In [1]:
```
import mglearn
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Logistic Regression

In [2]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.model_selection import train_test_split

from sklearn.datasets import load_breast_cancer
```

By default, model applies an L2 regularization, in the same way that Ridge does for regression.

In [3]:
```
cancer = load_breast_cancer()
```

In [4]:
```
X_train, X_test, y_train, y_test = train_test_split(
 cancer.data, cancer.target, stratify=cancer.target, random_state=42)
```

In [5]:
```
logreg = LogisticRegression(solver='liblinear').fit(X_train, y_train)
##You can try different solvers if lbfgs isn't performing well --> different solvers (l
```

In [6]:
```
print("Training set score: {:.3f}".format(logreg.score(X_train, y_train)))
print("Test set score: {:.3f}".format(logreg.score(X_test, y_test)))
```

```
Training set score: 0.955
Test set score: 0.958
```

The default value of C=1 provides quite good performance, with 95% accuracy on both the training and the test set.

In [7]:
```
logreg100 = LogisticRegression(solver='liblinear', C=100).fit(X_train, y_train)
```

In [8]:
```
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: 0.972
Test set score: 0.965
```

Using C=100 results in higher training set accuracy, and also a slightly increased test set accuracy, confirming our intuition that a more complex model should perform better.

In [9]:
```
logreg001 = LogisticRegression(solver='liblinear', C=0.01).fit(X_train, y_train)
```

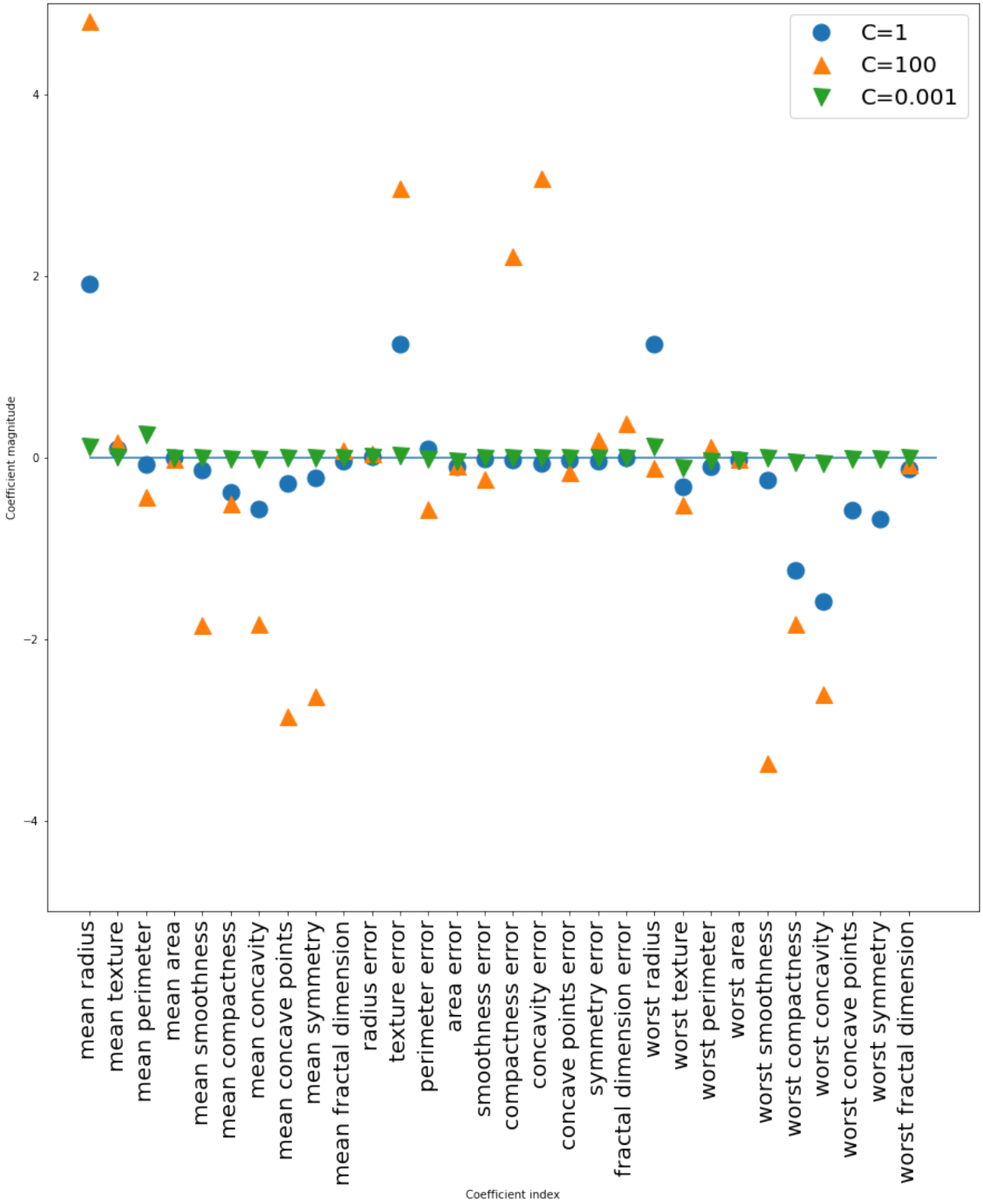
In [10]:
```
print("Training set score: {:.3f}".format(logreg001.score(X_train, y_train)))
print("Test set score: {:.3f}".format(logreg001.score(X_test, y_test)))
```

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

Both training and test set accuracy decrease relative to the default parameters.

In [11]:
```
plt.figure(figsize=(15,15))
plt.plot(logreg.coef_.T, 'o', label="C=1", markersize = 15)
plt.plot(logreg100.coef_.T, '^', label="C=100", markersize = 15)
plt.plot(logreg001.coef_.T, 'v', label="C=0.001", markersize = 15)
plt.xticks(range(cancer.data.shape[1]), cancer.feature_names, rotation=90, size=20)
plt.hlines(0, 0, cancer.data.shape[1])
plt.ylim(-5, 5)
plt.xlabel("Coefficient index")
plt.ylabel("Coefficient magnitude")
plt.legend(fontsize=20)
```

Out[11]:
```
<matplotlib.legend.Legend at 0x2173eb7a100>
```



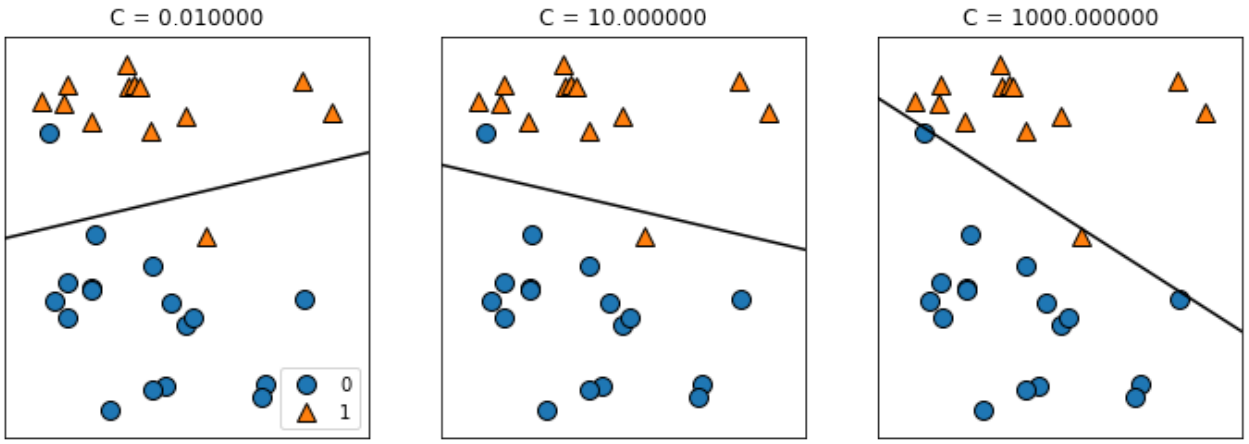
If we desire a more interpretable model, using L1 regularization might help, as it limits the model to using only a few features. Here is the coefficient plot and classification accuracies for L1 regularization.

Example : L1 regularization(penalty parameter)

lr_l1 = LogisticRegression(solver='liblinear', C=1, penalty="l1").fit(X_train, y_train)

Linear SVC

In [12]:
```
mglearn.plots.plot_linear_svc_regularization()
```



For LogisticRegression and LinearSVC the trade-off parameter that determines the strength of the regularization is called C, and higher values of C correspond to less regularization. In other words, when you use a high value for the parameter C, LogisticRegression and LinearSVC try to fit the training set as best as possible, while with low values of the parameter C, the models put more emphasis on finding a coefficient vector (w) that is close to zero.