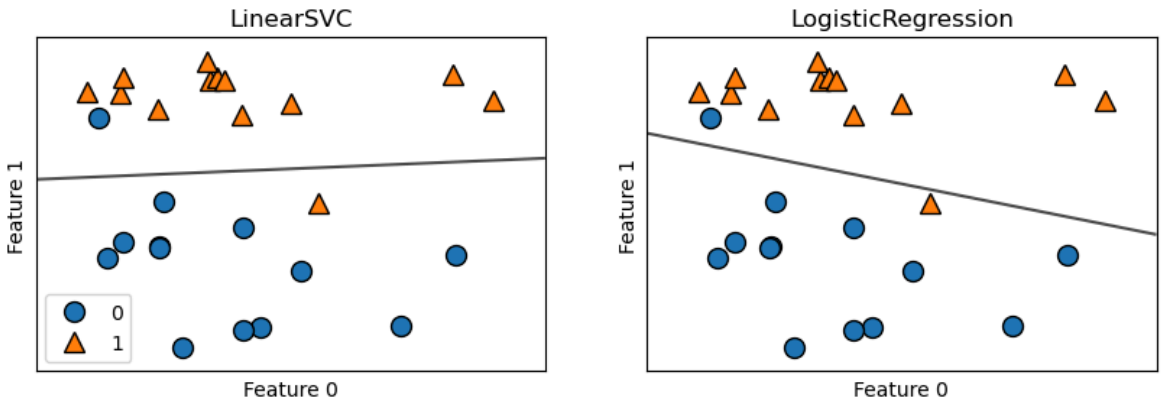
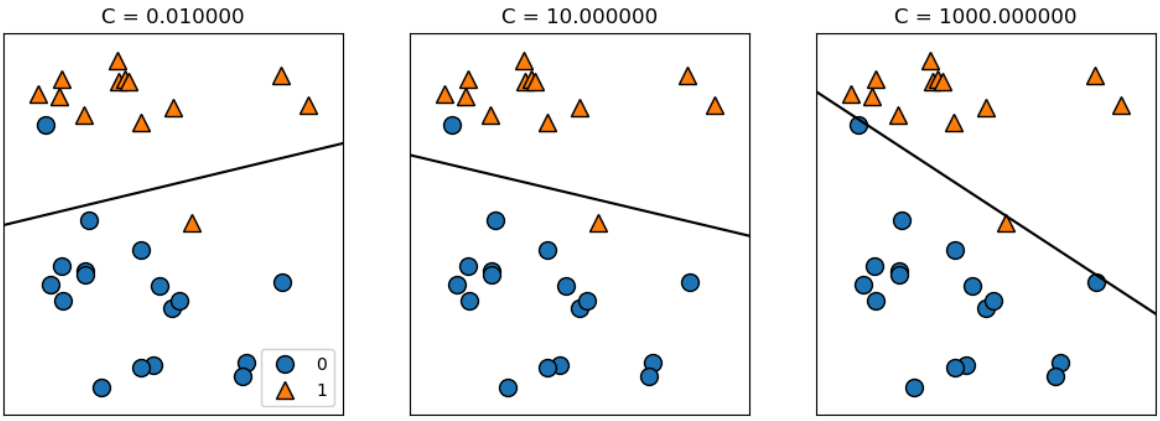
In [4]:
```
# LinearModels For Classification
# 线性模型在分类问题中也被广泛应用。对于二元分类，其预测公式如下
# ŷ = w[0] * x[0] + w[1] * x[1] + ... + w[p] * x[p] + b
# 在分类中，线性模型的公式看起来与线性回归的公式非常相似
# 但在处理预测值时，有一个关键区别：分类通过零作为阈值来判断类别
# 如果函数输出值小于零，预测类别为-1
# 如果函数输出值大于零，预测类别为+1
# 这一预测规则是所有线性分类模型的共同点
# 线性回归的输出ŷ可以是直线、平面或更高维空间的超平面
# 线性分类的决策边界：通过线、平面或超平面将两个类别分隔开
# 线性模型算法的不同主要体现在以下两方面：
# 拟合数据的方法：衡量参数w和b对训练数据的拟合程度的方式不同
# 正则化的使用：是否使用正则化，以及使用何种正则化（如 L1、L2）
# 最常用的两种线性分类算法是：
# 逻辑回归，利用逻辑函数估计样本属于特定类别的概率
# 线性支持向量机：与逻辑回归不同，它使用最大化分类间隔的原则
import mglearn
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
X, y = mglearn.datasets.make_forge()
fig, axes = plt.subplots(1, 2, figsize=(10, 3))

for model, ax in zip([LinearSVC(), LogisticRegression()], axes):
    clf = model.fit(X, y)
    mglearn.plots.plot_2d_separator(clf, X, fill=False, eps=0.5,
                                    ax=ax, alpha=0.7)
    mglearn.discrete_scatter(X[:, 0], X[:, 1], y, ax=ax)
    ax.set_title("{}".format(clf.__class__.__name__))
    ax.set_xlabel("Feature 0")
    ax.set_ylabel("Feature 1")
axes[0].legend()
# 图中分别展示了LinearSVC和LogisticRegression模型找到的决策边界
# 决策边界将上方分类为类别1的区域与下方分类为类别0的区域分隔开
# 位于直线边界之上的数据点都会被归为类别1
# 而位于黑线之下的点则会被归为类别0
# 这两个模型得出的决策边界非常相似，且都错误分类了其中两个数据点
# 默认情况下，这两个模型都使用了L2正则化
# 这种正则化方式与Ridge回归中采用的方式相同
```

Out[4]:
```
<matplotlib.legend.Legend at 0x253ceffbb30>
```



In [5]:
```
# 控制正则化强度的权衡参数称为C，其值越大，对正则化的约束越弱
# 当C取较高值时，LogisticRegression和LinearSVC会尽可能拟合训练集
# 当C取较低值时，模型会更强调找到接近零的系数向量，即更强的正则化
```

```
# 较低的C值会使算法更倾向于适应数据点的“大多数”，忽略少量异常点
# 而使用较高的C值则会强调准确分类每一个数据点的重要性
mglearn.plots.plot_linear_svc_regularization()
# 在左图中，使用了一个非常小的 C 值，这对应于强正则化
# 属于类别0的大多数点位于顶部，而属于类别 1 的大多数点位于底部
# 强正则化的模型选择了一条相对水平的决策边界，导致两个点被错误分类
# 在中间图中，C值略高
# 模型更多地关注之前被错误分类的两个样本，因此决策边界有所倾斜
# 在右图中，C值非常高，模型将决策边界大幅倾斜
# 正确分类了类别0中的所有点，类别1中的一个点仍然被错误分类
# 因为用一条直线无法正确分类这个数据集中的所有点
# 高C模型虽尽力正确分类了所有点，但可能无法很好地捕捉整体分布情况
# 换句话说，该模型可能出现了过拟合
# 线性分类模型类似于线性回归
# 低维空间的决策边界看起来可能非常受限，仅允许是直线或平面
# 在高维空间中，线性分类模型会变得非常强大
# 同时随着特征数量的增加，防止过拟合显得更加重要
```



In [6]:
```
# LinearLogistic模型在乳腺癌数据集上的表现
from sklearn.model_selection import train_test_split
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=42)
logreg1 = LogisticRegression(C=1, solver='liblinear').fit(X_train, y_train)
print("Training set score: {:.3f}".format(logreg1.score(X_train, y_train)))
print("Test set score: {:.3f}".format(logreg1.score(X_test, y_test)))
```

```
Training set score: 0.953
Test set score: 0.958
```

In [7]:
```
# 当我们使用C=1，模型在训练集和测试集上的准确率都达95%左右
# 由于训练集和测试集的性能非常接近，这表明模型可能存在欠拟合问题
# 即模型复杂度不足，未能充分捕捉数据的潜在模式
# 为了缓解欠拟合问题，可以通过增大 C的值来训练一个更加灵活的模型
# 更高的C值会降低正则化的强度，从而允许模型更关注于拟合训练数据
logreg100 = LogisticRegression(C=100, solver='liblinear').fit(X_train, y_train)
print("Training set score: {:.3f}".format(logreg100.score(X_train, y_train)))
print("Test set score: {:.3f}".format(logreg100.score(X_test, y_test)))
```

```
Training set score: 0.974
Test set score: 0.965
```

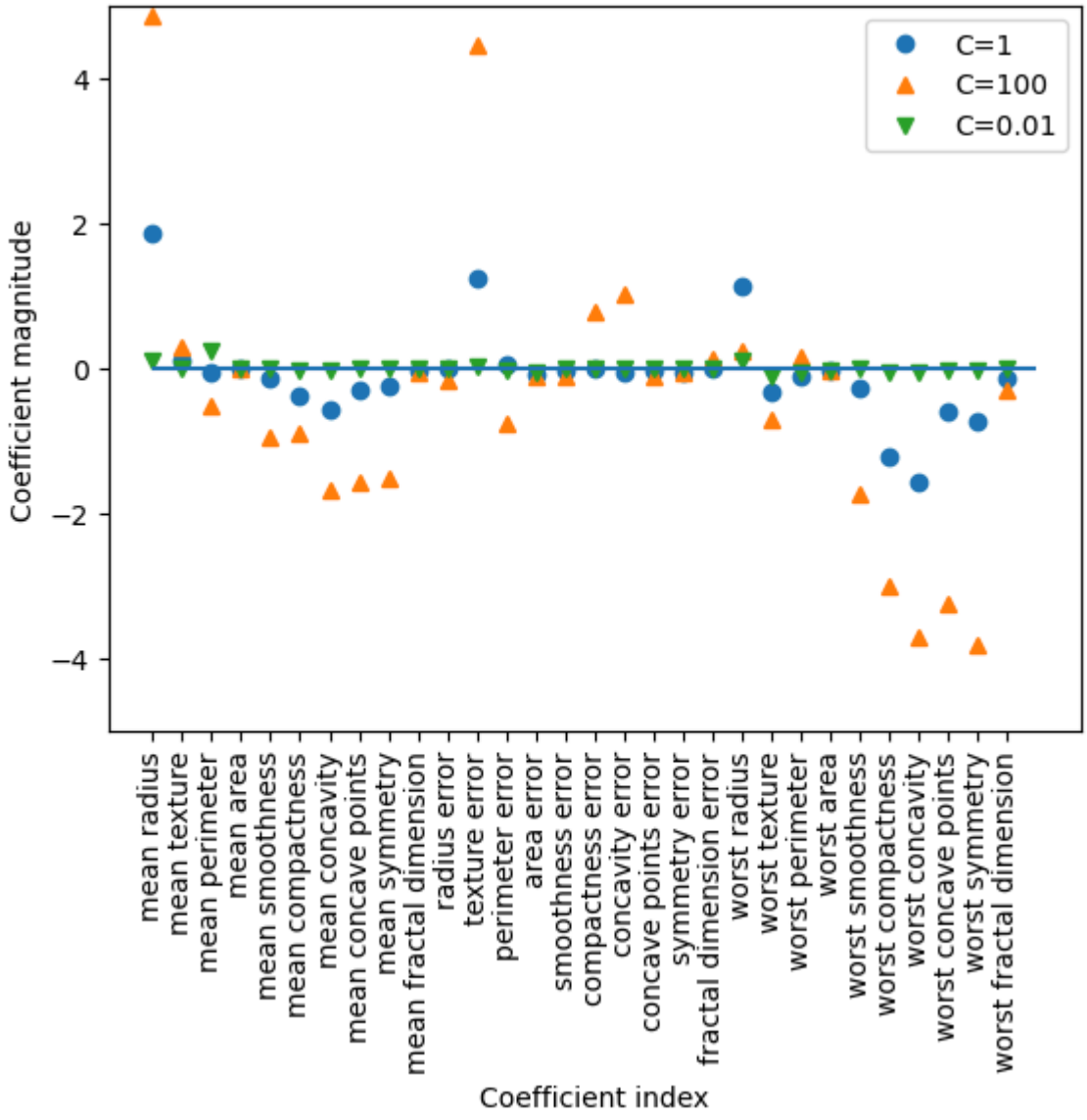
In [8]:
```
# 当C设置为更大的值，训练集的准确率提高，测试集的准确率也略有增加
# 验证了：更复杂的模型在此情况下可能表现得更好。
# 然而，如果尝试使用比C=1更强的正则化，例如C=0.01
logreg001 = LogisticRegression(C=0.01, solver='liblinear').fit(X_train, y_train)
```

```
print("Training set score: {:.3f}".format(logreg001.score(X_train, y_train)))
print("Test set score: {:.3f}".format(logreg001.score(X_test, y_test)))
```

```
Training set score: 0.934
Test set score: 0.930
```

In [11]:
```
# 正如预期，训练集和测试集的准确率都会降低
plt.plot(logreg1.coef_.T, 'o', label="C=1")
plt.plot(logreg100.coef_.T, '^', label="C=100")
plt.plot(logreg001.coef_.T, 'v', label="C=0.01")
plt.xticks(range(cancer.data.shape[1]), cancer.feature_names, rotation=90)
plt.hlines(0, 0, cancer.data.shape[1])
plt.ylim(-5, 5)
plt.xlabel("Coefficient index")
plt.ylabel("Coefficient magnitude")
plt.legend()
```

Out[11]: <matplotlib.legend.Legend at 0x253d0d55cd0>



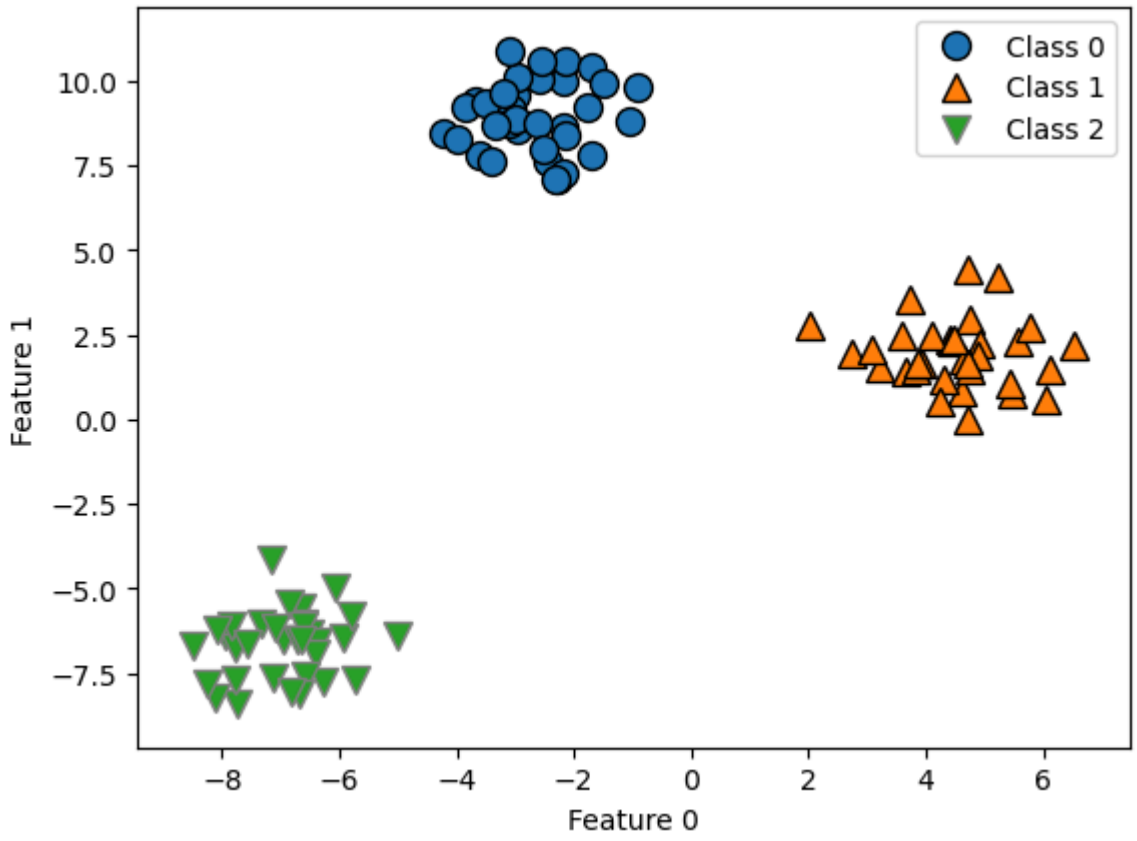
In [20]:
```
# 如果希望模型更具可解释性，使用 L1 正则化可能会有所帮助
# 因为它限制模型仅使用少数特征，令部分w系数为0
# 相当于对意义不大的特征进行了淘汰
for C, marker in zip([0.001, 1, 100], ['o', '^', 'v']):
    # 利用C控制正则化强弱，penalty指定了L1正则化
    lr_l1 = LogisticRegression(C=C, penalty="l1", solver='liblinear').fit(X_trai
    # 训练集精度
```

```
    print("Training accuracy of l1 logreg with C={:.3f}: {:.2f}".format(
        C, lr_l1.score(X_train, y_train)))
    # 测试集精度
    print("Test accuracy of l1 logreg with C={:.3f}: {:.2f}".format(
        C, lr_l1.score(X_test, y_test)))

# 原书代码的问题在于使用了LogisticRegression的penalty="l1"参数
# 但没有指定solver参数，而默认的solver="lbfgs"并不支持 penalty="l1"
# 在定义LogisticRegression时，添加solver="liblinear"即可
```

```
Training accuracy of l1 logreg with C=0.001: 0.91
Test accuracy of l1 logreg with C=0.001: 0.92
Training accuracy of l1 logreg with C=1.000: 0.96
Test accuracy of l1 logreg with C=1.000: 0.96
Training accuracy of l1 logreg with C=100.000: 0.99
Test accuracy of l1 logreg with C=100.000: 0.98
```

In [22]:
```
# Linear Models For multiclass classification
# 许多线性分类模型仅适用于二元分类，不能自然地扩展到多类分类
# （逻辑回归是个例外）
# 一种常用的将二元分类算法扩展到多类分类算法的方法是one-vs.-rest方法
# 为每个类别学习一个二元模型，该模型试图将该类别与所有其他类别分开，最终得到与类别
# 进行预测时，所有的二元分类器都会在测试点上运行，得分最高的分类器“胜出”，并返回其
# 创建一个二维数据集，其中每个类别由从高斯分布采样的数据表示
import mglearn
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
X, y = make_blobs(random_state=42)
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
plt.xlabel("Feature 0")
plt.ylabel("Feature 1")
plt.legend(["Class 0", "Class 1", "Class 2"])
```

Out[22]:
```
<matplotlib.legend.Legend at 0x253d1328830>
```

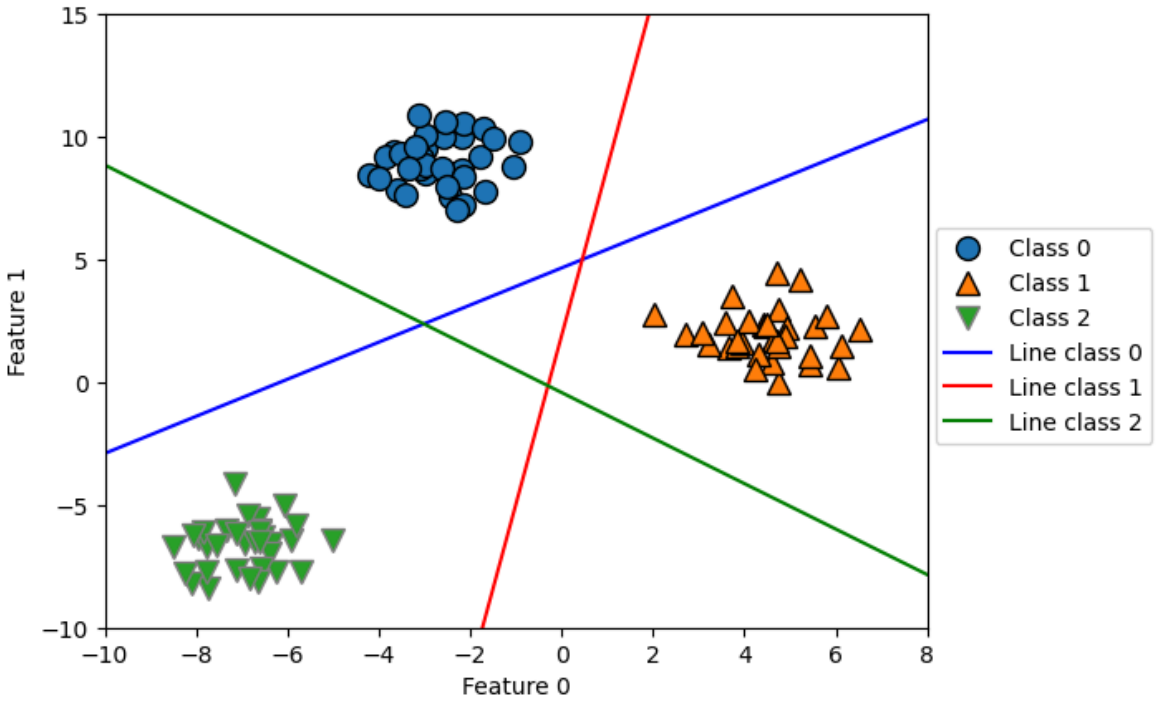


In [24]:
```
# LinearSVC分类器
linear_svm = LinearSVC().fit(X, y)
print("Coefficient shape: ", linear_svm.coef_.shape)
print("Intercept shape: ", linear_svm.intercept_.shape)
# coef_ 的形状是3*2的矩阵，这意味着 coef_有三行都包含一个类别的系数向量，而对应于
# intercept_ 现在是一个一维数组，用于存储每个类别的截距。
```

```
Coefficient shape:  (3, 2)
Intercept shape:  (3,)
```

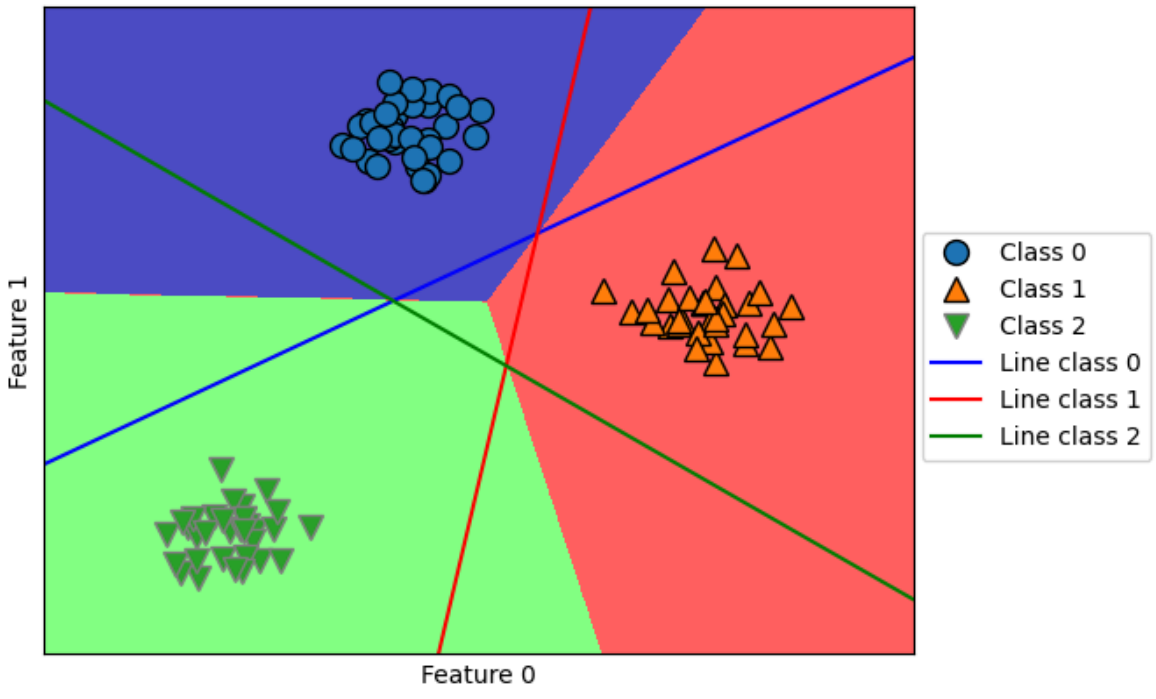
In [26]:
```
# 可视化3个分类器划定的边界
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
line = np.linspace(-15, 15)
for coef, intercept, color in zip(linear_svm.coef_, linear_svm.intercept_,
 ['b', 'r', 'g']):
 plt.plot(line, -(line * coef[0] + intercept) / coef[1], c=color)
plt.ylim(-10, 15)
plt.xlim(-10, 8)
plt.xlabel("Feature 0")
plt.ylabel("Feature 1")
plt.legend(['Class 0', 'Class 1', 'Class 2', 'Line class 0', 'Line class 1',
 'Line class 2'], loc=(1.01, 0.3))
```

Out[26]:
```
<matplotlib.legend.Legend at 0x253d13d10d0>
```



In [28]:
```
# 二维空间所有区域预测
mglearn.plots.plot_2d_classification(linear_svm, X, fill=True, alpha=.7)
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
line = np.linspace(-15, 15)
for coef, intercept, color in zip(linear_svm.coef_, linear_svm.intercept_,
                                  ['b', 'r', 'g']):
    plt.plot(line, -(line * coef[0] + intercept) / coef[1], c=color)
    plt.legend(['Class 0', 'Class 1', 'Class 2', 'Line class 0', 'Line class 1',
                'Line class 2'], loc=(1.01, 0.3))
plt.xlabel("Feature 0")
plt.ylabel("Feature 1")
```

Out[28]:
```
Text(0, 0.5, 'Feature 1')
```



In [ ]:
```
# 线性模型的主要参数是正则化参数，回归模型中称为alpha，在LinearSVC和LogisticRegre
# 较大的alpha值或较小的C值意味着简单的模型
```

```
# 特别是在回归模型中，调优这些参数非常重要。通常，C和alpha的搜索是在对数尺度上进行
# 另一个决策是是否希望使用L1正则化或L2正则化
# 如果只有少数特征实际上很重要，应该使用L1。否则，默认使用L2。
# 如果模型的可释性很重要，L1也很有用。因为L1只会使用少数特征，所以更容易解释哪些特
# 线性模型训练速度非常快，预测也很快。它们能够扩展到非常大的数据集，并且在稀疏数据
# 线性模型的另一个优点是它们使得理解预测的过程相对简单，使用先前提及的回归和分类公
# 但通常并不完全清楚为何系数会是这样的，尤其是当数据集具有高度相关的特征时，在这种
```