CAPAIAN PEMBELAJARAN PRAKTIKUM

1. Menjelaskan konsep-konsep dari pembelajaran mesin untuk kemudian bisa diterapkan pada berbagai permasalahan– (C2);
2. Mengimplementasikan algoritma dan/atau metode di pembelajaran mesin sehingga bisa mendapatkan pemecahan masalah dan model yang sesuai – (C3)
3. Menghasilkan suatu model terbaik dari permasalahan dan terus melakukan perbaikan jika terdapat perkembangan di bidang rekayasa cerdas – (C6)

MINGGU 6
Instance Based Learning

DESKRIPSI TEMA

Mahasiswa mempelajari berbagai varian dari algoritma tetangga terdekat dan menerapkannya dengan Bahasa pemrograman python.

CAPAIAN PEMBELAJARAN MINGGUAN (SUB-CAPAIAN PEMBELAJARAN)

Mahasiswa dapat menerapkan algoritma dengan tetangga terdekat sehingga bisa menghasilkan model terbaik dengan bahasa pemrograman python – SCPMK-06

PERALATAN YANG DIGUNAKAN

Anaconda Python 3

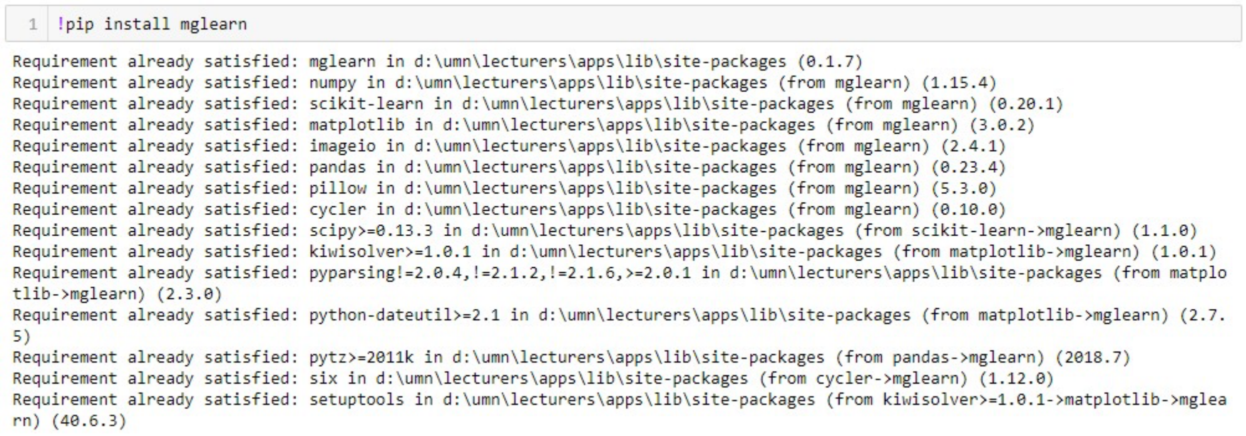
Laptop atau Personal Computer

LANGKAH-LANGKAH PRAKTIKUM

Sample dataset

We will use several datasets to illustrate the different algorithms. Some of the datasets will be small and synthetic (meaning made-up), designed to highlight particular aspects of the algorithms. Other datasets will be large, real-world examples.

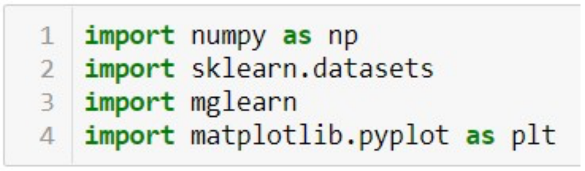
1.Install mglearn dataset from jupyter notebook.



```
!pip install mglearn
```

```
Requirement already satisfied: mglearn in d:\umn\lecturers\apps\lib\site-packages (0.1.7)
Requirement already satisfied: numpy in d:\umn\lecturers\apps\lib\site-packages (from mglearn) (1.15.4)
Requirement already satisfied: scikit-learn in d:\umn\lecturers\apps\lib\site-packages (from mglearn) (0.20.1)
Requirement already satisfied: matplotlib in d:\umn\lecturers\apps\lib\site-packages (from mglearn) (3.0.2)
Requirement already satisfied: imageio in d:\umn\lecturers\apps\lib\site-packages (from mglearn) (2.4.1)
Requirement already satisfied: pandas in d:\umn\lecturers\apps\lib\site-packages (from mglearn) (0.23.4)
Requirement already satisfied: pillow in d:\umn\lecturers\apps\lib\site-packages (from mglearn) (5.3.0)
Requirement already satisfied: cycler in d:\umn\lecturers\apps\lib\site-packages (from mglearn) (0.10.0)
Requirement already satisfied: scipy>=0.13.3 in d:\umn\lecturers\apps\lib\site-packages (from scikit-learn->mglearn) (1.1.0)
Requirement already satisfied: kiwisolver>=1.0.1 in d:\umn\lecturers\apps\lib\site-packages (from matplotlib->mglearn) (1.0.1)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in d:\umn\lecturers\apps\lib\site-packages (from matplo
tlib->mglearn) (2.3.0)
Requirement already satisfied: python-dateutil>=2.1 in d:\umn\lecturers\apps\lib\site-packages (from matplotlib->mglearn) (2.7.
5)
Requirement already satisfied: pytz>=2011k in d:\umn\lecturers\apps\lib\site-packages (from pandas->mglearn) (2018.7)
Requirement already satisfied: six in d:\umn\lecturers\apps\lib\site-packages (from cycler->mglearn) (1.12.0)
Requirement already satisfied: setuptools in d:\umn\lecturers\apps\lib\site-packages (from kiwisolver>=1.0.1->matplotlib->mglea
rn) (40.6.3)
```

2. Import libraries

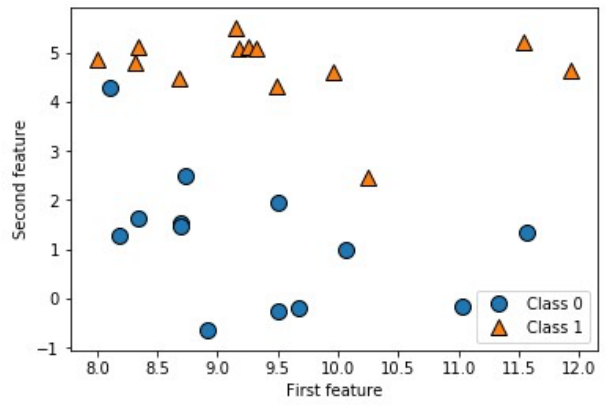


```
import numpy as np
import sklearn.datasets
import mglearn
import matplotlib.pyplot as plt
```

3. An example of two-class classification dataset is forge dataset, which has two features.

```python
# generate dataset
X, y = mglearn.datasets.make_forge()
# plot dataset
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
plt.legend(["Class 0", "Class 1"], loc=4)
plt.xlabel("First feature")
plt.ylabel("Second feature")
print("X.shape: {}".format(X.shape))
```

```
X.shape: (26, 2)
```



The plot has the first feature on the x-axis and the second feature on the y-axis. As is always the case in scatter plots, each data point is represented as one dot. The color and shape of the dot indicates its class

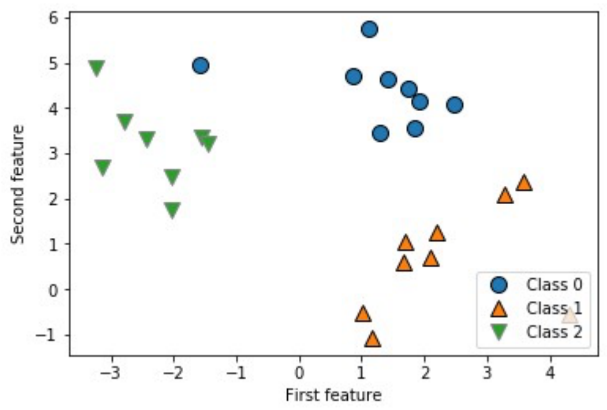
4. An example of a synthetic multiclass classification dataset is the make_blobs dataset, which has two features. The following code creates a scatter plot visualizing all of the data points in this dataset.

```python
from sklearn.datasets.samples_generator import make_blobs
X, y = make_blobs(n_samples=26, n_features=2, random_state=0)

# plot dataset
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
plt.legend(["Class 0", "Class 1", "Class 2"], loc=4)
plt.xlabel("First feature")
plt.ylabel("Second feature")

print("Target:", y)
print("X.shape: {}".format(X.shape))
```

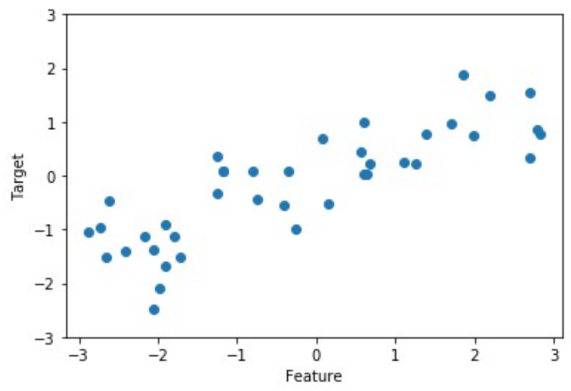
```
Target: [0 2 0 0 2 1 0 1 1 1 2 2 2 0 0 1 1 2 0 2 0 1 2 1 0 1]
X.shape: (26, 2)
```



5. To illustrate regression algorithms, we will use the synthetic wave dataset. The wave dataset has a single input feature and a continuous target variable (or response) that we want to model.

```
X, y = mglearn.datasets.make_wave(n_samples=40)
plt.plot(X, y, 'o')
plt.ylim(-3, 3)
plt.xlabel("Feature")
plt.ylabel("Target")
```

```
Text(0, 0.5, 'Target')
```



The plot created above shows the single feature on the x-axis and the regression target (the output) on the y-axis.

6. Wisconsin Breast Cancer Data
   Wisconsin Breast Cancer dataset (cancer, for short), which records clinical measurements of breast cancer tumors. Each tumor is labeled as “benign” (for harmless tumors) or “malignant” (for cancerous tumors), and the task is to learn to predict whether a tumor is malignant based on the measurements of the tissue.

```
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
print("cancer.keys(): \n{}".format(cancer.keys()))
```

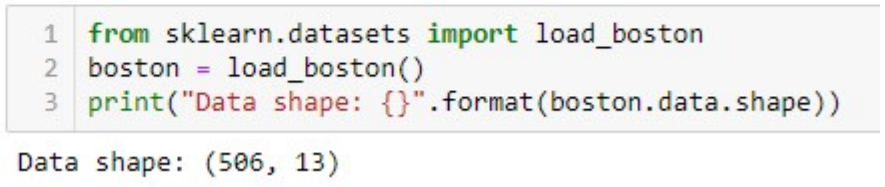
```
cancer.keys():
dict_keys(['data', 'target', 'target_names', 'DESCR', 'feature_names', 'filename'])
```

7. The dataset consists of 569 data points with 30 features each. Of these 569 data points, 212 are labeled as malignant and 357 as benign.
8. To get a description of the semantic meaning of each feature, we can have a look at the feature_names attribute.

```
print("Feature names:\n{}".format(cancer.feature_names))
```

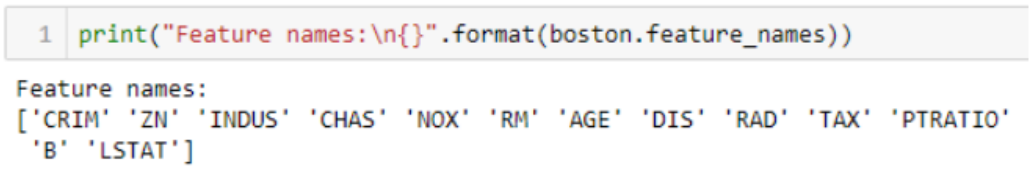
```
Feature names:
['mean radius' 'mean texture' 'mean perimeter' 'mean area'
 'mean smoothness' 'mean compactness' 'mean concavity'
 'mean concave points' 'mean symmetry' 'mean fractal dimension'
 'radius error' 'texture error' 'perimeter error' 'area error'
 'smoothness error' 'compactness error' 'concavity error'
 'concave points error' 'symmetry error' 'fractal dimension error'
 'worst radius' 'worst texture' 'worst perimeter' 'worst area'
 'worst smoothness' 'worst compactness' 'worst concavity'
 'worst concave points' 'worst symmetry' 'worst fractal dimension']
```

9. We will also be using a real-world regression dataset, the Boston Housing dataset. The task associated with this dataset is to predict the median value of homes in several Boston neighborhoods in the 1970s, using information such as crime rate, proximity to the Charles River, highway accessibility, and so on.



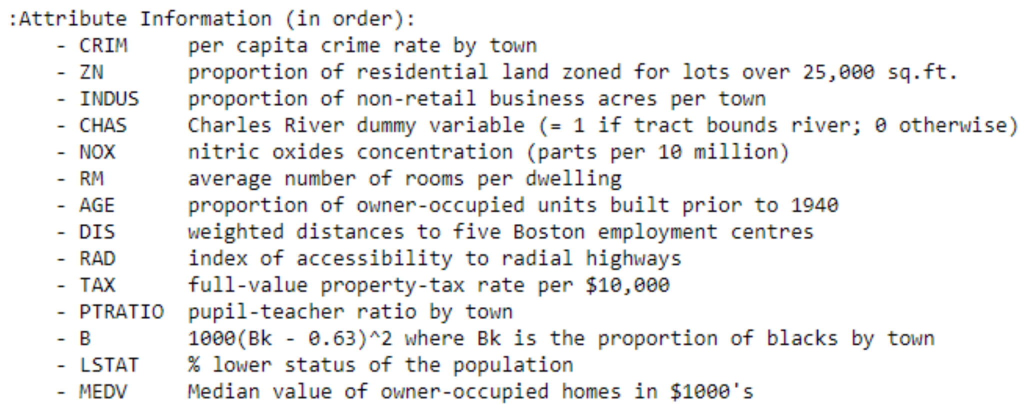
```python
from sklearn.datasets import load_boston
boston = load_boston()
print("Data shape: {}".format(boston.data.shape))
```

```
Data shape: (506, 13)
```



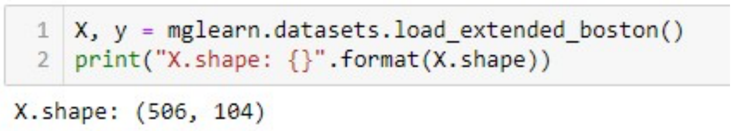
```python
print("Feature names:\n{}".format(boston.feature_names))
```

```
Feature names:
['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO'
 'B' 'LSTAT']
```



```
:Attribute Information (in order):
    - CRIM     per capita crime rate by town
    - ZN       proportion of residential land zoned for lots over 25,000 sq.ft.
    - INDUS    proportion of non-retail business acres per town
    - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
    - NOX      nitric oxides concentration (parts per 10 million)
    - RM       average number of rooms per dwelling
    - AGE      proportion of owner-occupied units built prior to 1940
    - DIS      weighted distances to five Boston employment centres
    - RAD      index of accessibility to radial highways
    - TAX      full-value property-tax rate per $10,000
    - PTRATIO  pupil-teacher ratio by town
    - B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
    - LSTAT    % lower status of the population
    - MEDV     Median value of owner-occupied homes in $1000's
```

10. We will not only consider crime rate and highway accessibility as features, but also the product of crime rate and highway accessibility. This derived dataset can be loaded using the load_extended_boston function:



```python
X, y = mglearn.datasets.load_extended_boston()
print("X.shape: {}".format(X.shape))
```

```
X.shape: (506, 104)
```

The resulting 104 features are the 13 original features together with the 91 possible combinations of two features within those 13.

k-Nearest Neighbor

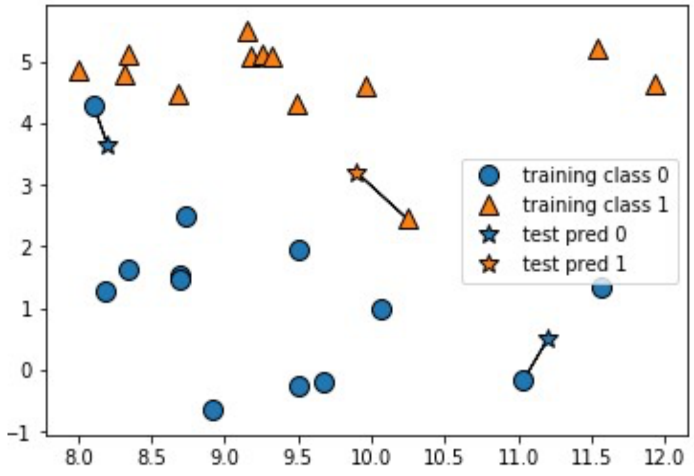
The k-NN algorithm is arguably the simplest machine learning algorithm. Building the model consists only of storing the training dataset. To make a prediction for a new data point, the algorithm finds the closest data points in the training dataset—its “nearest neighbors.”

k-Neighbors Classification

In its simplest version, the k-NN algorithm only considers exactly one nearest neighbor, which is the closest training data point to the point we want to make a prediction for.

11. k-NN for the case of classification on the forge dataset.

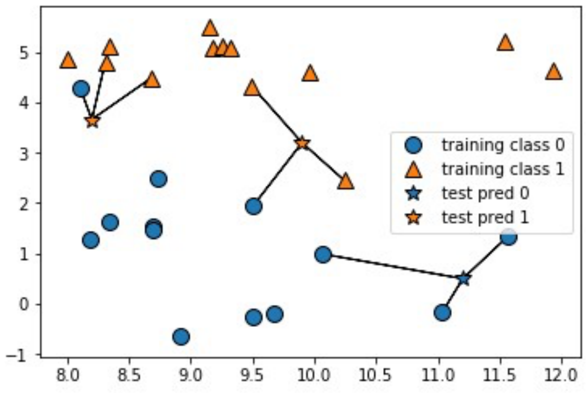
```
mglearn.plots.plot_knn_classification(n_neighbors=1)
```



Here, we added three new data points, shown as stars. For each of them, we marked the closest point in the training set. The prediction of the one-nearest-neighbor algorithm is the label of that point (shown by the color of the cross).

12. Instead of considering only the closest neighbor, we can also consider an arbitrary number, k, of neighbors. This is where the name of the k-nearest neighbors algorithm comes from. Try to change n_neighbors = 3.

```
mglearn.plots.plot_knn_classification(n_neighbors=3)
```



13. Apply the k-nearest neighbors algorithm using scikit-learn. First, we split our data into a training and a test set so we can evaluate generalization performance.

```
from sklearn.model_selection import train_test_split
X, y = mglearn.datasets.make_forge()

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
```

14. Next, we import and instantiate the class. This is when we can set parameters, like the number of neighbors to use. Here, we set it to 3.

```
from sklearn.neighbors import KNeighborsClassifier
clf = KNeighborsClassifier(n_neighbors=3)
```

Now, we fit the classifier using the training set. For KNeighborsClassifier this means storing the dataset, so we can compute neighbors during prediction.

```
clf.fit(X_train, y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=3, p=2,
           weights='uniform')
```

15. To make predictions on the test data, we call the predict method. For each data point in the test set, this computes its nearest neighbors in the training set and finds the most common class among these.

```python
print("Test set: \n{} \n{}\n".format(X_test, y_test))
print("Test set predictions: {}".format(clf.predict(X_test)))
```

```
Test set:
[[11.54155807  5.21116083]
 [10.06393839  0.99078055]
 [ 9.49123469  4.33224792]
 [ 8.18378052  1.29564214]
 [ 8.30988863  4.80623966]
 [10.24028948  2.45544401]
 [ 8.34468785  1.63824349]]
[1 0 1 0 1 1 0]

Test set predictions: [1 0 1 0 1 0 0]
```

16. To evaluate how well our model generalizes, we can call the score method with the test data together with the test labels.

```python
print("Test set accuracy: {:.2f}".format(clf.score(X_test, y_test)))
```

```
Test set accuracy: 0.86
```

<u>Analyzing KNeighborsClassifier</u>

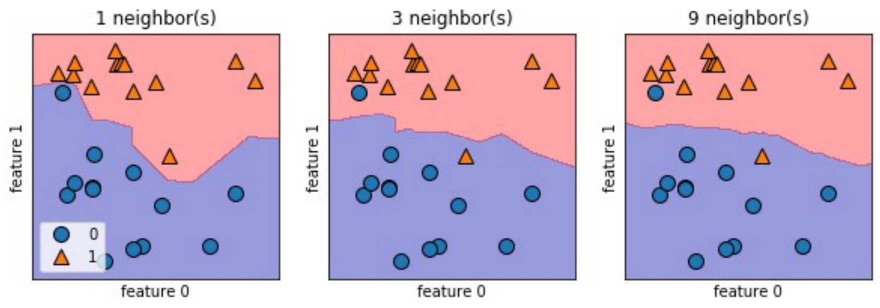
This lets us view the decision boundary, which is the divide between where the algorithm assigns class 0 versus where it assigns class 1.

17. The following code produces the visualizations of the decision boundaries for one, three, and nine neighbors.

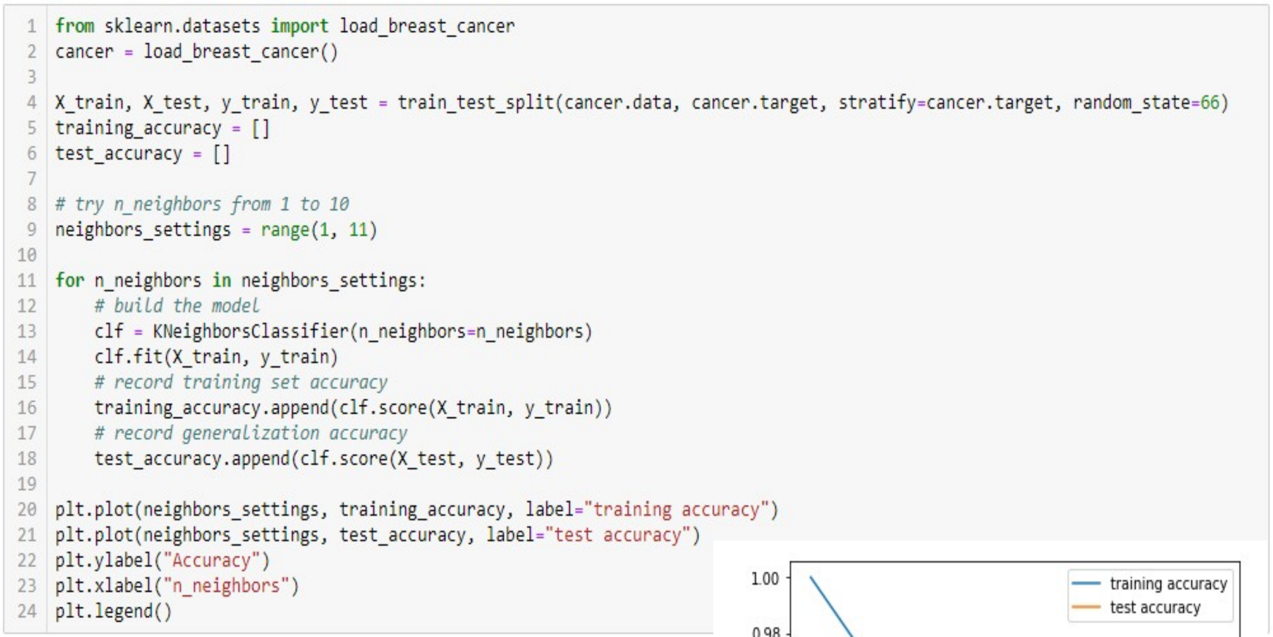
```python
fig, axes = plt.subplots(1, 3, figsize=(10, 3))

for n_neighbors, ax in zip([1, 3, 9], axes):
    # the fit method returns the object self, so we can instantiate
    # and fit in one line
    clf = KNeighborsClassifier(n_neighbors=n_neighbors).fit(X, y)
    mglearn.plots.plot_2d_separator(clf, X, fill=True, eps=0.5, ax=ax, alpha=.4)
    mglearn.discrete_scatter(X[:, 0], X[:, 1], y, ax=ax)
    ax.set_title("{} neighbor(s)".format(n_neighbors))
    ax.set_xlabel("feature 0")
    ax.set_ylabel("feature 1")
axes[0].legend(loc=3)
```

```
<matplotlib.legend.Legend at 0x182f3055a20>
```



18. Let's investigate whether we can confirm the connection between model complexity and generalization that we discussed earlier. We will do this on the real-world Breast Cancer dataset.



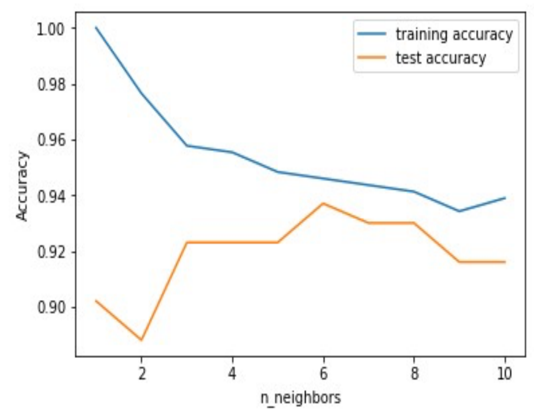
```python
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()

X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target, stratify=cancer.target, random_state=66)
training_accuracy = []
test_accuracy = []

# try n_neighbors from 1 to 10
neighbors_settings = range(1, 11)

for n_neighbors in neighbors_settings:
    # build the model
    clf = KNeighborsClassifier(n_neighbors=n_neighbors)
    clf.fit(X_train, y_train)
    # record training set accuracy
    training_accuracy.append(clf.score(X_train, y_train))
    # record generalization accuracy
    test_accuracy.append(clf.score(X_test, y_test))

plt.plot(neighbors_settings, training_accuracy, label="training accuracy")
plt.plot(neighbors_settings, test_accuracy, label="test accuracy")
plt.ylabel("Accuracy")
plt.xlabel("n_neighbors")
plt.legend()
```

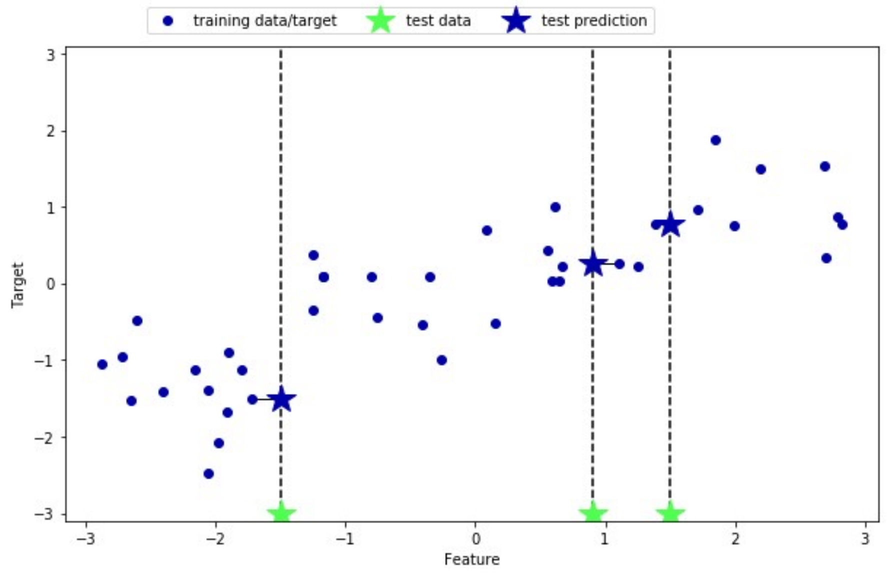


k-Neighbors Regression

There is also a regression variant of the k-nearest neighbors algorithm.

19. Let's start by using the single nearest neighbor, this time using the wave dataset.

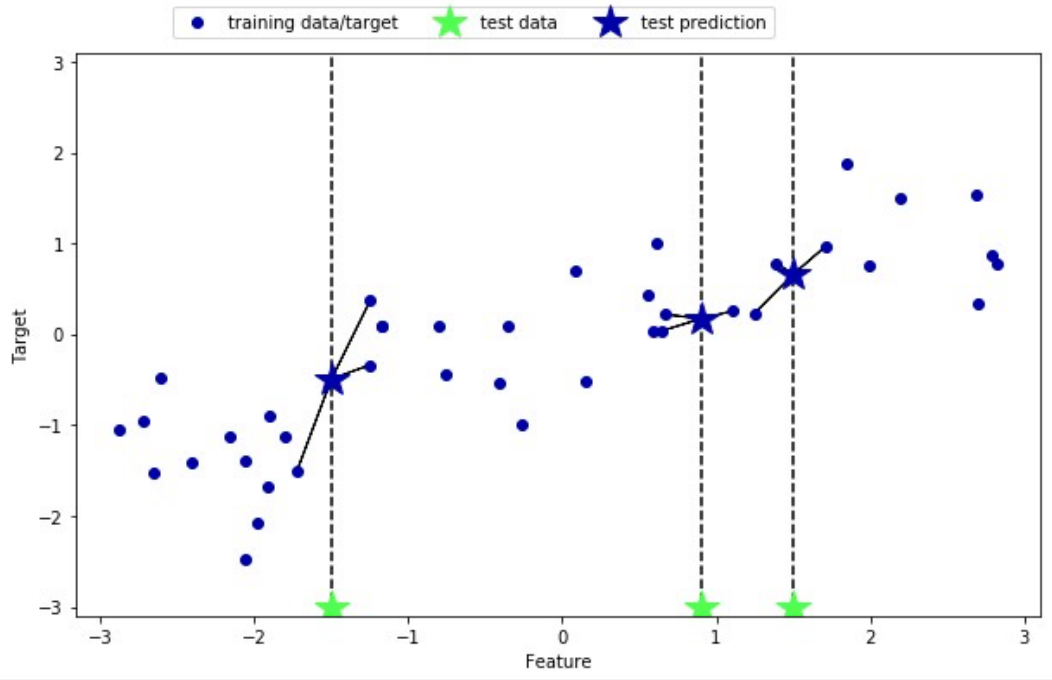
```
mglearn.plots.plot_knn_regression(n_neighbors=1)
```



We've added three test data points as green stars on the x-axis. The prediction using a single neighbor is just the target value of the nearest neighbor.

20. When using multiple nearest neighbors, the prediction is the average, or mean, of the relevant neighbors.

```
mglearn.plots.plot_knn_regression(n_neighbors=3)
```



21. The k-nearest neighbors algorithm for regression is implemented in the KNeighbors Regressor class in scikit-learn. It's used similarly to KNeighborsClassifier.

```python
from sklearn.neighbors import KNeighborsRegressor
X, y = mglearn.datasets.make_wave(n_samples=40)

# split the wave dataset into a training and a test set
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

# instantiate the model and set the number of neighbors to consider to 3
reg = KNeighborsRegressor(n_neighbors=3)

# fit the model using the training data and training targets
reg.fit(X_train, y_train)

print("Test set: \n{} \n{}\n".format(X_test, y_test))
print("Test set predictions: \n{}".format(reg.predict(X_test)))
```

22. We can also evaluate the model using the score method, which for regressors returns the R2 score.
    The R2 score, also known as the coefficient of determination, is a measure of goodness of a prediction for a regression model, and yields a score between 0 and 1.

```python
print("Test set R^2: {:.2f}".format(reg.score(X_test, y_test)))
```

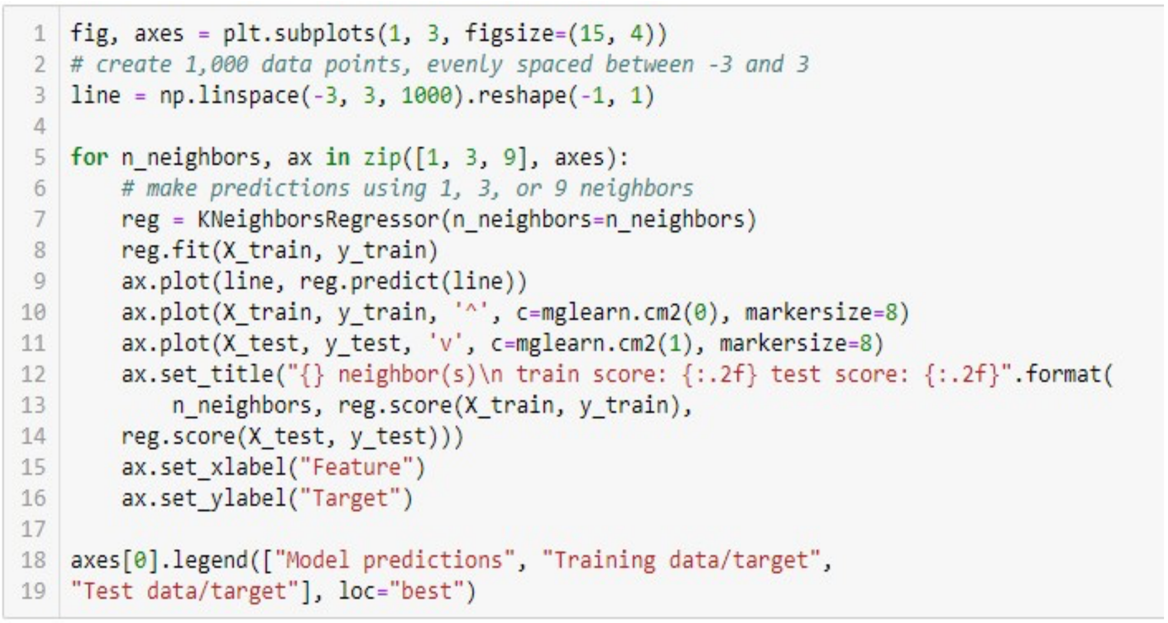
```
Test set R^2: 0.83
```

Value of 1 corresponds to a perfect prediction, and a value of 0 corresponds to a constant model that just predicts the mean of the training set responses.

<u>Analyzing KNeighborsRegressor</u>

For our one-dimensional dataset, we can see what the predictions look like for all possible feature values. To do this, we create a test dataset consisting of many points on the line.

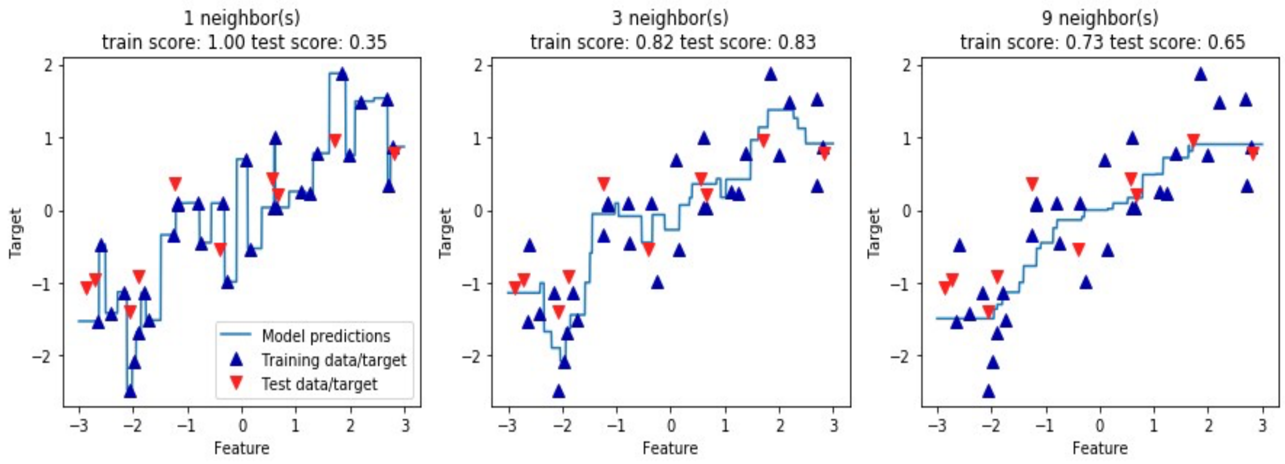
23. The k-nearest neighbors classification algorithm is implemented in the KNeighborsClassifier class in the neighbors module. Before we can use the model, we need to instantiate the class into an object. This is when we will set any parameters of the model.



```python
fig, axes = plt.subplots(1, 3, figsize=(15, 4))
# create 1,000 data points, evenly spaced between -3 and 3
line = np.linspace(-3, 3, 1000).reshape(-1, 1)

for n_neighbors, ax in zip([1, 3, 9], axes):
    # make predictions using 1, 3, or 9 neighbors
    reg = KNeighborsRegressor(n_neighbors=n_neighbors)
    reg.fit(X_train, y_train)
    ax.plot(line, reg.predict(line))
    ax.plot(X_train, y_train, '^', c=mglearn.cm2(0), markersize=8)
    ax.plot(X_test, y_test, 'v', c=mglearn.cm2(1), markersize=8)
    ax.set_title("{} neighbor(s)\n train score: {:.2f} test score: {:.2f}".format(
        n_neighbors, reg.score(X_train, y_train),
    reg.score(X_test, y_test)))
    ax.set_xlabel("Feature")
    ax.set_ylabel("Target")

axes[0].legend(["Model predictions", "Training data/target",
"Test data/target"], loc="best")
```



As we can see from the plot, using only a single neighbor, each point in the training set has an obvious influence on the predictions, and the predicted values go through all of the data points. This leads to a very unsteady prediction. Considering more neighbors leads to smoother predictions, but these do not fit the training data as well.

REFERENSI

1. Geron A. 2017. Hands on Machine Learning wirh Scikit Learn and TensorFlow. O Reilly Media Inc
2. Van derPlas J. 2016. Pyton Data Science Handbook. Oreilly Media Inc
3. Raschka S, Mirjailili. 2017. Phython Machine Leraning 2nd edition. Packt Publishing