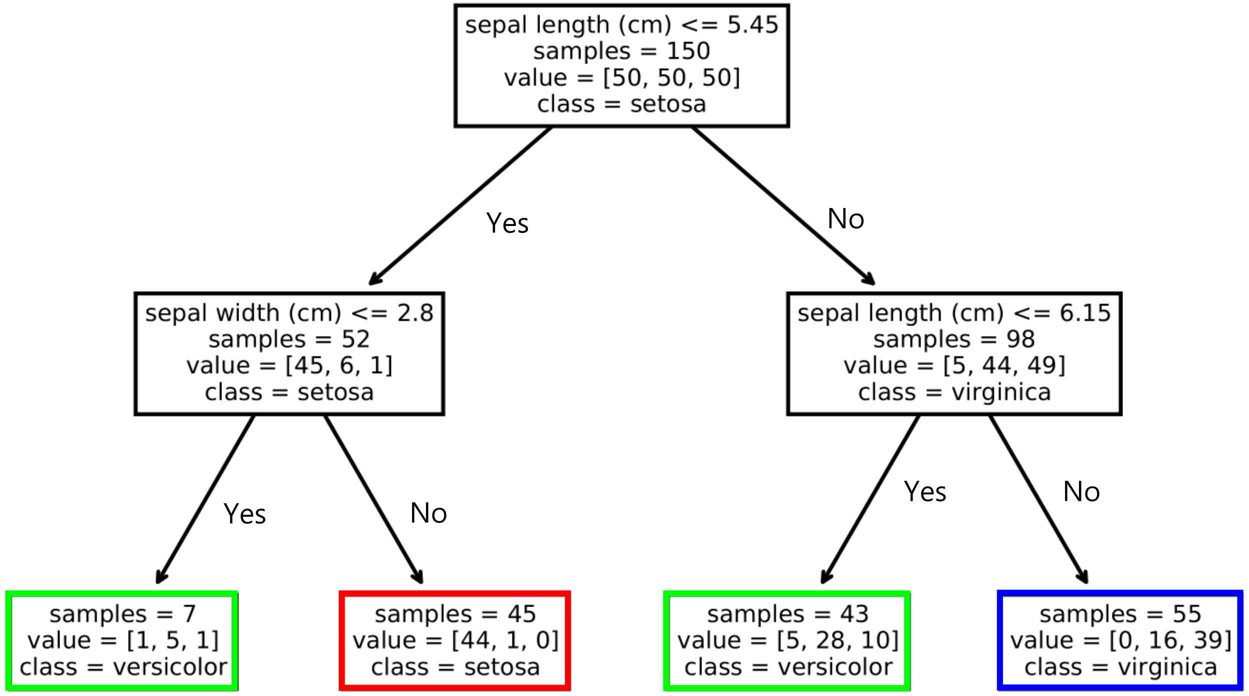
- **Support vector machine (SVM)** [scikit-learn] [API]
  - Example) SVM classifiers with various kernels (2/3)

```
x_min, x_max = iris.data[:, 0].min() - 1, iris.data[:, 0].max() + 1
y_min, y_max = iris.data[:, 1].min() - 1, iris.data[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01), np.arange(y_min, y_max, 0.01))
xy = np.vstack((xx.flatten(), yy.flatten())).T

for model in models:
    # Train a model
    model['obj'].fit(iris.data, iris.target)

    # Test the model
    predict = model['obj'].predict(iris.data)
    model['acc'] = metrics.balanced_accuracy_score(iris.target, predict)

    # Visualize training results (decision boundaries)
    zz = model['obj'].predict(xy)
    plt.figure()
    plt.contourf(xx, yy, zz.reshape(xx.shape), cmap=ListedColormap(iris.color), alpha=0.2)

    # Visualize testing results
    plt.title(model['name'] + f' ({model["acc"]:.3f})')
    plt.scatter(iris.data[:,0], iris.data[:,1], c=iris.color[iris.target], edgecolors=iris.color[predict])
    plt.xlabel(iris.feature_names[0])
    plt.ylabel(iris.feature_names[1])

plt.show()
```

Classification

- **Decision tree** [scikit-learn] [API]
  - A tree-like model for decision with rules and outcomes (~ a if-then-else flowchart)
  - Example) Decision tree classifier training and visualization (1/2)





sepal length (cm) <= 5.45
samples = 150
value = [50, 50, 50]
class = setosa

Yes

No

sepal width (cm) <= 2.8
samples = 52
value = [45, 6, 1]
class = setosa

sepal length (cm) <= 6.15
samples = 98
value = [5, 44, 49]
class = virginica

Yes

No

Yes

No

samples = 7
value = [1, 5, 1]
class = versicolor

samples = 45
value = [44, 1, 0]
class = setosa

samples = 43
value = [5, 28, 10]
class = versicolor

samples = 55
value = [0, 16, 39]
class = virginica

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn import (datasets, tree, metrics)
from matplotlib.colors import ListedColormap

# Load a dataset partially
iris = datasets.load_iris()
iris.data = iris.data[:,0:2]
iris.feature_names = iris.feature_names[0:2]
iris.color = np.array([(1, 0, 0), (0, 1, 0), (0, 0, 1)])

# Train a model
model = tree.DecisionTreeClassifier(max_depth=2) # Try deeper
model.fit(iris.data, iris.target)

# Visualize training results (decision boundaries)
x_min, x_max = iris.data[:, 0].min() - 1, iris.data[:, 0].max() + 1
y_min, y_max = iris.data[:, 1].min() - 1, iris.data[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01), np.arange(y_min, y_max, 0.01))
xy = np.vstack((xx.flatten(), yy.flatten())).T
zz = model.predict(xy)

# Test the model
predict = model.predict(iris.data)
accuracy = metrics.balanced_accuracy_score(iris.target, predict)

# Visualize testing results
plt.figure()
plt.title(f'Decision tree ({accuracy:.3f})')
plt.contourf(xx, yy, zz.reshape(xx.shape), cmap=ListedColormap(iris.color), alpha=0.2)
plt.scatter(iris.data[:,0], iris.data[:,1], c=iris.color[iris.target], edgecolors=iris.color[predict])
plt.xlabel(iris.feature_names[0])
plt.ylabel(iris.feature_names[1])

# Visualize training results (the trained tree)
plt.figure()
tree.plot_tree(model, feature_names=iris.feature_names, class_names=iris.target_names, impurity=False)
plt.show()
```

Classification

- **Decision tree** [scikit-learn] [API]
  - A tree-like model for decision with rules and outcomes (~ a if-then-else flowchart)
  - Advantages (because it is a white box model)
    - Simple to understand, interpret, and visualize
    - Easy to modify the trained tree
    - Easy to be combined with other decision methods
  - Disadvantages
    - Unstable and sensitive
      - A small change in the data can lead a large change in the structure of the optimal decision tree.
    - Prone to fall in overfitting
    - Discontinuous prediction
    - Relatively inaccurate
      - This disadvantage was improved by random forest, a ensemble of decision trees.

Image: Amidi's ML cheatsheet

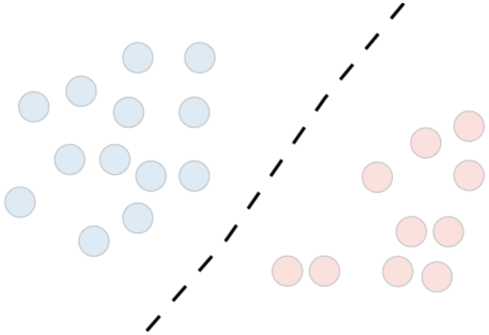
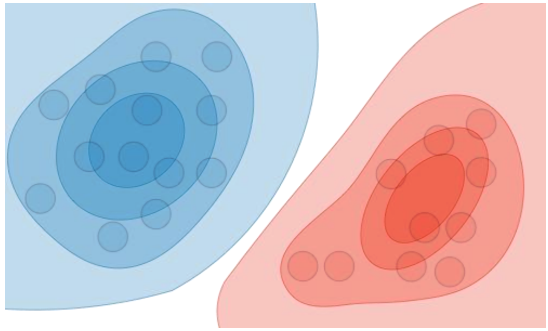
Classification

- **Naïve Bayes classifiers** [scikit-learn] [API]
  - A classifier based on Bayes' theorem under the Naïve assumption

$$P(y|x_1,\ldots,x_n) = \frac{P(y)\,P(x_1,\ldots,x_n|y)}{P(x_1,\ldots,x_n)} = \frac{P(y)\,\prod_{i=1}^{n} p(x_i|y)}{P(x_1,\ldots,x_n)}$$

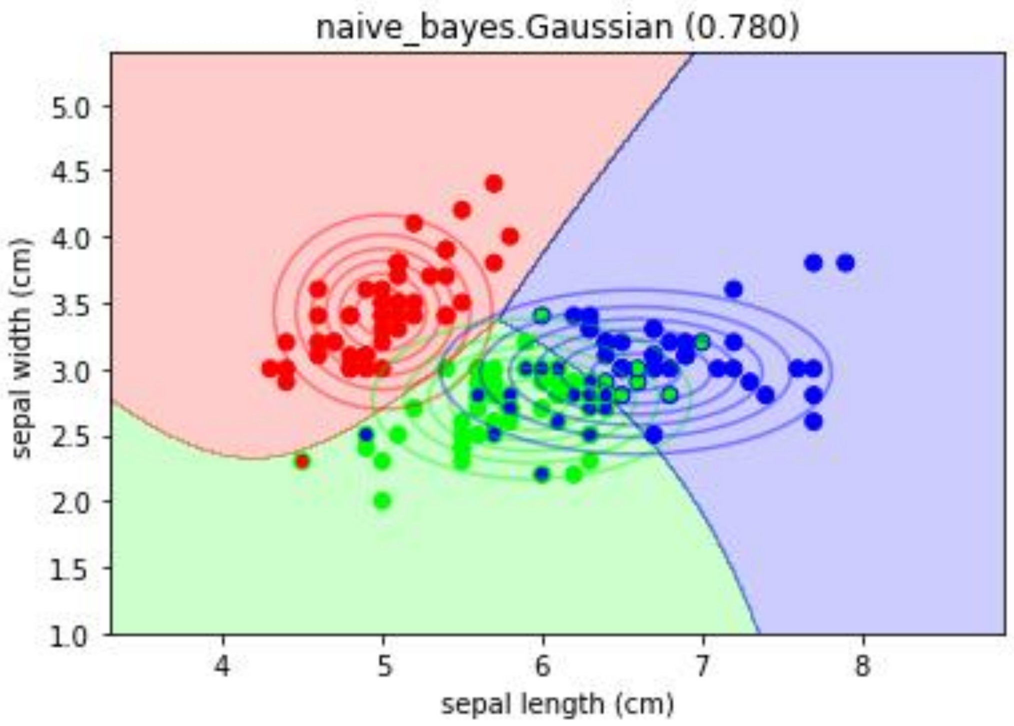
(posterior, likelihood, prior, and marginalization; y: class, x: data)

  - *Gaussian* Naïve Bayes classifier if we assume $p(x|y) = \frac{1}{\sigma_y\sqrt{2\pi}}\exp\left(-\frac{1}{2}\left(\frac{x-\mu_y}{\sigma_y}\right)^2\right)$

	Discriminative model	**Generative model**
Goal	Directly estimate $P(y\|x)$	Estimate $P(x\|y)$, then deduce $P(y\|x)$
Trained	Decision boundaries	Probability distributions of the data
Illustration		
Examples	SVM, decision tree, regression, NN	**Naïve Bayes**, GMM, VAE, GAN

Classification

- **Naïve Bayes classifiers** [scikit-learn] [API]
  - Example) Naïve Bayes classifier training and visualization (1/3)



```
## Class 0
  * Trained prior    = 0.333
  * Manual  prior    = 0.333
  * Trained mean     = [5.006 3.428]
  * Manual  mean     = [5.006 3.428]
  * Trained variance = [0.122 0.141]
  * Manual  variance = [0.122 0.141]
## Class 1
  * Trained prior    = 0.333
  * Manual  prior    = 0.333
  * Trained mean     = [5.936 2.77 ]
  * Manual  mean     = [5.936 2.77 ]
  * Trained variance = [0.261 0.097]
  * Manual  variance = [0.261 0.097]
## Class 2
  * Trained prior    = 0.333
  * Manual  prior    = 0.333
  * Trained mean     = [6.588 2.974]
  * Manual  mean     = [6.588 2.974]
  * Trained variance = [0.396 0.102]
  * Manual  variance = [0.396 0.102]
```

- **Naïve Bayes classifiers** [scikit-learn] [API]
  - Example) Naïve Bayes classifier training and visualization (2/3)

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn import (datasets, naive_bayes, metrics)
from matplotlib.colors import ListedColormap
from scipy.stats import multivariate_normal

# Load a dataset partially
iris = datasets.load_iris()
iris.data = iris.data[:,0:2]
iris.feature_names = iris.feature_names[0:2]
iris.color = np.array([(1, 0, 0), (0, 1, 0), (0, 0, 1)])

# Train a model
model = naive_bayes.GaussianNB()
model.fit(iris.data, iris.target)
#model.class_prior_ = [0.1, 0.6, 0.3] # Try this to give manual prior

# Validate training
for c in range(len(model.classes_)):
    data = iris.data[iris.target == c,:]
    print(f'## Class {c}')
    print('  * Trained prior    = ' + np.array2string(model.class_prior_[c], precision=3))
    print('  * Manual  prior    = ' + '{:.3f}'.format(len(data) / len(iris.data)))
    print('  * Trained mean     = ' + np.array2string(model.theta_[c], precision=3))
    print('  * Manual  mean     = ' + np.array2string(np.mean(data, axis=0), precision=3))
    print('  * Trained variance = ' + np.array2string(model.sigma_[c], precision=3))
    print('  * Manual  variance = ' + np.array2string(np.var(data, axis=0), precision=3))
```

- **Naïve Bayes classifiers** [scikit-learn] [API]
  - Example) Naïve Bayes classifier training and visualization (3/3)

```python
# Visualize training results (decision boundaries)
x_min, x_max = iris.data[:, 0].min() - 1, iris.data[:, 0].max() + 1
y_min, y_max = iris.data[:, 1].min() - 1, iris.data[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01), np.arange(y_min, y_max, 0.01))
xy = np.vstack((xx.flatten(), yy.flatten())).T
zz = model.predict(xy)
plt.contourf(xx, yy, zz.reshape(xx.shape), cmap=ListedColormap(iris.color), alpha=0.2)

# Visualize training results (trained Gaussians)
for c in range(len(model.classes_)):
    likelihood = multivariate_normal(model.theta_[c], np.diag(model.sigma_[c]))
    zz = model.class_prior_[c] * likelihood.pdf(xy)
    plt.contour(xx, yy, zz.reshape(xx.shape), cmap=ListedColormap(iris.color[c]), alpha=0.4)

# Test the model
predict = model.predict(iris.data)
accuracy = metrics.balanced_accuracy_score(iris.target, predict)

# Visualize testing results
plt.title(f'naive_bayes.Gaussian ({accuracy:.3f})')
plt.scatter(iris.data[:,0], iris.data[:,1], c=iris.color[iris.target], edgecolors=iris.color[predict])
plt.xlabel(iris.feature_names[0])
plt.ylabel(iris.feature_names[1])
plt.show()
```

Classification

- Example) More classifiers

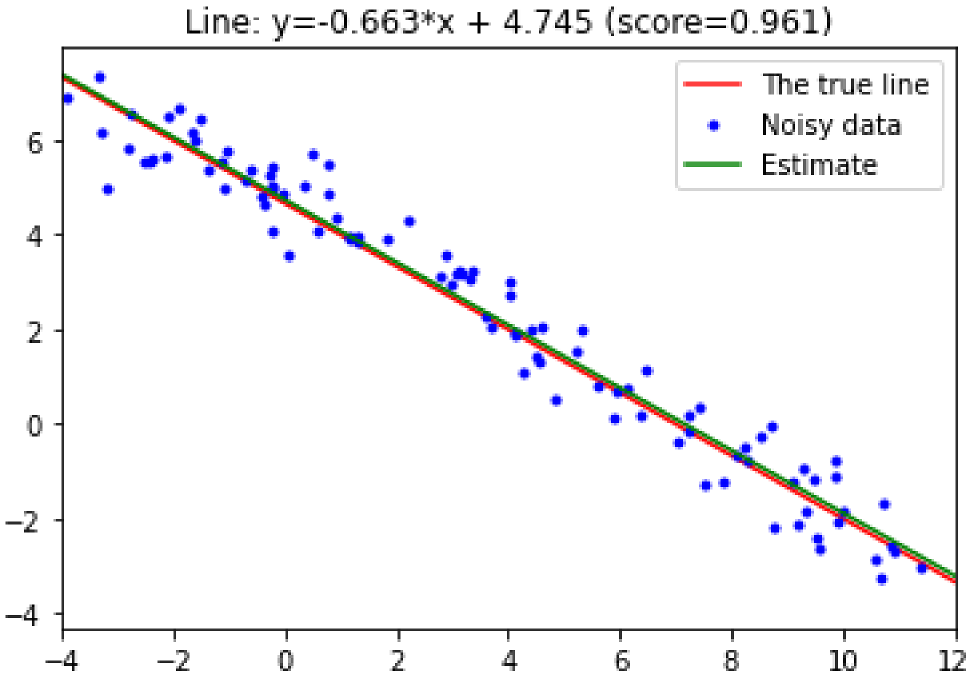
```
# Instantiate classification models
models = [
    {'name': 'linear_model.SGD',            'obj': linear_model.SGDClassifier()},
    {'name': 'naive_bayes.Gaussian',        'obj': naive_bayes.GaussianNB()},
    {'name': 'neural_network.MLP',          'obj': neural_network.MLPClassifier()},
    {'name': 'neighbors.KNN',               'obj': neighbors.KNeighborsClassifier()},

    {'name': 'svm.LinearSVC',               'obj': svm.LinearSVC()},
    {'name': 'svm.SVC(linear)',             'obj': svm.SVC(kernel='linear')},
    {'name': 'svm.SVC(poly,2)',             'obj': svm.SVC(kernel='poly', degree=2)},
    {'name': 'svm.SVC(poly,3)',             'obj': svm.SVC(kernel='poly')},
    {'name': 'svm.SVC(poly,4)',             'obj': svm.SVC(kernel='poly', degree=4)},
    {'name': 'svm.SVC(rbf)',                'obj': svm.SVC(kernel='rbf')},
    {'name': 'svm.SVC(rbf,$\gamma$=1)',     'obj': svm.SVC(kernel='rbf', gamma=1)},
    {'name': 'svm.SVC(rbf,$\gamma$=4)',     'obj': svm.SVC(kernel='rbf', gamma=4)},
    {'name': 'svm.SVC(rbf,$\gamma$=16)',    'obj': svm.SVC(kernel='rbf', gamma=16)},
    {'name': 'svm.SVC(rbf,$\gamma$=64)',    'obj': svm.SVC(kernel='rbf', gamma=64)},
    {'name': 'svm.SVC(sigmoid)',            'obj': svm.SVC(kernel='sigmoid')},

    {'name': 'tree.DecisionTree(2)',        'obj': tree.DecisionTreeClassifier(max_depth=2)},
    {'name': 'tree.DecisionTree(4)',        'obj': tree.DecisionTreeClassifier(max_depth=4)},
    {'name': 'tree.DecisionTree(N)',        'obj': tree.DecisionTreeClassifier()},
    {'name': 'tree.ExtraTree',              'obj': tree.ExtraTreeClassifier()},

    {'name': 'ensemble.RandomForest(10)',   'obj': ensemble.RandomForestClassifier(n_estimators=10)},
    {'name': 'ensemble.RandomForest(100)',  'obj': ensemble.RandomForestClassifier()},
    {'name': 'ensemble.ExtraTrees(10)',     'obj': ensemble.ExtraTreesClassifier(n_estimators=10)},
    {'name': 'ensemble.ExtraTrees(100)',    'obj': ensemble.ExtraTreesClassifier()},
    {'name': 'ensemble.AdaBoost(DTree)',    'obj': ensemble.AdaBoostClassifier(tree.DecisionTreeClassifier())},
]
```

Regression

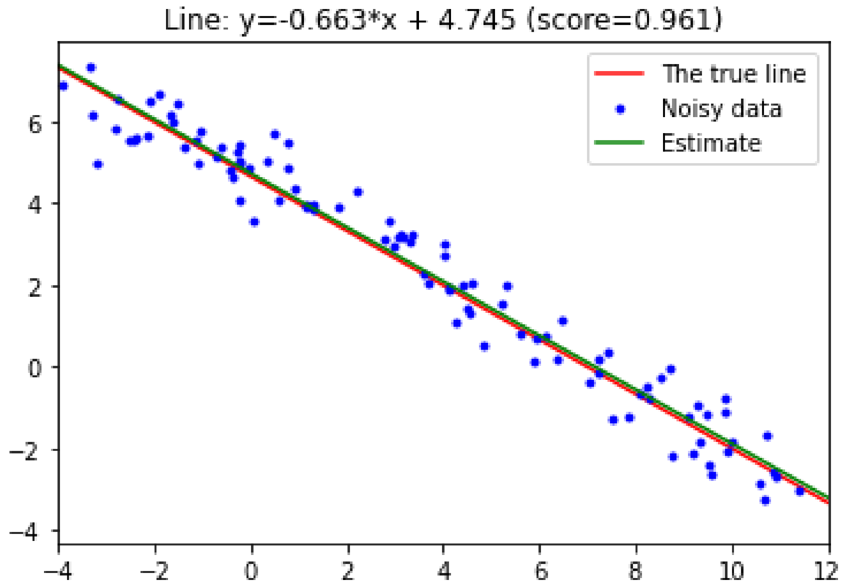
- **Linear regression** [scikit-learn] [API]
  - A linear modeling for the relationship between a scalar response (y) and one or more observed variables (x_i)

$$y = b + a_1 x_1 + \cdots + a_n x_n$$

    - e.g. 2D line fitting: $y = b + ax$
  - Example) Line fitting with scikit-learn (1/2)



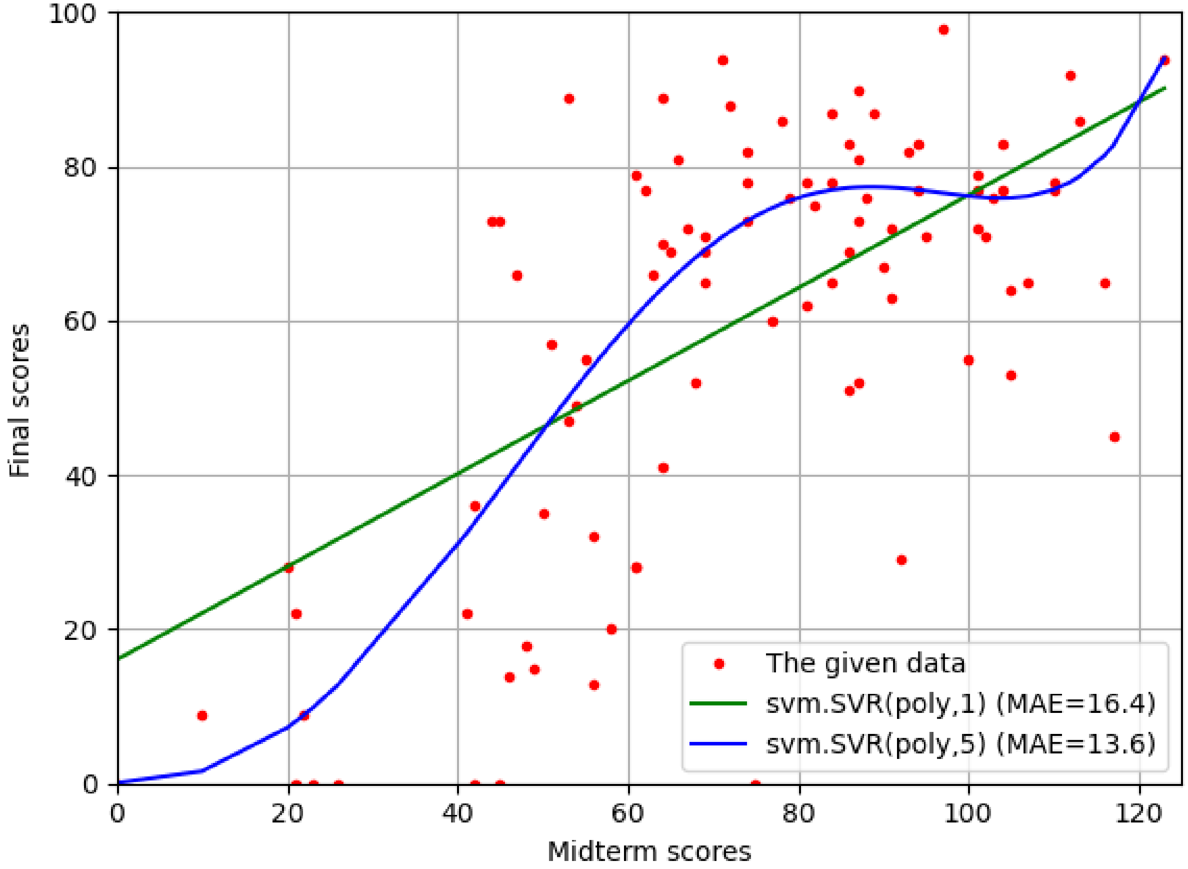
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import (linear_model, metrics)

true_line = lambda x: -2/3*x + 14/3
data_range = np.array([-4, 12])
data_num = 100
noise_std = 0.5

# Generate the true data
x = np.random.uniform(data_range[0], data_range[1], size=data_num)
y = true_line(x) # y = -2/3*x + 10/3

# Add Gaussian noise
xn = x + np.random.normal(scale=noise_std, size=x.shape)
yn = y + np.random.normal(scale=noise_std, size=y.shape)

# Train a model
model = linear_model.LinearRegression()
model.fit(xn.reshape(-1, 1), yn)
score = model.score(xn.reshape(-1, 1), yn)

# Plot the data and result
plt.title(f'Line: y={model.coef_[0]:.3f}*x + {model.intercept_:.3f} (score={score:.3f})')
plt.plot(data_range, true_line(data_range), 'r-', label='The true line')
plt.plot(xn, yn, 'b.', label='Noisy data')
plt.plot(data_range, model.coef_[0]*data_range + model.intercept_, 'g-', label='Estimate')
plt.xlim(data_range)
plt.legend()
plt.show()
```



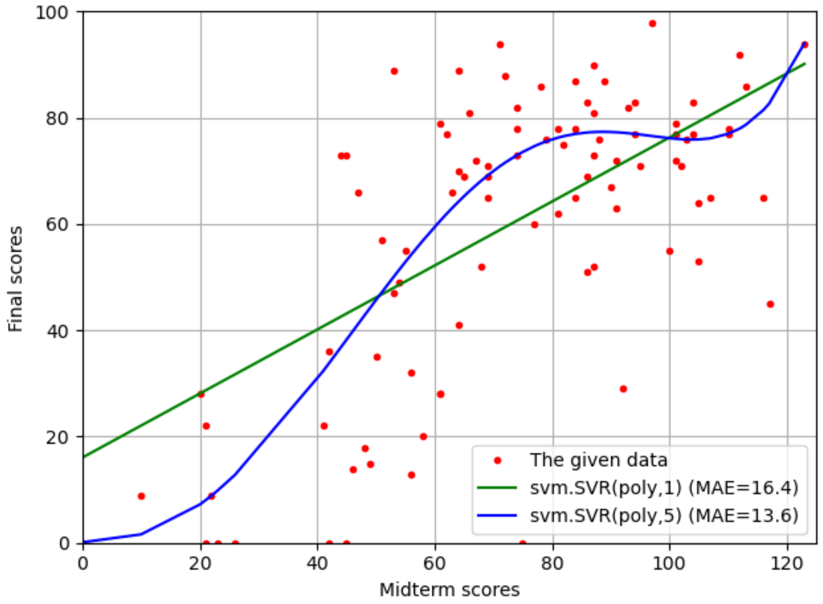
Regression

- **Support vector ~~machine~~ regression (SVR)** [scikit-learn] [API]
  - Example) Final exam score prediction with SVR (1/2)
    - The given data (file: `data/class_score_en.csv`)

```
# midterm (max 125), final (max 100)
113, 86
104, 83
110, 78
101, 79
101, 77
103, 76
71, 94
102, 71
88, 76
101, 72
. . .
```



```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm

midterm_range = np.array([0, 125])
final_range = np.array([0, 100])

# Load score data
class_kr = np.loadtxt('data/class_score_kr.csv', delimiter=',')
class_en = np.loadtxt('data/class_score_en.csv', delimiter=',')
data = np.vstack((class_kr, class_en))
x = data[:,0].reshape(-1, 1)
y = data[:,1]

# Instantiate regression models
models = [
    {'name': 'svm.SVR(poly,1)', 'obj': svm.SVR(kernel='poly', degree=1, coef0=1), 'color': 'g'},
    {'name': 'svm.SVR(poly,5)', 'obj': svm.SVR(kernel='poly', degree=5, coef0=1), 'color': 'b'},
]

# Prepare for plotting
plt.figure()
plt.plot(data[:,0], data[:,1], 'r.', label='The given data')
idx_sort = np.argsort(data[:,0])

for model in models:
    # Train a model
    model['obj'].fit(x, y)

    # Visualize regression results (the estimated line)
    y_pred = model['obj'].predict(x)
    model['mae'] = np.mean(np.abs(y_pred - y))
    plt.plot(x[idx_sort,0], y_pred[idx_sort], label=f'{model["name"]} (MAE={model["mae"]:.1f})', color=model['color'])
```



Note) Mean Absolute Error (MAE)

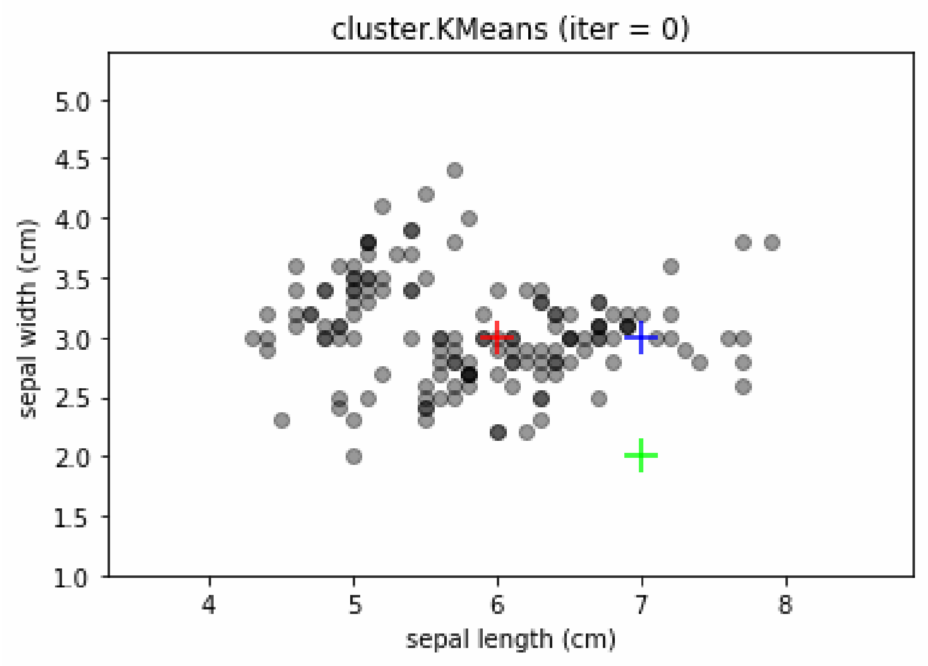
$$\mathrm{MAE}(Y, \hat{Y}) = \frac{1}{N}\sum_{i=1}^{N} |\hat{y}_i - y_i|$$

Clustering

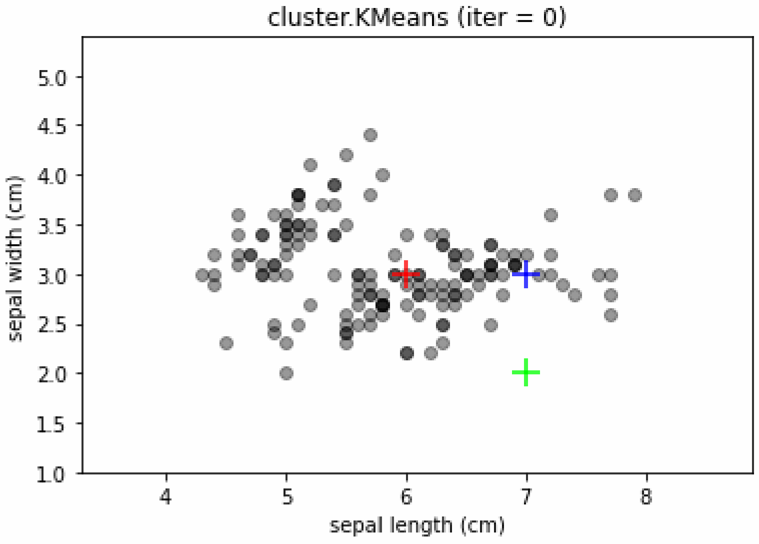
- **k-means clustering** [scikit-learn] [API]
  - k-means algorithm moves centroids to minimize the within-cluster sum of squares (WCSS; a.k.a. variance)
    - Initialization: Randomly select the initial centroids
    - Iterative update: Recalculate each centroid as the mean of its assigned data points

$$\text{The } j\text{-th centroid: } \mu_j^{k+1} = \frac{\sum_{i=1}^{n} x_i \, 1(c_i = j)}{\sum_{i=1}^{n} 1(c_i = j)} \quad \text{where} \quad c_i = \underset{j}{\operatorname{argmin}} \left\| x_i - \mu_j^k \right\|^2$$

  - Example) k-means clustering with Iris flower dataset



```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn import (datasets, cluster)
from matplotlib.colors import ListedColormap

# Load a dataset partially
iris = datasets.load_iris()
iris.data = iris.data[:,0:2]                 # Try [:,2:4]
iris.feature_names = iris.feature_names[0:2] # Try [:,2:4]
iris.color = np.array([(1, 0, 0), (0, 1, 0), (0, 0, 1)])

# Train a model
model = cluster.KMeans(n_clusters=3)
model.fit(iris.data)

# Visualize training results (decision boundaries)
x_min, x_max = iris.data[:, 0].min() - 1, iris.data[:, 0].max() + 1
y_min, y_max = iris.data[:, 1].min() - 1, iris.data[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01), np.arange(y_min, y_max, 0.01))
xy = np.vstack((xx.flatten(), yy.flatten())).T
zz = model.predict(xy)
plt.contourf(xx, yy, zz.reshape(xx.shape), cmap=ListedColormap(iris.color), alpha=0.2)

# Visualize testing results
plt.title('cluster.KMeans')
plt.scatter(iris.data[:,0], iris.data[:,1], c=iris.color[iris.target])
plt.xlabel(iris.feature_names[0])
plt.ylabel(iris.feature_names[1])

# Visualize training results (mean values)
for c in range(model.n_clusters):
    plt.scatter(*model.cluster_centers_[c], marker='+', s=200, color='k')
plt.show()
```



Machine Learning FAQ

- **Q) How to select a good ML algorithm or model?**
- **A) Mostly deep learning (if enough data are available), but not always**

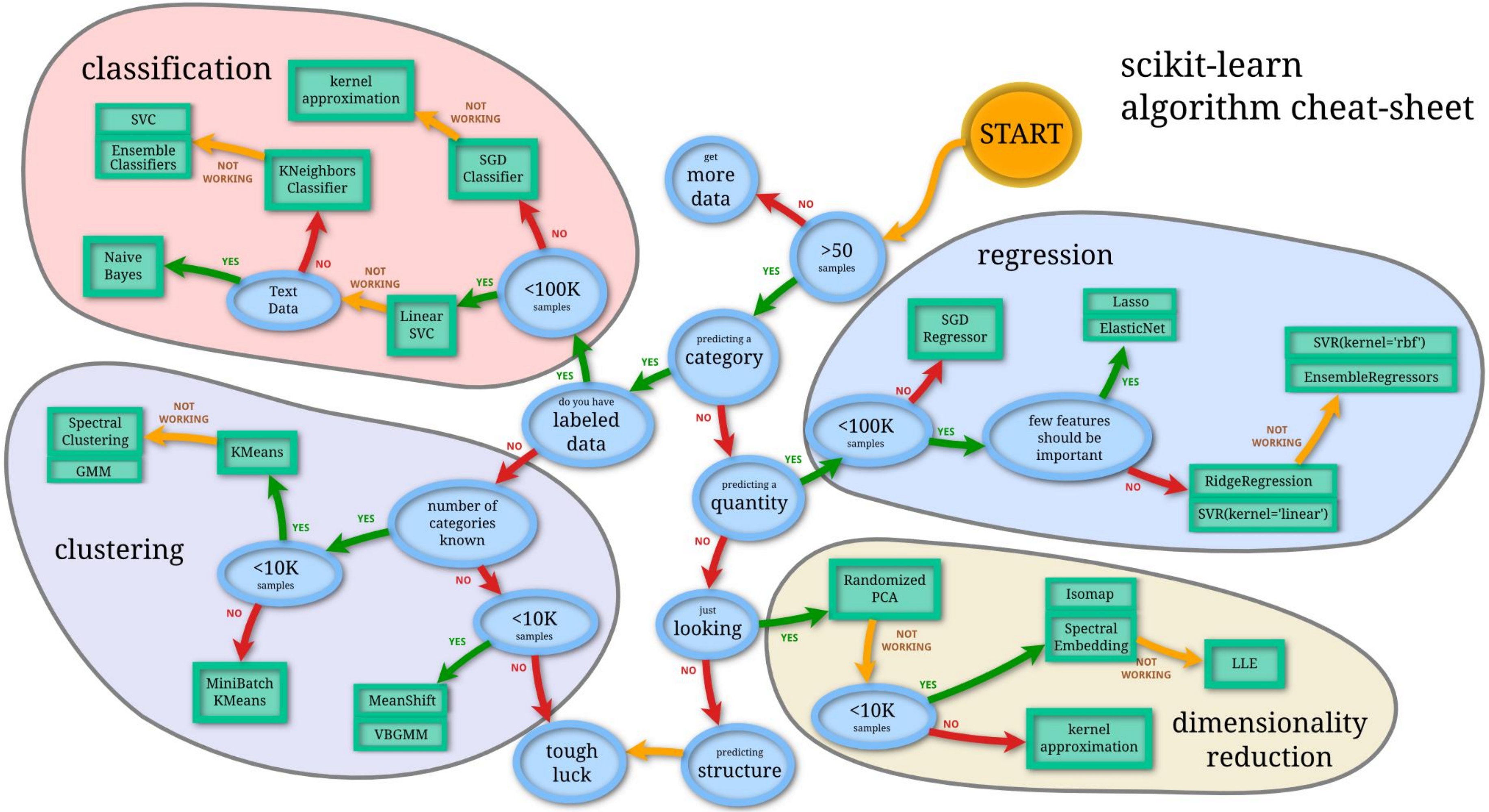
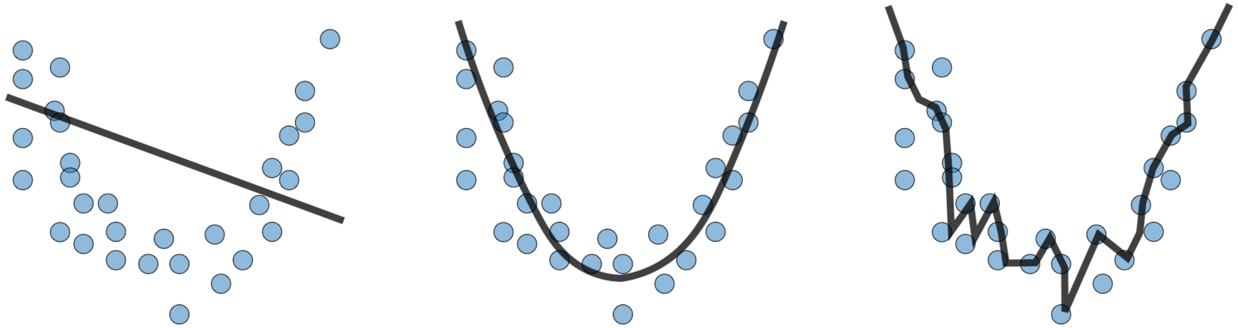


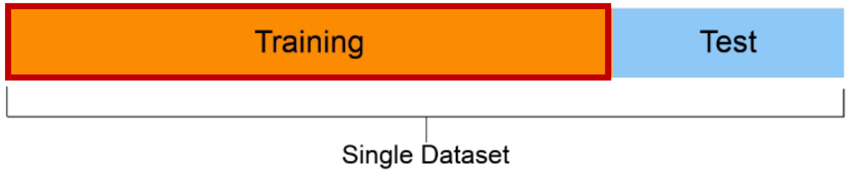
Image: scikit-learn (Note: The online figure has hyperlinks for further information.)

Image: Amidi's ML cheatsheet and Wikipedia

Machine Learning FAQ

- **Q) How to achieve a good result (good training)?**
- **A) More trials (more time/computing power), but your intuition and experience are important.**
  - Good hyperparameter values?
    - Grid search (~ try all combinations, often on a log-scale) or your intuition and experience
  - Enough model complexity? Enough training?
    - Observing test results: Underfitting or overfitting

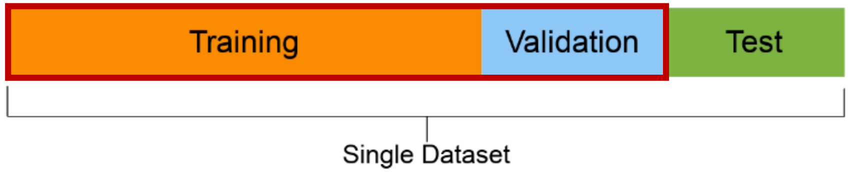




Training | Test

Single Dataset

- Training, validation, and test data separation
  - Note) **Validation data**: Test data for intermediate trained models



Training | Validation | Test

Single Dataset

- Enough data?
  - More data are always good if possible.
  - Please think about the dimension of data and the number of model parameters.

Underfitting vs. Overfitting

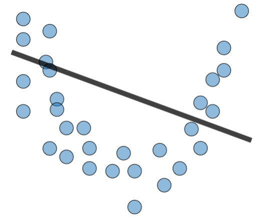
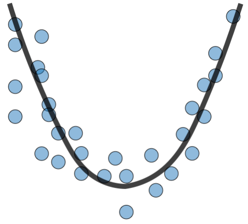
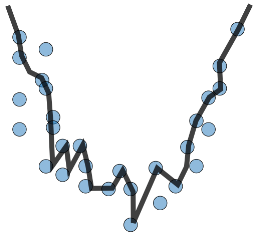
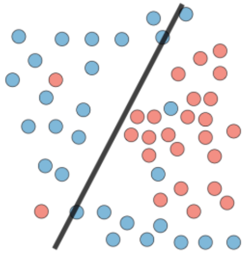
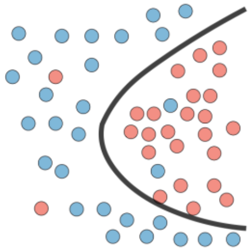
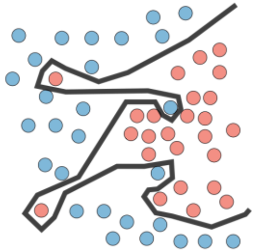
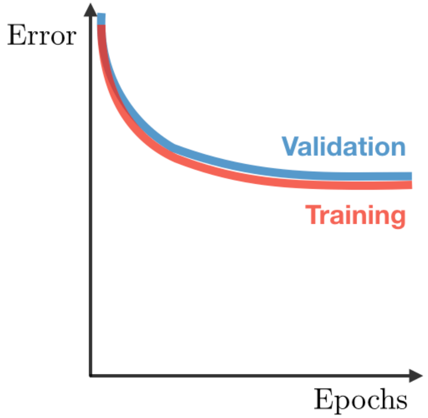
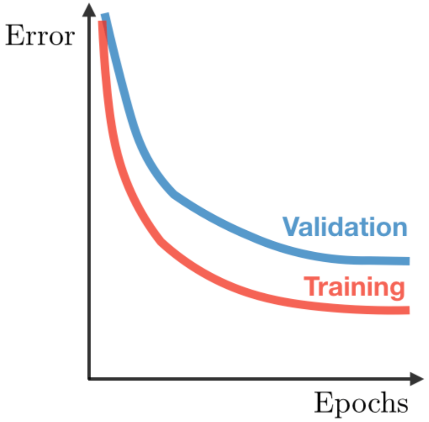
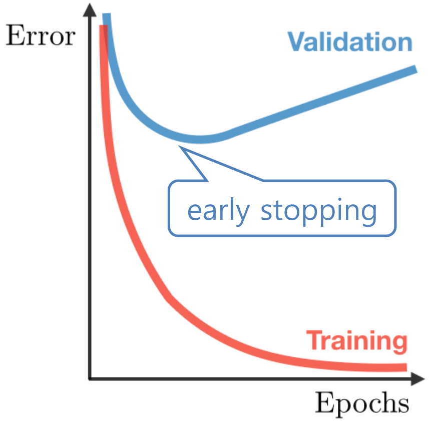
	Underfitting	Just fit	Overfitting
Symptoms	▪ High training error ▪ **Training error close to test error** ▪ **High bias**	▪ Training error slightly lower than test error	▪ Very low training error ▪ **Training error much lower than test error** ▪ **High variance**
Regression Illustration			
Classification Illustration			
Learning Curve			
Possible Remedies	▪ Increase model complexity ▪ Add more features ▪ Train longer		▪ Simplify your model ▪ Add regularization ▪ Get more data

Image: [Amidi's ML cheatsheet]

Underfitting vs. Overfitting

- Bias-variance tradeoff

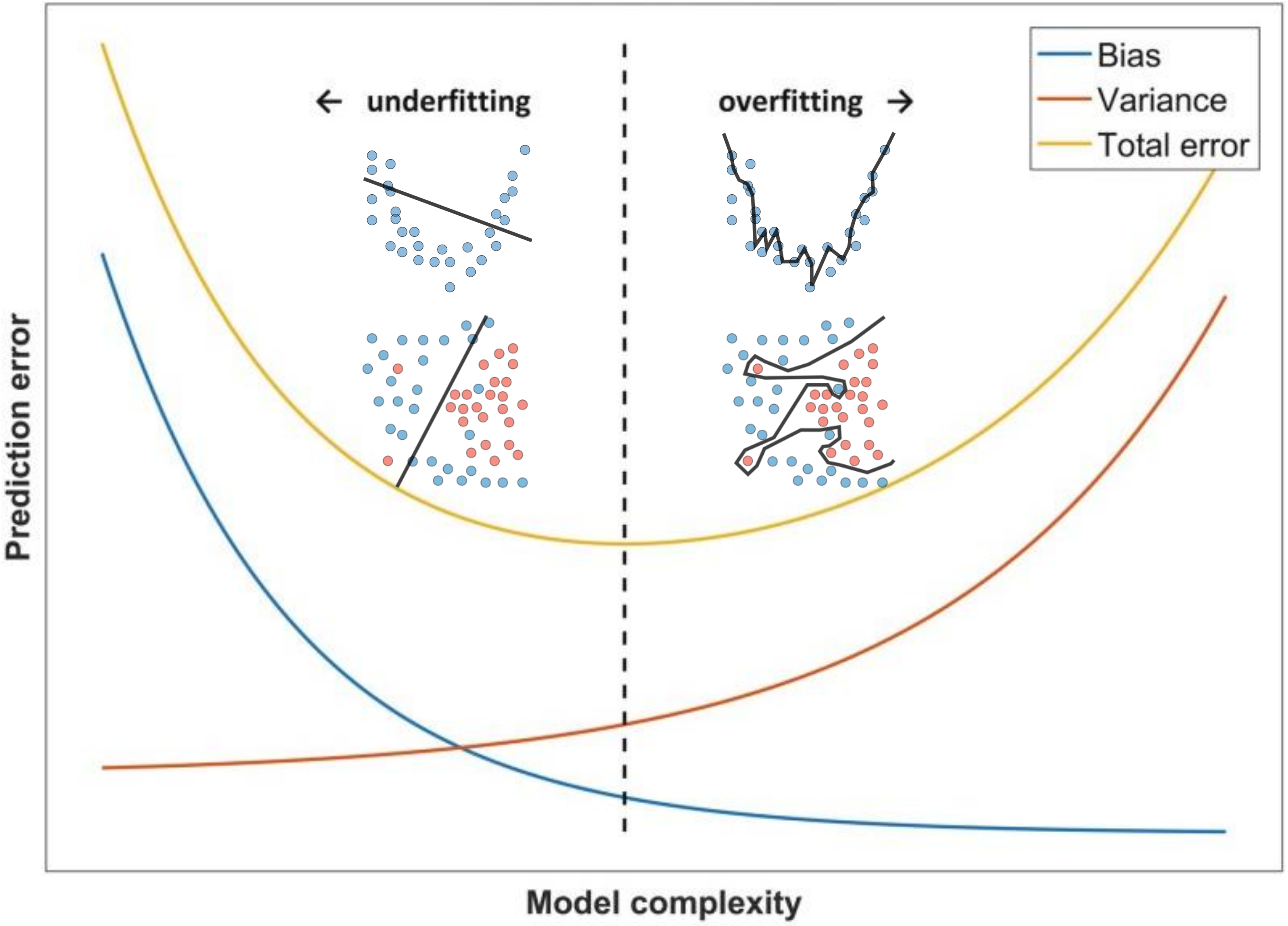


Image: NCBI and Amidi's ML cheatsheet

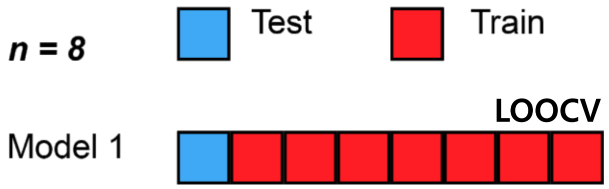
Training and Test Data Separation

- **Holdout method**: A simple training and test data separation
  - Holdout method separates training and test data in the given ratio (e.g. 7:3 or 9:1) and does not shuffle them.
  - Holdout method has danger of overfitting to the fixed test data.
  - Example) Holdout validation with scikit-learn

```python
from sklearn import (datasets, svm, model_selection, metrics)

# Load a dataset
iris = datasets.load_iris()
iris.data = iris.data[:,0:2]
iris.feature_names = iris.feature_names[0:2]
X_train, X_test, y_train, y_test = model_selection.train_test_split(iris.data, iris.target, test_size=0.3)

# Train a model
model = svm.SVC()
model.fit(X_train, y_train)

# Evaluate the model
acc_train = metrics.balanced_accuracy_score(y_train, model.predict(X_train))
acc_test = metrics.balanced_accuracy_score(y_test, model.predict(X_test))
print(f'* Accuracy @ training data: {acc_train:.3f}')
print(f'* Accuracy @ test data: {acc_test:.3f}')
```

Training and Test Data Separation

- **Cross-validation** (a.k.a. rotation estimation, out-of-sample testing)
  - Cross-validation is a model validation technique to assess how the trained model will generalize to an independent data set (e.g. unseen dataset from the real world).
    - It *was* common for academic researches (with *small-size* datasets).
    - It utilizes the whole data more effectively for training and testing a model (compared to holdout method).
  - Exhaustive cross-validation → Usually not feasible
    - **Leave-p-out cross-validation (LpOCV)**
      - Training $n - p$ data and test p data for *all possible combinations*
      - C_p^n times of trainings (e.g. $C_{30}^{100} \approx 3 \times 10^{25}$)
    - **Leave-one-out cross-validation (LOOCV)**
      - Training $n - 1$ data and test a single datum for n *possible combinations*
      - n times of trainings ($p = 1$)
  - Non-exhaustive cross-validation
    - **k-fold cross-validation**
      - LOOCV on k equal sized subsamples: k times of trainings



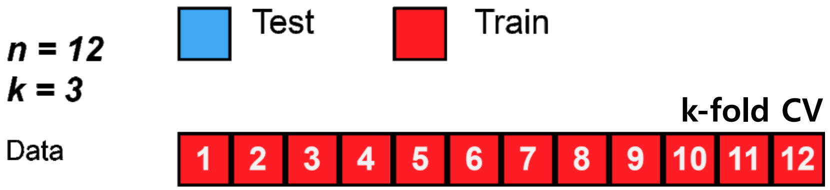


Image: Wikipedia

Machine Learning FAQ

- **Cross-validation**
  - Example) k-fold cross-validation with scikit-learn

```
import numpy as np
from sklearn import (datasets, svm, model_selection)

# Load a dataset
iris = datasets.load_iris()
iris.data = iris.data[:,0:2]
iris.feature_names = iris.feature_names[0:2]

# Train a model
model = svm.SVC()
cv_results = model_selection.cross_validate(model, iris.data, iris.target, cv=5, return_train_score=True)

# Evaluate the model
acc_train = np.mean(cv_results['train_score'])
acc_test = np.mean(cv_results['test_score'])
print(f'* Accuracy @ training data: {acc_train:.3f}')
print(f'* Accuracy @ test data: {acc_test:.3f}')
```

Summary

- **Machine Learning**
  - In short, data-driven approaches, inductive approaches
  - Problem formulation: Classification, regression, clustering, …
- **scikit-learn**
  - Usage (3 steps): Instantiation → training (`fit`) → inference/testing (`predict`)
- **Classification**
  - Performance measures
    - Accuracy
    - Confusion matrix (balanced accuracy, precision/recall, F1-measure)
    - ROC curve
  - **Support vector machine (SVM)** [scikit-learn] [API]
  - **Decision tree** [scikit-learn] [API]
  - **Naïve Bayes classifiers** [scikit-learn] [API]
  - Example) Iris flower data classification

Summary

- **Regression**
  - **Linear regression** [scikit-learn] [API]
  - Example) Line fitting with scikit-learn
  - **Support vector regression (SVR)** [scikit-learn] [API]
  - Example) Final exam score prediction with SVR
- **Clustering**
  - **k-means clustering** [scikit-learn] [API]
  - Example) Iris flower data clustering
- **Machine Learning FAQ**
  - Good algorithm or model?
    - Mostly deep learning, but not always
  - Enough model complexity? Enough training?
    - Data separation: Train, validation, and test data
      - Holdout method ~~vs. Cross-validation (not used with *large-scale* data)~~
    - Learning curve
      - Underfitting vs. Overfitting (Early stopping)