CoxAssignment10

Task: This assignment takes you through exploring Support Vector Machines. This should give you a detailed look at how to classify data and the intuition behind how SVMs work.

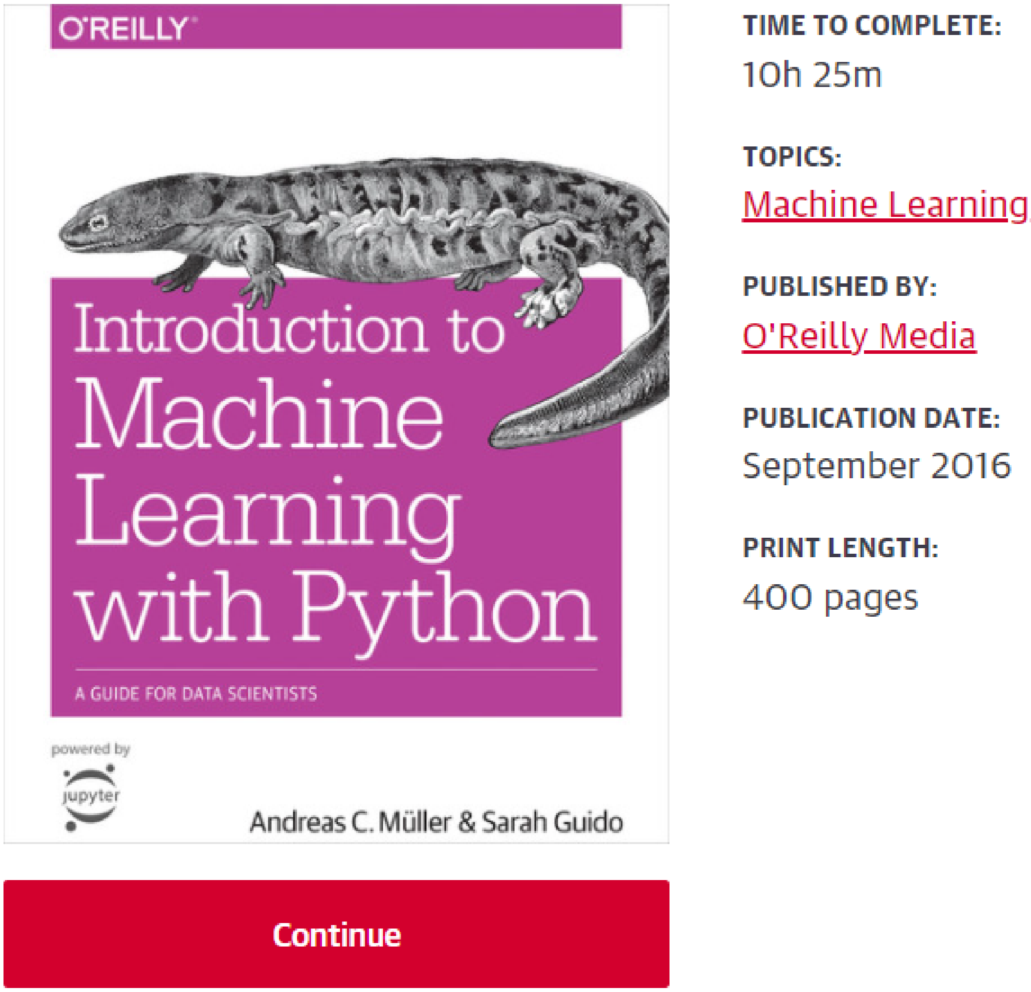
Dataset: Included in guide.

Instructions

Please follow the directions below to access the assignment:

1. Navigate to: https://library.dsu.edu/c.php?g=857936&p=6146747 and scroll down to "Featured Databases". Click on O'Reilly for Higher Education.
2. If you have an existing account, you will autosign in. If you do not, create a new account and make sure to use your DSU email address. DSU has free access to these technical books.
3. Click on the search bar in the top and search for "Introduction to Machine Learning with Python" The book is written by By Andreas C. Müller and Sarah Guido. Select that book

after searching. I have a picture shown below.



4. Select the table of contents button on the right side.
5. Select the content labeled: 2.3.7 Kernelized Support Vector Machines.
6. Write the code from the beginning up to "Strengths, weaknesses, and parameters."
7. As you are writing code, discuss what is happening in the notebook during each block of code you are writing.
8. At the end, summarize strengths, weaknesses, and the importance of SVM parameters.
9. Submit the Jupyter Notebook as a PDF named YourLastNameAssignment10.

In [27]:
```
from sklearn.datasets import make_blobs
import pandas as pd
import numpy as np
from sklearn import datasets
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import LinearSVC
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.datasets import make_moons
from sklearn.preprocessing import PolynomialFeatures
import matplotlib.pyplot as plt
from sklearn.svm import SVC
```

```
import mglearn
from sklearn.model_selection import train_test_split, GridSearchCV
```

In [36]:
```
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
print("cancer.keys():\n", cancer.keys())
```

```
cancer.keys():
 dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'fil
ename', 'data_module'])
```

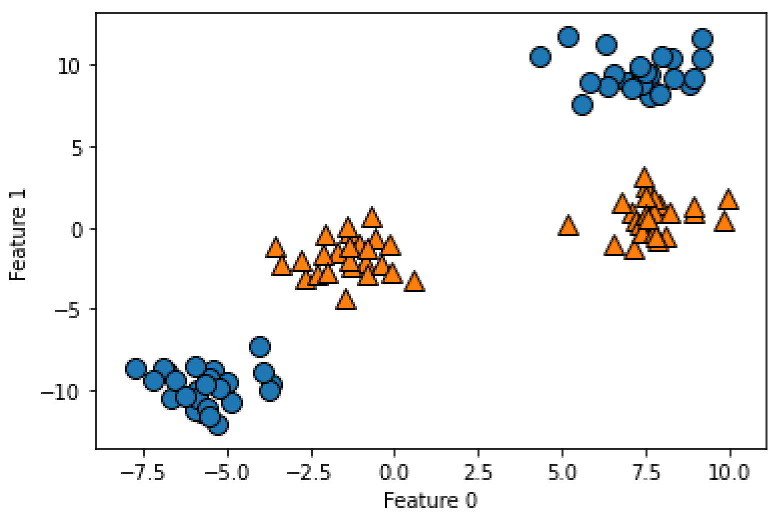
In [37]:
```
print("Shape of cancer data:", cancer.data.shape)
```

```
Shape of cancer data: (569, 30)
```

Linear Models and Nonlinear Features

In [28]:
```
X, y = make_blobs(centers=4, random_state=8)
y = y % 2

mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
plt.xlabel("Feature 0")
plt.ylabel("Feature 1")
```

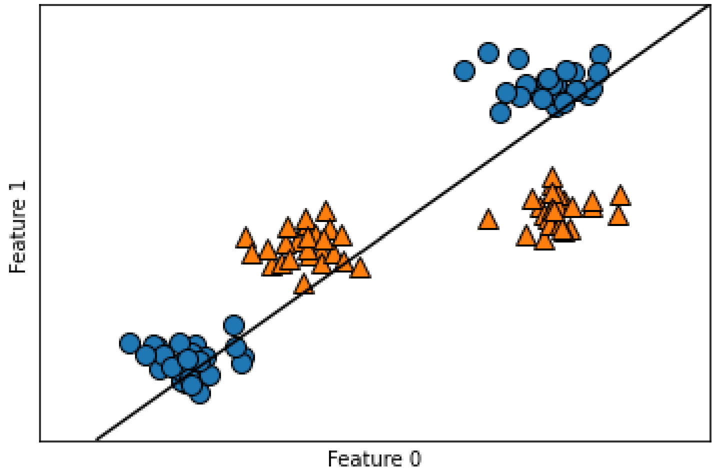
Out[28]:
```
Text(0, 0.5, 'Feature 1')
```



In [29]:
```
from sklearn.svm import LinearSVC
linear_svm = LinearSVC().fit(X, y)

mglearn.plots.plot_2d_separator(linear_svm, X)
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
plt.xlabel("Feature 0")
plt.ylabel("Feature 1")
```

```
C:\Users\jcjcb\anaconda3\lib\site-packages\sklearn\svm\_base.py:1206: ConvergenceWarn
ing: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
```

Out[29]:
```
Text(0, 0.5, 'Feature 1')
```

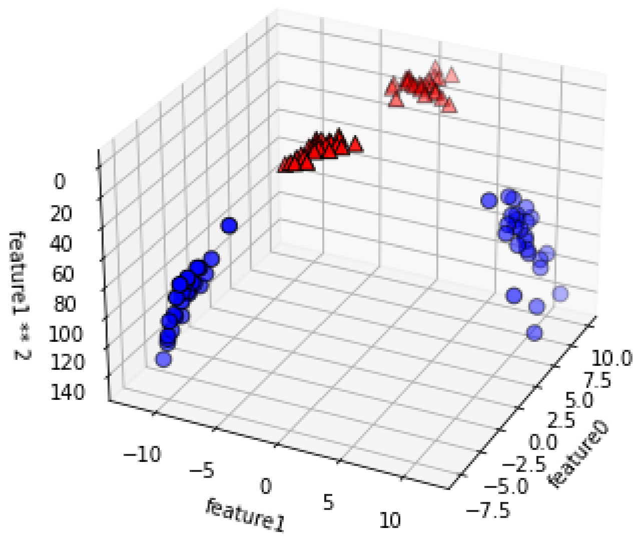


- The charts above is showing that a linear model classification does not do a great job on this data set as you can see some features are on both sides of the line

In [30]:
```
# add the squared second feature
X_new = np.hstack([X, X[:, 1:] ** 2])

from mpl_toolkits.mplot3d import Axes3D, axes3d
figure = plt.figure()
# visualize in 3D
ax = Axes3D(figure, elev=-152, azim=-26)
# plot first all the points with y == 0, then all with y == 1
mask = y == 0
ax.scatter(X_new[mask, 0], X_new[mask, 1], X_new[mask, 2], c='b',
           cmap=mglearn.cm2, s=60, edgecolor='k')
ax.scatter(X_new[~mask, 0], X_new[~mask, 1], X_new[~mask, 2], c='r', marker='^',
           cmap=mglearn.cm2, s=60, edgecolor='k')
ax.set_xlabel("feature0")
ax.set_ylabel("feature1")
ax.set_zlabel("feature1 ** 2")
```

```
C:\Users\jcjcb\AppData\Local\Temp\ipykernel_9764\4191468807.py:7: MatplotlibDeprecati
onWarning: Axes3D(fig) adding itself to the figure is deprecated since 3.4. Pass the
keyword argument auto_add_to_figure=False and use fig.add_axes(ax) to suppress this w
arning. The default value of auto_add_to_figure will change to False in mpl3.5 and Tr
ue values will no longer work in 3.6.  This is consistent with other Axes classes.
  ax = Axes3D(figure, elev=-152, azim=-26)
```

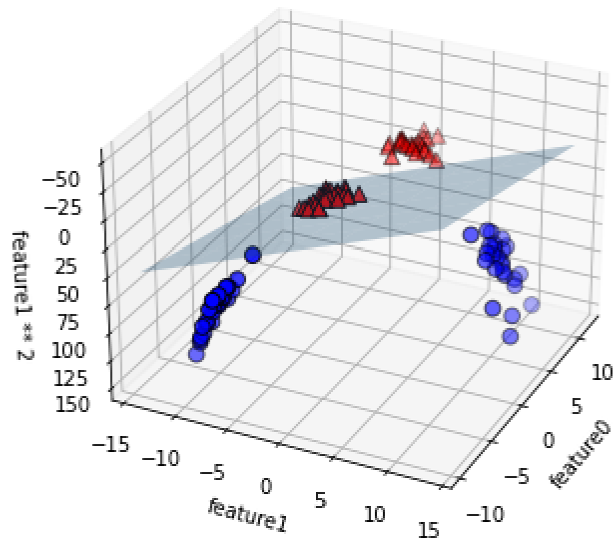
Out[30]:
```
Text(0.5, 0, 'feature1 ** 2')
```



In [31]:
```
linear_svm_3d = LinearSVC().fit(X_new, y)
coef, intercept = linear_svm_3d.coef_.ravel(), linear_svm_3d.intercept_

# show linear decision boundary
figure = plt.figure()
ax = Axes3D(figure, elev=-152, azim=-26)
xx = np.linspace(X_new[:, 0].min() - 2, X_new[:, 0].max() + 2, 50)
yy = np.linspace(X_new[:, 1].min() - 2, X_new[:, 1].max() + 2, 50)

XX, YY = np.meshgrid(xx, yy)
ZZ = (coef[0] * XX + coef[1] * YY + intercept) / -coef[2]
ax.plot_surface(XX, YY, ZZ, rstride=8, cstride=8, alpha=0.3)
ax.scatter(X_new[mask, 0], X_new[mask, 1], X_new[mask, 2], c='b',
           cmap=mglearn.cm2, s=60, edgecolor='k')
ax.scatter(X_new[~mask, 0], X_new[~mask, 1], X_new[~mask, 2], c='r', marker='^',
           cmap=mglearn.cm2, s=60, edgecolor='k')

ax.set_xlabel("feature0")
ax.set_ylabel("feature1")
ax.set_zlabel("feature1 ** 2")
```

```
C:\Users\jcjcb\anaconda3\lib\site-packages\sklearn\svm\_base.py:1206: ConvergenceWarn
ing: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
C:\Users\jcjcb\AppData\Local\Temp\ipykernel_9764\1615430027.py:6: MatplotlibDeprecati
onWarning: Axes3D(fig) adding itself to the figure is deprecated since 3.4. Pass the
keyword argument auto_add_to_figure=False and use fig.add_axes(ax) to suppress this w
arning. The default value of auto_add_to_figure will change to False in mpl3.5 and Tr
ue values will no longer work in 3.6.  This is consistent with other Axes classes.
  ax = Axes3D(figure, elev=-152, azim=-26)
```

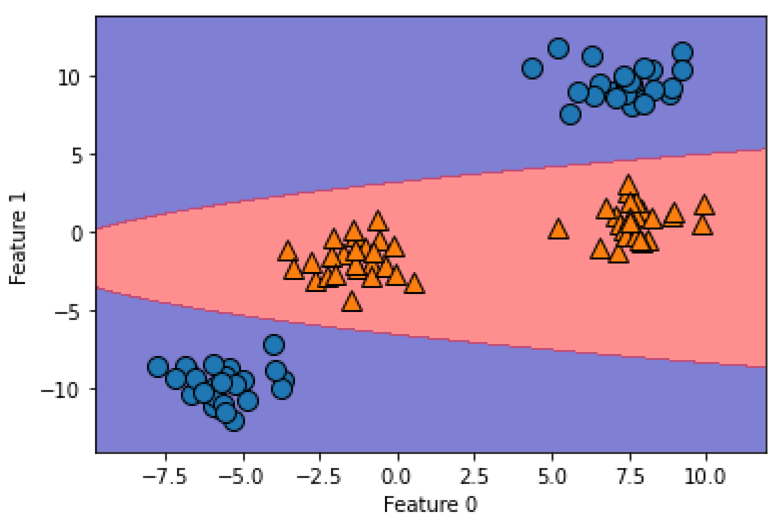
Out[31]:
```
Text(0.5, 0, 'feature1 ** 2')
```



- With the new way of viewing the data you can see that it is now possible to seperate the classes using a linear model using a 3-D model view.

In [32]:
```
ZZ = YY ** 2
dec = linear_svm_3d.decision_function(np.c_[XX.ravel(), YY.ravel(), ZZ.ravel()])
plt.contourf(XX, YY, dec.reshape(XX.shape), levels=[dec.min(), 0, dec.max()],
             cmap=mglearn.cm2, alpha=0.5)
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
plt.xlabel("Feature 0")
plt.ylabel("Feature 1")
```

Out[32]: Text(0, 0.5, 'Feature 1')



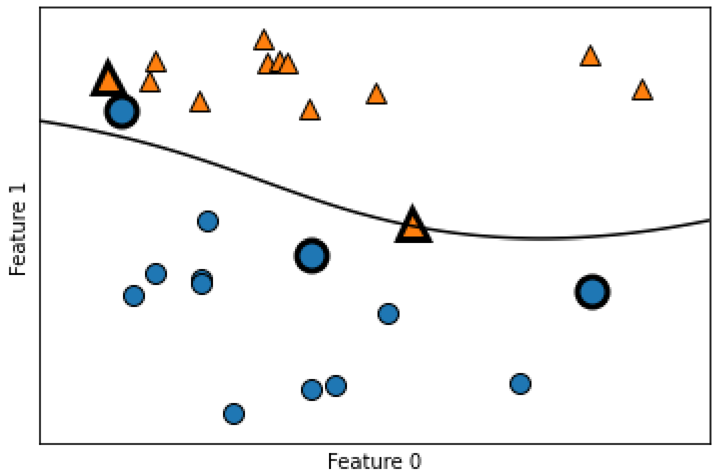
- Because we used a 3-D model the linear model is not a straight line anymore but rather an ellipse that gives us a representation of seperation in the data.

In [33]:
```
from sklearn.svm import SVC
X, y = mglearn.tools.make_handcrafted_dataset()
```

```
svm = SVC(kernel='rbf', C=10, gamma=0.1).fit(X, y)
mglearn.plots.plot_2d_separator(svm, X, eps=.5)
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
# plot support vectors
sv = svm.support_vectors_
# class labels of support vectors are given by the sign of the dual coefficients
sv_labels = svm.dual_coef_.ravel() > 0
mglearn.discrete_scatter(sv[:, 0], sv[:, 1], sv_labels, s=15, markeredgewidth=3)
plt.xlabel("Feature 0")
plt.ylabel("Feature 1")
```

Out[33]:

```
Text(0, 0.5, 'Feature 1')
```



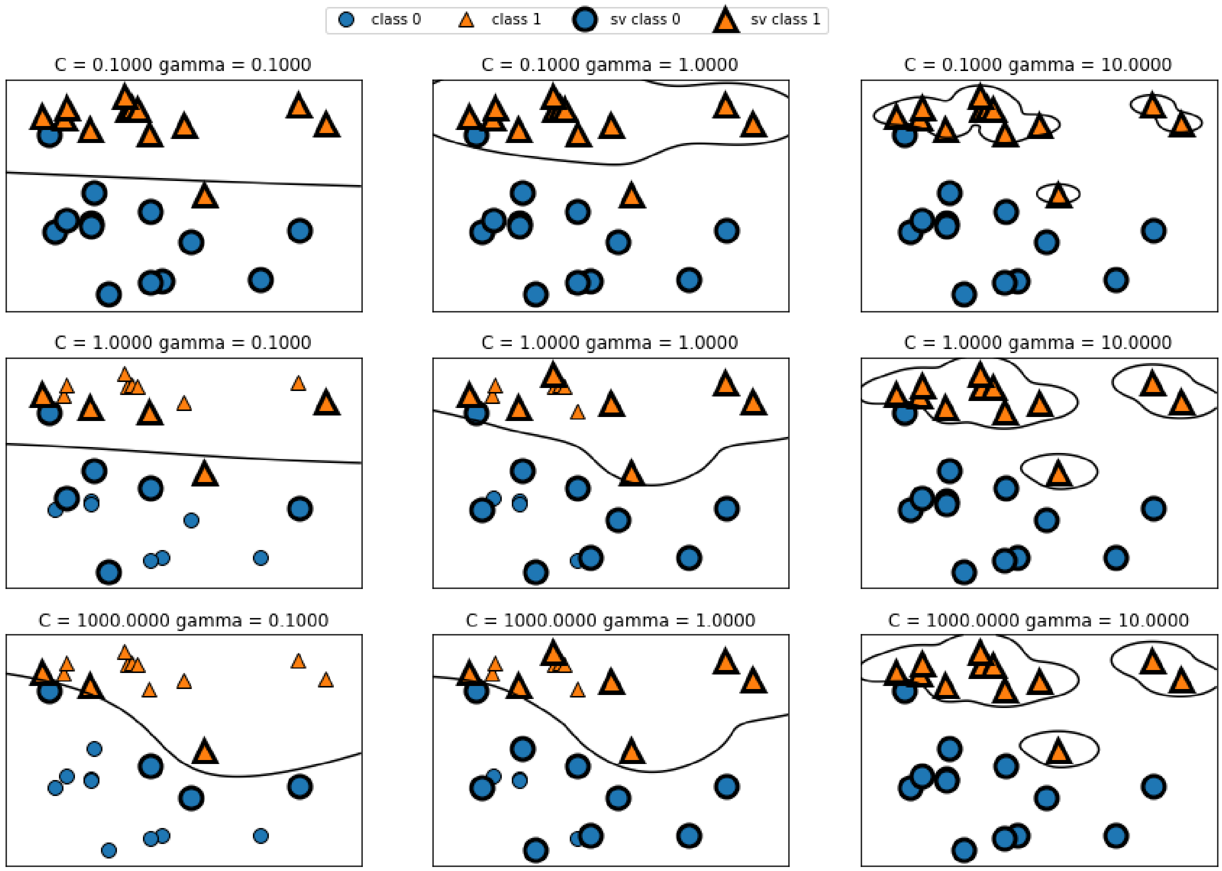
- This plot trains an SVM on the forge dataset that now shows a nonlinear line due to the tuning of the SVM parameters.

In [34]:

```
fig, axes = plt.subplots(3, 3, figsize=(15, 10))

for ax, C in zip(axes, [-1, 0, 3]):
    for a, gamma in zip(ax, range(-1, 2)):
        mglearn.plots.plot_svm(log_C=C, log_gamma=gamma, ax=a)

axes[0, 0].legend(["class 0", "class 1", "sv class 0", "sv class 1"],
                  ncol=4, loc=(.9, 1.2))
```

Out[34]:

```
<matplotlib.legend.Legend at 0x28144320c10>
```



    - This chart shows different outcomes when you adjust the gamma, in turn causing the decision boundary to adjust its complexity. The higher the gamma, the more complex the boundary becomes.

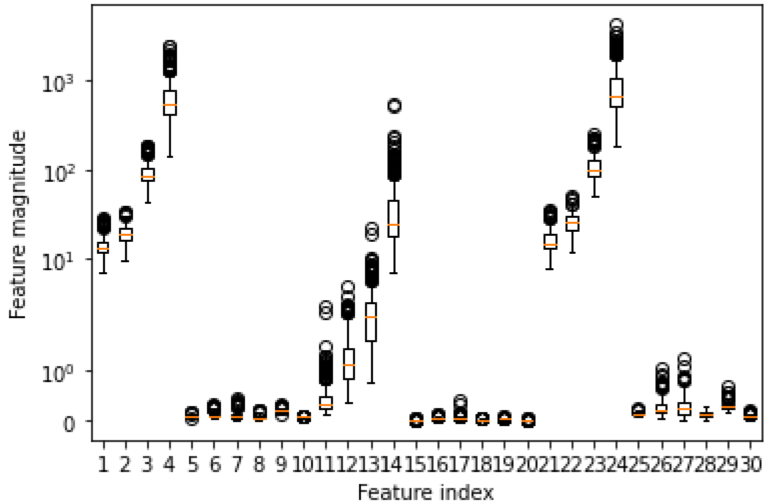
In [38]:
```
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, random_state=0)

svc = SVC()
svc.fit(X_train, y_train)

print("Accuracy on training set: {:.2f}".format(svc.score(X_train, y_train)))
print("Accuracy on test set: {:.2f}".format(svc.score(X_test, y_test)))
```

```
Accuracy on training set: 0.90
Accuracy on test set: 0.94
```

    - This is displaying the accuracy of the training and testing set.

In [41]:
```
plt.boxplot(X_train)
plt.yscale("symlog")
plt.xlabel("Feature index")
plt.ylabel("Feature magnitude")
```

Out[41]:
```
Text(0, 0.5, 'Feature magnitude')
```



-This plot shows the different orders of magnitude for the data set which is not ideal but by rescaling each feature so they are relatively on the same scale you can acheive better results, as shown below.

In [42]:
```
# compute the minimum value per feature on the training set
min_on_training = X_train.min(axis=0)
# compute the range of each feature (max - min) on the training set
range_on_training = (X_train - min_on_training).max(axis=0)

# subtract the min, and divide by range
# afterward, min=0 and max=1 for each feature
X_train_scaled = (X_train - min_on_training) / range_on_training
print("Minimum for each feature\n", X_train_scaled.min(axis=0))
print("Maximum for each feature\n", X_train_scaled.max(axis=0))
```

```
Minimum for each feature
 [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.]
Maximum for each feature
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1.]
```

In [43]:
```
# use THE SAME transformation on the test set,
# using min and range of the training set (see Chapter 3 for details)
X_test_scaled = (X_test - min_on_training) / range_on_training
```

In [44]:
```
svc = SVC()
svc.fit(X_train_scaled, y_train)

print("Accuracy on training set: {:.3f}".format(
        svc.score(X_train_scaled, y_train)))
print("Accuracy on test set: {:.3f}".format(svc.score(X_test_scaled, y_test)))
```

```
Accuracy on training set: 0.984
Accuracy on test set: 0.972
```

In [45]:
```
svc = SVC(C=1000)
svc.fit(X_train_scaled, y_train)

print("Accuracy on training set: {:.3f}".format(
```

```
    svc.score(X_train_scaled, y_train)))
print("Accuracy on test set: {:.3f}".format(svc.score(X_test_scaled, y_test)))
```

```
Accuracy on training set: 1.000
Accuracy on test set: 0.958
```

- As you can see from above, rescalling each feature allowed our accuracy score to improve on our training set, and by increasing "C" we improved it further.