# Schedule

## Block 1 (09:00 - 10:30 am)

(1) Introduction to machine learning

**(2) The scikit-learn API**

**30 min break (10:30 - 11:00 am)**

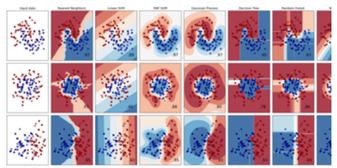
# The "main" machine learning library for Python


**http://scikit-learn.org**

### Classification

Identifying which category an object belongs to.

**Applications:** Spam detection, image recognition.
**Algorithms:** Gradient boosting, nearest neighbors, random forest, logistic regression, and more...

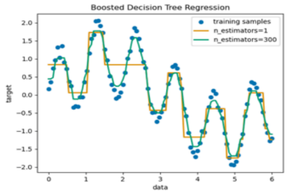


Examples

### Regression

Predicting a continuous-valued attribute associated with an object.

**Applications:** Drug response, Stock prices.
**Algorithms:** Gradient boosting, nearest neighbors, random forest, ridge, and more...

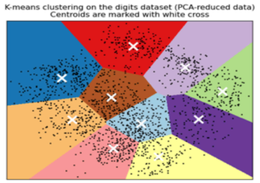



Examples

### Clustering

Automatic grouping of similar objects into sets.

**Applications:** Customer segmentation, Grouping experiment outcomes
**Algorithms:** k-Means, HDBSCAN, hierarchical clustering, and more...

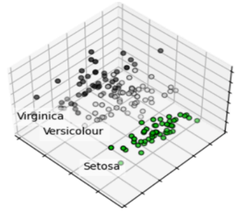



Examples

### Dimensionality reduction

Reducing the number of random variables to consider.

**Applications:** Visualization, Increased efficiency
**Algorithms:** PCA, feature selection, non-negative matrix factorization, and more...

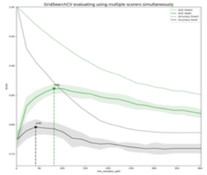



Examples

### Model selection

Comparing, validating and choosing parameters and models.

**Applications:** Improved accuracy via parameter tuning
**Algorithms:** grid search, cross validation, metrics, and more...

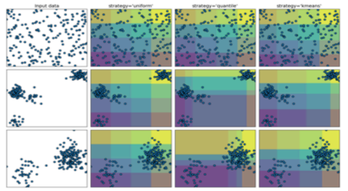


Examples

### Preprocessing

Feature extraction and normalization.

**Applications:** Transforming input data such as text for use with machine learning algorithms.
**Algorithms:** preprocessing, feature extraction, and more...



Examples

# A quick example …

```
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier

X, y = load_iris(return_X_y=True)
est = KNeighborsClassifier()

est.fit(X, y)
```

▼ KNeighborsClassifier

KNeighborsClassifier()

```
est.predict(X)
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 2, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

```
est.score(X, y)
```

```
0.9666666666666667
```

# The scikit-learn estimator API follows an object-oriented paradigm

Class name

```
class SupervisedEstimator(...):

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

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

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

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

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

# The scikit-learn estimator API follows an object-oriented paradigm

Methods in all scikit-learn classifiers

```
class SupervisedEstimator(...):

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

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

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

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

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

# The scikit-learn estimator API follows an object-oriented paradigm

Constructor

```
class SupervisedEstimator(...):

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

```
est = KNeighborsClassifier(n_neighbors=5)
```

```
est.n_neighbors
```

```
5
```

# The scikit-learn estimator API follows an object-oriented paradigm

```
class SupervisedEstimator(...):

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

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

```
est = KNeighborsClassifier(n_neighbors=5)
```

```
est.fit(X, y)
```

▼ KNeighborsClassifier

KNeighborsClassifier()

```
est.classes_
```

```
array([0, 1, 2])
```

# The scikit-learn estimator API follows an object-oriented paradigm

```
class SupervisedEstimator(...):

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

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

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

```
est.predict(X)
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 2, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

# The scikit-learn estimator API follows an object-oriented paradigm

```
class SupervisedEstimator(...):

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

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

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

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

```
est.score(X, y)
```

```
0.9666666666666667
```

# The scikit-learn estimator API follows an object-oriented paradigm

```
class SupervisedEstimator(...):

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

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

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

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

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

```
import numpy as np

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

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

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

    def score(self, X, y):
        y_pred = self.predict(X)
        return np.mean(y_pred == y)

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

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

        sorted_dist_idx_pairs = sorted(dist_idx_pairs)

        return sorted_dist_idx_pairs
```

```
clf = KNNClassifier(n_neighbors=5)
clf.fit(X, y)
```

```
clf.predict(X)
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 2, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

```
clf.score(X, y)
```

```
0.9666666666666667
```

## 2D Decision region example

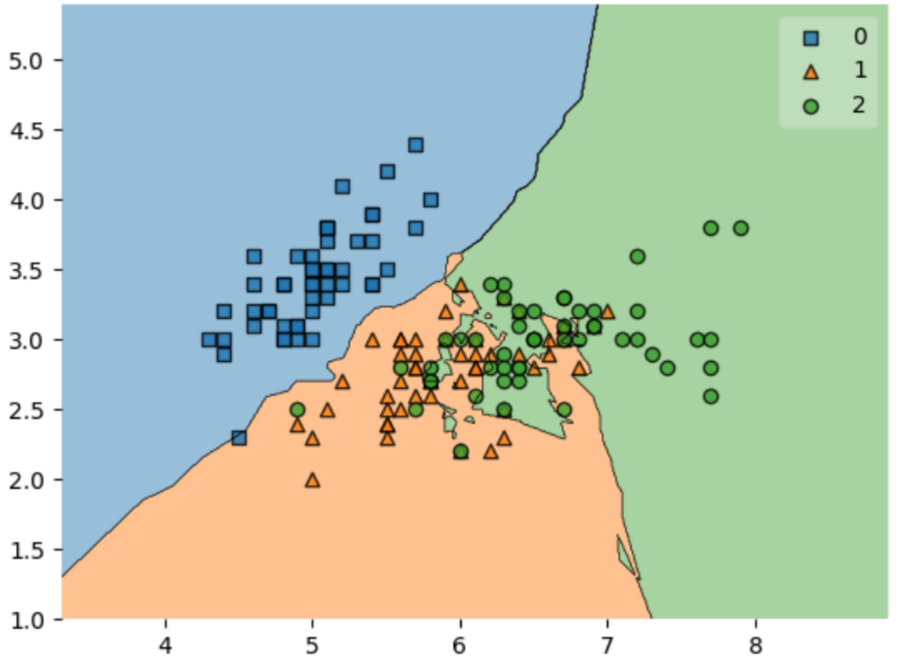
```
#pip install mlxtend
```

```
from mlxtend.plotting import plot_decision_regions
```

```
est = KNeighborsClassifier(n_neighbors=5)
est.fit(X[:, :2], y) # only use 2 features

plot_decision_regions(X[:, :2], y, est)
```

```
<Axes: >
```



# Exercise:

Classification with train/val/test split

🎮 https://github.com/rasbt/posit2023-python-ml