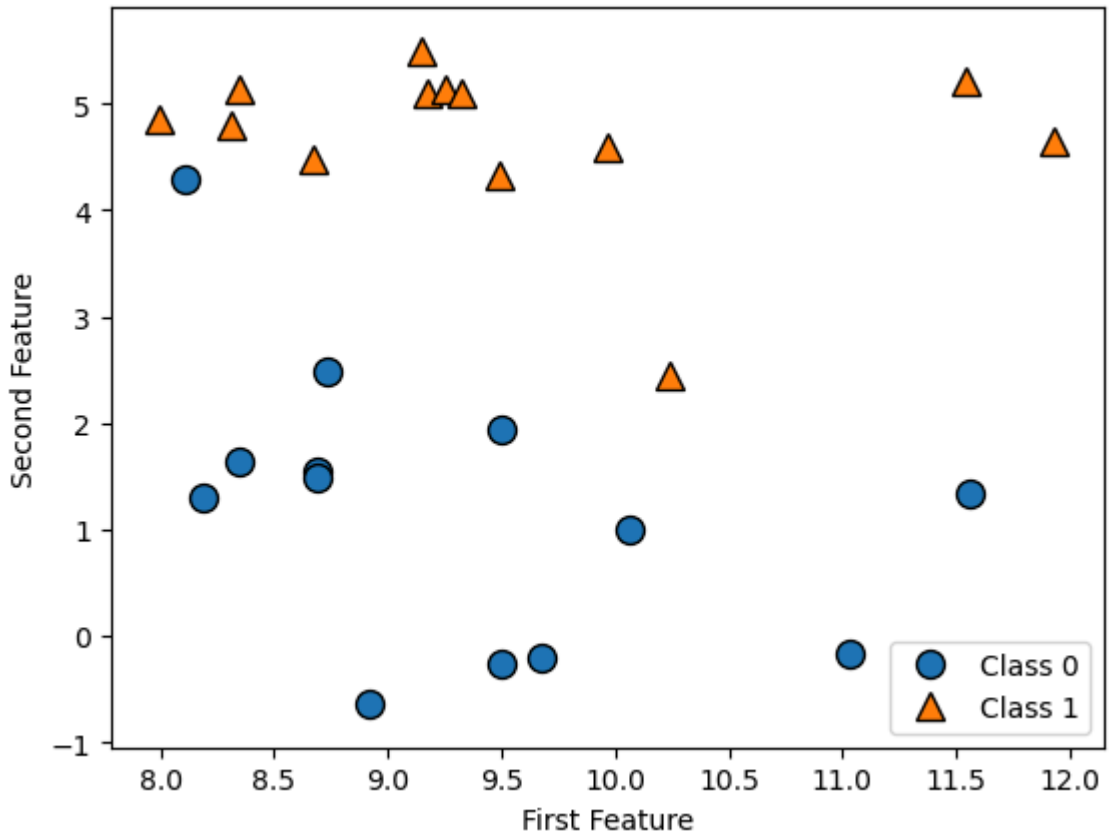
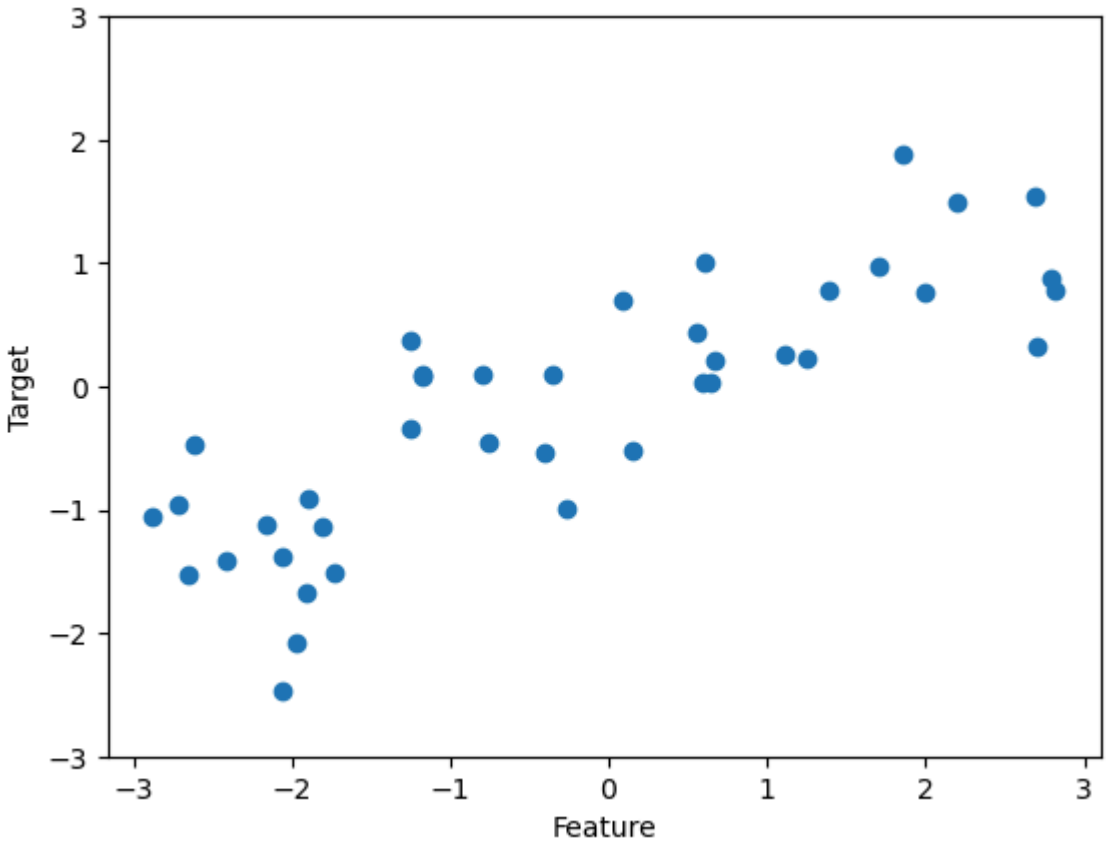
In [42]:
```
# SupervisedLearning
import matplotlib.pyplot as plt
import mglearn
# 用于分类的forge数据集，2个特征输入
# 生成forge样本的特征X和目标y
X, y = mglearn.datasets.make_forge()
# 使用样本的第0列特征和第1列特征作为绘制的横坐标和纵坐标，y作为图案
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
# 在右下角画一个图例，即2个分类
plt.legend(["Class 0", "Class 1"], loc = 4)
# 横坐标说明
plt.xlabel("First Feature")
# 纵坐标说明
plt.ylabel("Second Feature")
# 样本的个数和特征的维度
print("X.shape: {}".format(X.shape))
```

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



In [44]:
```
# 用于回归的wave数据集，1个特征输入
# 构造40个样本
X, y = mglearn.datasets.make_wave(n_samples = 40)
# X只有1维，可以直接画散点图
plt.plot(X, y, 'o')
# y的连续值范围
plt.ylim(-3, 3)
# 横坐标说明
plt.xlabel("Feature")
# 纵坐标说明
plt.ylabel("Target")
```

Out[44]:
```
Text(0, 0.5, 'Target')
```

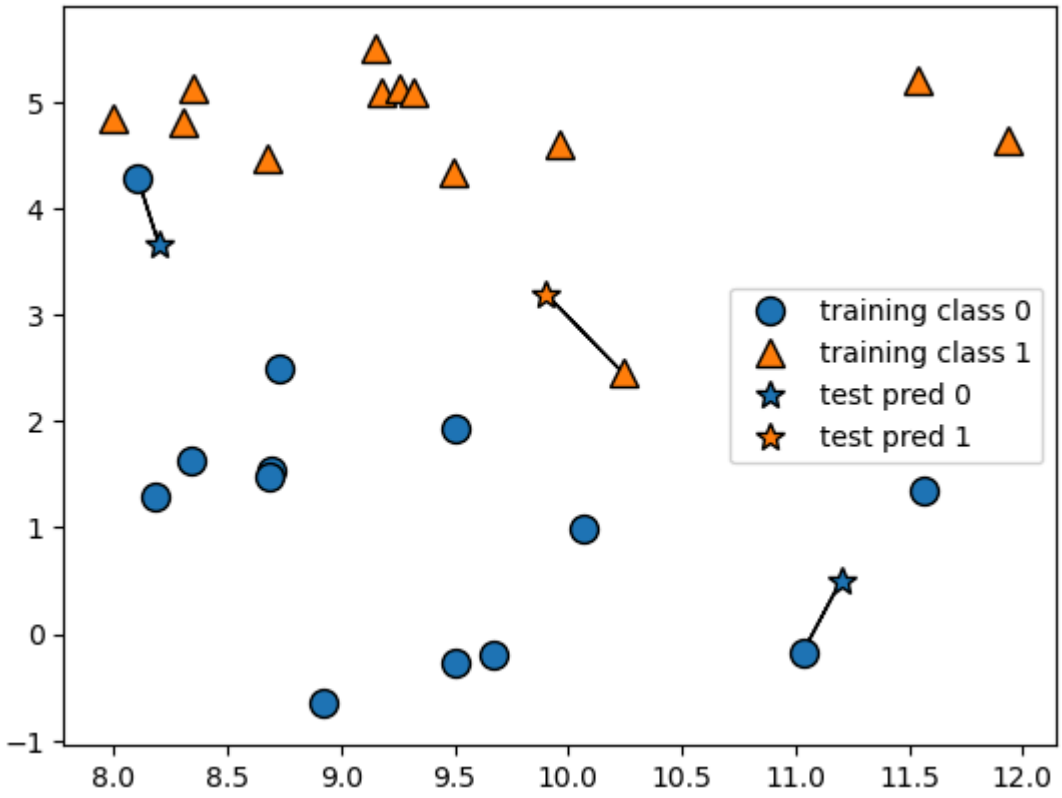


In [46]:

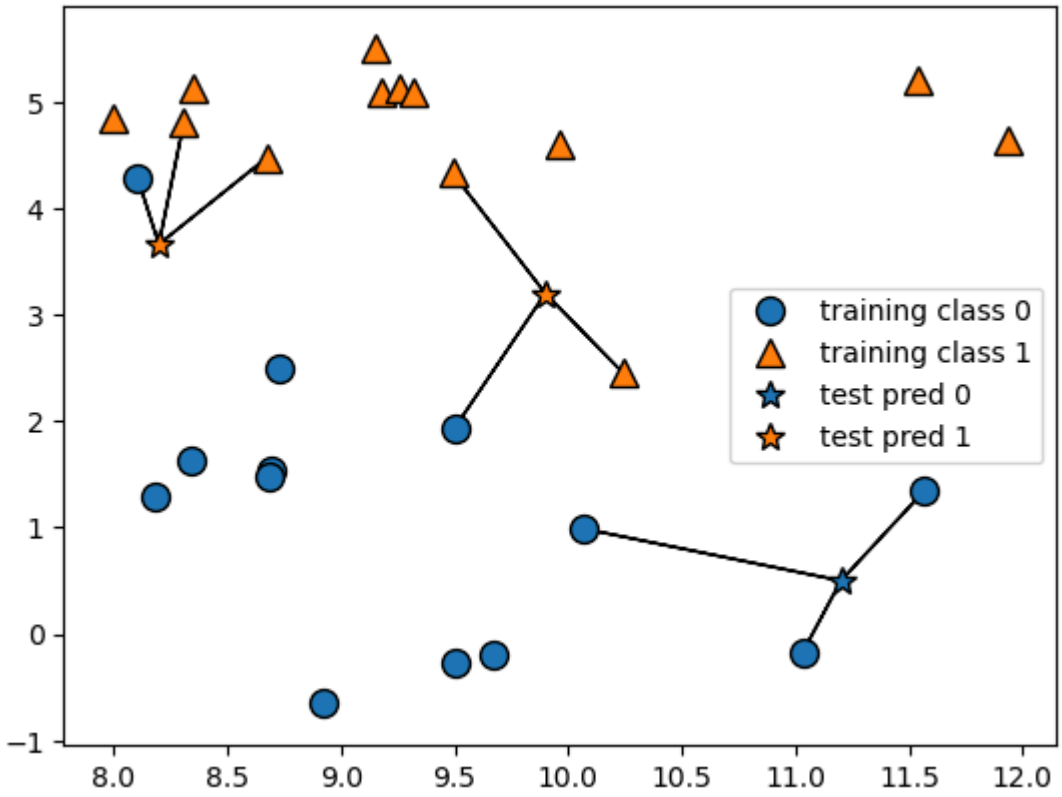
```
from sklearn.datasets import load_breast_cancer
# 加载数据集
cancer = load_breast_cancer()
# 打印样本规模和特征规模
print("cancer.keys(): \n{}".format(cancer.keys()))
print("Shape of cancer data: {}".format(cancer.data.shape))
# 打印不同分类的样本数量，np.bincount统计不同分类的个数
# 然后与分类的名字做1:1 zip，得到每个分类的样本数量
print("Sample counts per class:\n{}".format(
    {n: v for n, v in zip(cancer.target_names, np.bincount(cancer.target))}))
print("Feature names:\n{}".format(cancer.feature_names))
# boston房价数据集存在伦理问题
# 此外，创建该数据集的研究目标是研究空气质量的影响
# 但该研究未能充分证明这一假设的有效性
# 在sklearn1.2移除了boston数据集
# 用于回归的california房价数据集
from sklearn.datasets import fetch_california_housing
housing = fetch_california_housing()
print("Data shape:{}".format(housing.data.shape))
# 对原有特征经过简单的"特征工程"，增加了若干组合特征，
from sklearn.datasets import fetch_openml
housing = fetch_openml(name="house_prices", as_frame=True)
print("Data shape:{}".format(housing.data.shape))
```

```
cancer.keys():
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names',
'filename', 'data_module'])
Shape of cancer data: (569, 30)
Sample counts per class:
{'malignant': 212, 'benign': 357}
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']
Data shape:(20640, 8)
Data shape:(1460, 80)
```

In [48]:
```
# k-近邻
# 在最简单的版本中，k-NN算法只考虑一个最近邻
mglearn.plots.plot_knn_classification(n_neighbors=1)
```



In [50]:
```
# 不仅可以考虑最近的一个邻居，还可以考虑任意数量的邻居
# 记为k，这也是k-近邻算法名称的由来
mglearn.plots.plot_knn_classification(n_neighbors=3)
```

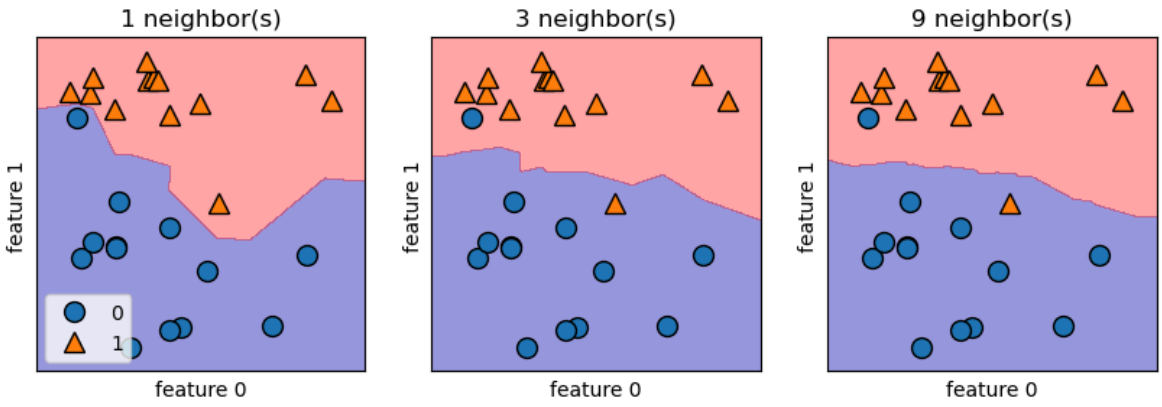


In [52]:
```
# 将数据分为训练集和测试集，以便评估泛化性能
from sklearn.model_selection import train_test_split
X, y = mglearn.datasets.make_forge()
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
# 使用训练集来拟合分类器
# 对于KNeighborsClassifier来说，这意味着存储数据集，以便在预测时计算邻居
from sklearn.neighbors import KNeighborsClassifier
clf = KNeighborsClassifier(n_neighbors=3)
clf.fit(X_train, y_train)
# 测试数据进行预测，调用predict方法
# 该方法会计算每个数据点在训练集中的最近邻，并找到这些邻居中最常见的类别
print("Test set predictions:{}".format(clf.predict(X_test)))
print("Test set accuracy:{}".format(clf.score(X_test, y_test)))
```

```
Test set predictions:[1 0 1 0 1 0 0]
Test set accuracy:0.8571428571428571
```

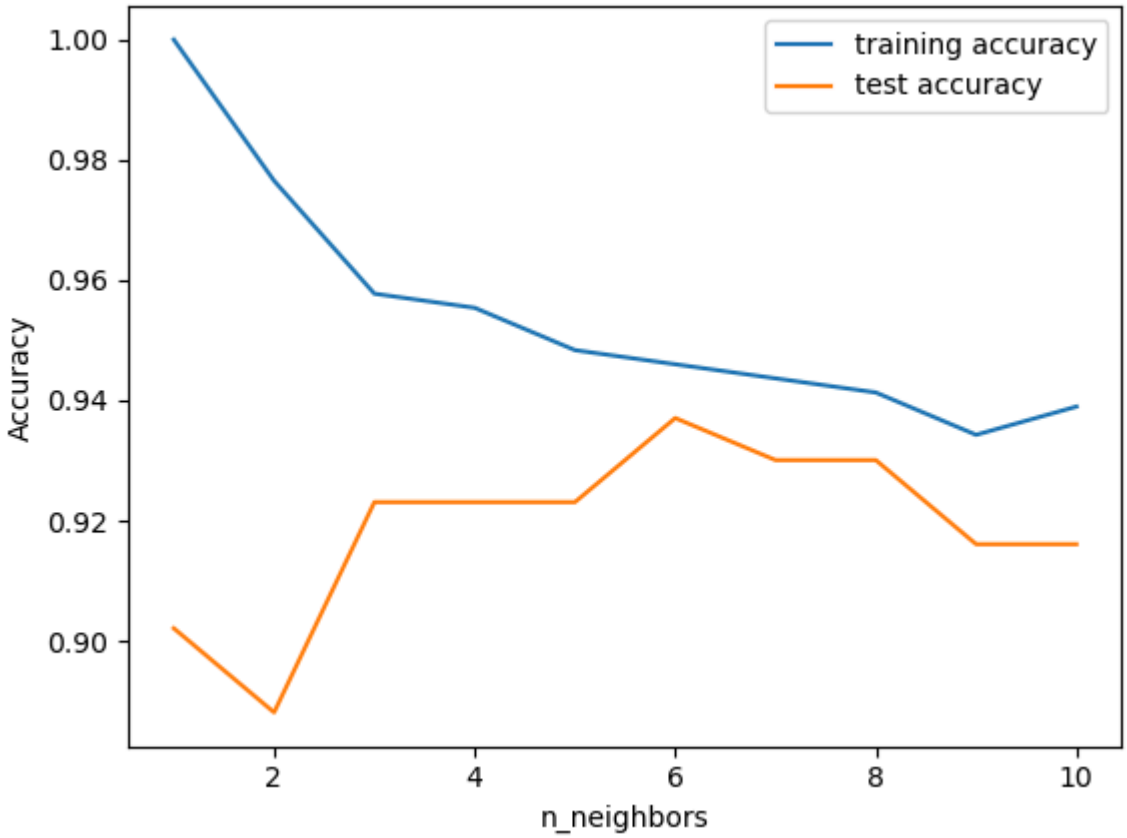
In [54]:
```
fig, axes = plt.subplots(1, 3, figsize=(10, 3))
for n_neighbors, ax in zip([1, 3, 9], axes):
    # 训练模型对象
    clf = KNeighborsClassifier(n_neighbors=n_neighbors).fit(X, y)
    mglearn.plots.plot_2d_separator(clf, X, fill=True, eps=0.5, ax=ax, alpha=.4)
    mglearn.discrete_scatter(X[:, 0], 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)
# 使用单个邻居会导致一个紧密跟随训练数据的决策边界
# 考虑越来越多的邻居会导致决策边界更加平滑
# 平滑的边界对应于更简单的模型
# 使用较少的邻居对应于高模型复杂度
# 使用较多的邻居对应于低模型复杂度
# 如果考虑极端情况，即邻居的数量等于训练集中所有数据点的数量
# 则每个测试点将有完全相同的邻居（所有训练点）
# 所有预测结果将相同：即训练集中最常见的类别
```

Out[54]: <matplotlib.legend.Legend at 0x1d93e85e480>



In [55]:
```
# 分类cancer
from sklearn.datasets import load_breast_cancer
# 加载数据
cancer = load_breast_cancer()
# 切分
X_train, X_test, y_train, y_test = train_test_split(
 cancer.data, cancer.target, stratify=cancer.target, random_state=66)
# 记录不同n_neighbors情况下，模型的训练集精度与测试集精度的变化
training_accuracy = []
test_accuracy = []
# n_neighbors取值从1到10
neighbors_settings = range(1, 11)
for n_neighbors in neighbors_settings:
    # 模型对象
    clf = KNeighborsClassifier(n_neighbors=n_neighbors)
    clf.fit(X_train, y_train)
    # 记录训练集精度
    training_accuracy.append(clf.score(X_train, y_train))
    # 记录测试集精度
    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()
# 调大n_neighbors则导致训练集精度下降，测试集精度上升
# 折衷点在n_neighbors=6，此时模型既不会过拟合也不会欠拟合，这就是调参
```

Out[55]: <matplotlib.legend.Legend at 0x1d9463ea270>


training accuracy
test accuracy
1.00
0.98
0.96
0.94
0.92
0.90
Accuracy
2
4
6
8
10
n_neighbors