ml-tutorial (/github/Varal7/ml-tutorial/tree/master)
/ Part1.ipynb (/github/Varal7/ml-tutorial/tree/master/Part1.ipynb)

6.86x - Introduction to ML Packages (Part 1)

This tutorial is designed to provide a bird's eye view of the ML packages landscape. The goal is not to give an in-depth explanation of all the features of each packages, but rather demonstrate the purpose of a few widely used ML packages. For more details, we refer the reader to the packages' documentation and other online tutorials.

You can go through the Jupyter, Numpy and Matplotlib sections before the course starts, and then start studying the next sections after you have completed unit 1.

https://github.com/varal7/ml-tutorial (https://github.com/varal7/ml-tutorial)

Jupyter

Jupyter (https://jupyter.org) is not strictly speaking an ML package. It provides a browser front-end connected to an instance of IPython which allows REPL for quick testing, allows to create documents that intertwines code, output, images, and text. This is great for prototyping, demonstrations and tutorials, but terrible for actual coding.

In [1]:
```
6*7
```
Out[1]:
```
42
```

In [2]:
```
def tokenize(text):
    return text.split(" ")
```

In [3]:
```
text = "In a shocking finding, scientist discovered a herd of unicorns
print(tokenize(text))
```
```
['In', 'a', 'shocking', 'finding,', 'scientist', 'discovered', 'a', 'he
```

Numpy

In [4]:
```
import numpy as np
```

Numpy (https://www.numpy.org) is desiged to handle large multidimensional arrays and enable efficient computations with them. In the back, it runs pre-compiled C code which is much faster than, say, a Python `for` loop

In the Numpy tutorial, we have covered the basics of Numpy, numpy arrays, element-wise operations, matrices operations and generating random matrices. In this section, we'll cover indexing, slicing and broadcasting, which are useful concepts that will be reused in `Pandas` and `PyTorch` .

Indexing and slicing

Numpy arrays can be indexed and sliced like regular python arrays

In [5]:
```
a_py = [1, 2, 3, 4, 5, 6, 7, 8, 9]
a_np = np.array(a_py)
```

In [6]:
```
print(a_py[3:7:2], a_np[3:7:2])
print(a_py[2:-1:2], a_np[2:-1:2])
print(a_py[::-1], a_np[::-1])
```

```
[4, 6] [4 6]
[3, 5, 7] [3 5 7]
[9, 8, 7, 6, 5, 4, 3, 2, 1] [9 8 7 6 5 4 3 2 1]
```

But you can also use arrays to index other arrays

In [7]:
```
idx = np.array([7,2])
a_np[idx]
```

Out[7]:
```
array([8, 3])
```

In [8]:
```
# a_py[idx]
```

Which allows convenient querying, reindexing and even sorting

In [9]:
```
ages = np.random.randint(low=30, high=60, size=10)
heights = np.random.randint(low=150, high=210, size=10)

print(ages)
print(heights)
```

```
[33 54 32 43 52 55 49 51 40 40]
[151 185 178 186 188 186 188 199 151 192]
```

In [10]:
```
print(ages < 50)
```

```
[ True False  True  True False False  True False  True  True]
```

In [11]:
```
print(heights[ages < 50])
print(ages[ages < 50])
```

```
[151 178 186 188 151 192]
[33 32 43 49 40 40]
```

In [12]:
```
shuffled_idx = np.random.permutation(10)
print(shuffled_idx)
print(ages[shuffled_idx])
print(heights[shuffled_idx])
```

```
[3 5 7 6 0 2 9 4 8 1]
[43 55 51 49 33 32 40 52 40 54]
[186 186 199 188 151 178 192 188 151 185]
```

In [13]:
```
sorted_idx = np.argsort(ages)
print(sorted_idx)
print(ages[sorted_idx])
print(heights[sorted_idx])
```

```
[2 0 8 9 3 6 7 4 1 5]
[32 33 40 40 43 49 51 52 54 55]
[178 151 151 192 186 188 199 188 185 186]
```

Broadcasting

When Numpy is asked to perform an operation between arrays of differents sizes, it "broadcasts" the smaller one to the bigger one.

In [14]:
```
a = np.array([4, 5, 6])
b = np.array([2, 2, 2])
a * b
```

Out[14]:
```
array([ 8, 10, 12])
```

In [15]:
```
a = np.array([4, 5, 6])
b = 2
a * b
```

Out[15]:
```
array([ 8, 10, 12])
```

The two snippets of code above are equivalent but the second is easier to read and also more efficient.

In [16]:
```
a = np.arange(10).reshape(1,10)
b = np.arange(12).reshape(12,1)
```

In [17]:
```
print(a)
print(b)
```

```
[[0 1 2 3 4 5 6 7 8 9]]
[[ 0]
 [ 1]
 [ 2]
 [ 3]
 [ 4]
 [ 5]
 [ 6]
 [ 7]
 [ 8]
 [ 9]
 [10]
 [11]]
```

In [18]:
```
print(a * b)
```

```
[[ 0  0  0  0  0  0  0  0  0  0]
 [ 0  1  2  3  4  5  6  7  8  9]
 [ 0  2  4  6  8 10 12 14 16 18]
 [ 0  3  6  9 12 15 18 21 24 27]
 [ 0  4  8 12 16 20 24 28 32 36]
 [ 0  5 10 15 20 25 30 35 40 45]
 [ 0  6 12 18 24 30 36 42 48 54]
 [ 0  7 14 21 28 35 42 49 56 63]
 [ 0  8 16 24 32 40 48 56 64 72]
 [ 0  9 18 27 36 45 54 63 72 81]
 [ 0 10 20 30 40 50 60 70 80 90]
 [ 0 11 22 33 44 55 66 77 88 99]]
```

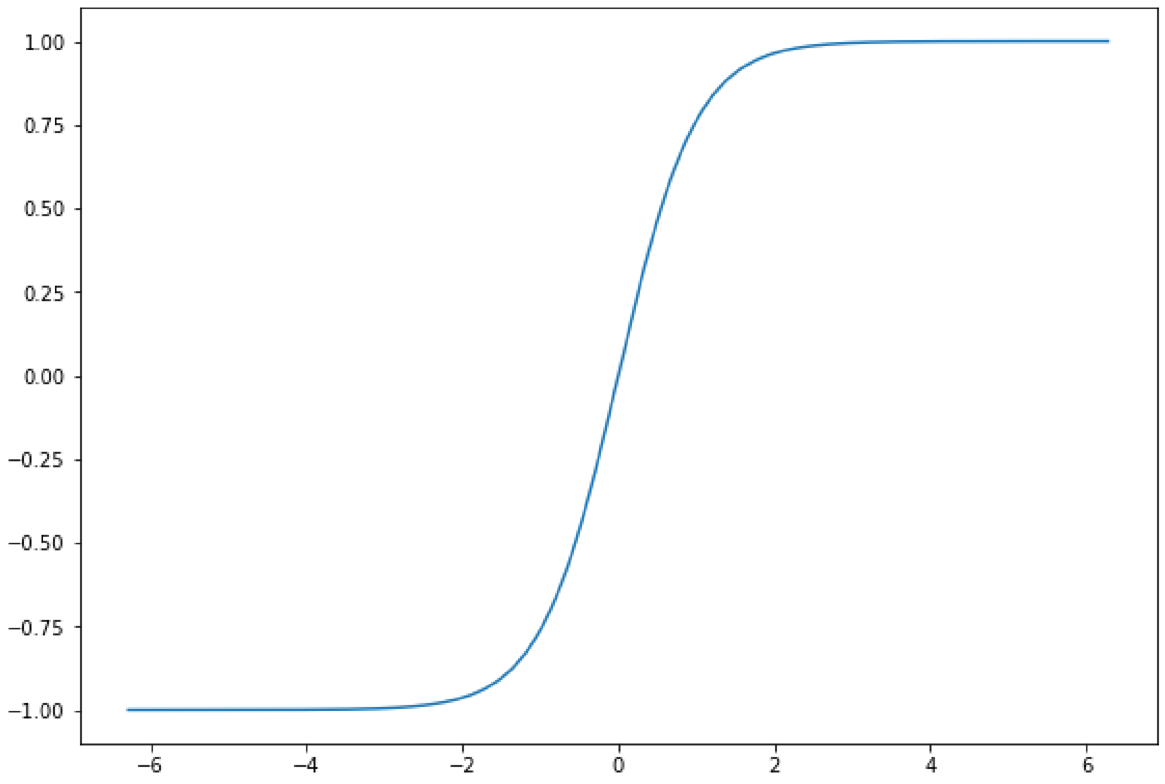
Matplotlib

In [19]:
```
%matplotlib inline
import matplotlib.pyplot as plt
```

In [20]:
```
plt.rcParams['figure.figsize'] = [10, 7]
```

Matplotlib (https://matplotlib.org) is the go-to library to produce plots with Python. It comes with two APIs: a MATLAB-like that a lot of people have learned to use and love, and an object-oriented API that we recommend using.

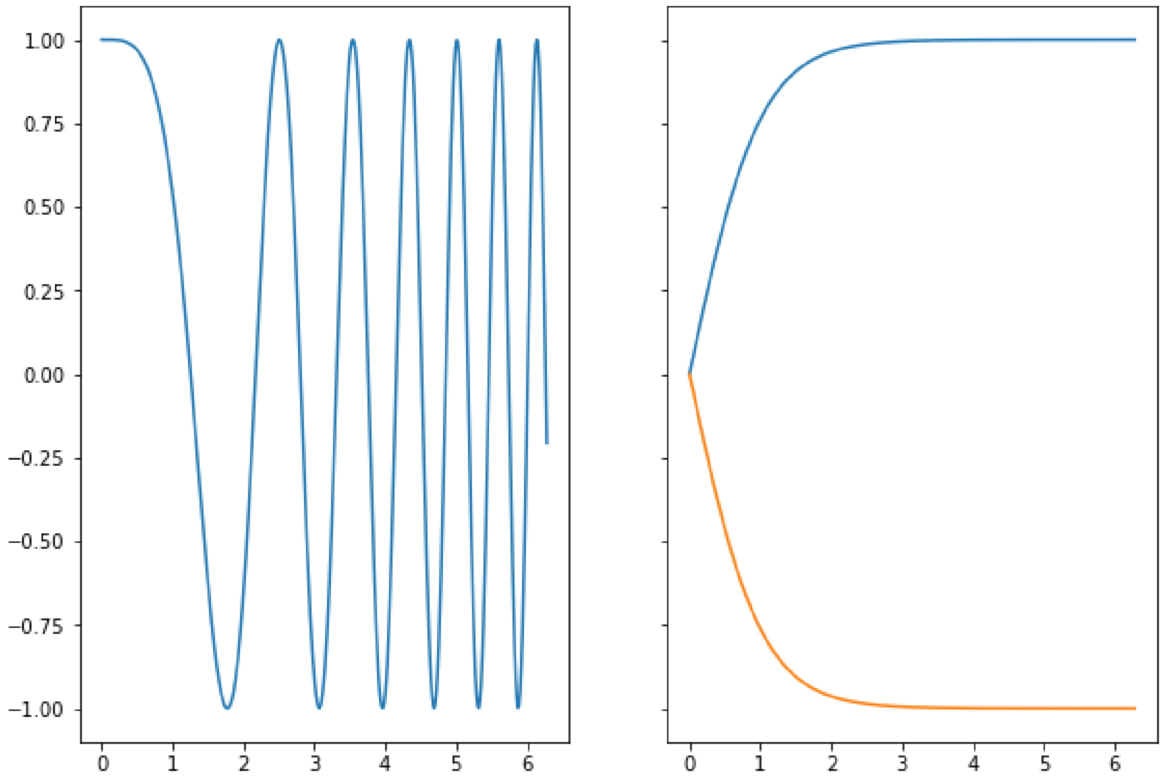
```
In [21]:  x = np.linspace(-2*np.pi, 2*np.pi, 400)
          y = np.tanh(x)
          fig, ax = plt.subplots()
          ax.plot(x, y)

Out[21]:  [<matplotlib.lines.Line2D at 0x115dcc160>]
```



You can plot multiple subplots in the same figure, or multiple functions in the same subplot

In [22]:
```
x = np.linspace(0, 2*np.pi, 400)
y1 = np.tanh(x)
y2 = np.cos(x**2)
fig, axes = plt.subplots(1, 2, sharey=True)
axes[1].plot(x, y1)
axes[1].plot(x, -y1)
axes[0].plot(x, y2)
```

Out[22]:
```
[<matplotlib.lines.Line2D at 0x11813fef0>]
```



Matplotlib also comes with a lot of different options to customize, the colors, the labels, the axes, etc.

For instance, see this introduction to matplotlib (https://nbviewer.jupyter.org/github/jrjohansson/scientific-python-lectures/blob/master/Lecture-4-Matplotlib.ipynb)

Scikit-learn (read this after you have completed unit 1)

Scikit-learn (https://scikit-learn.org/) includes a number of features and utilities to kickstart your journey in Machine Learning.

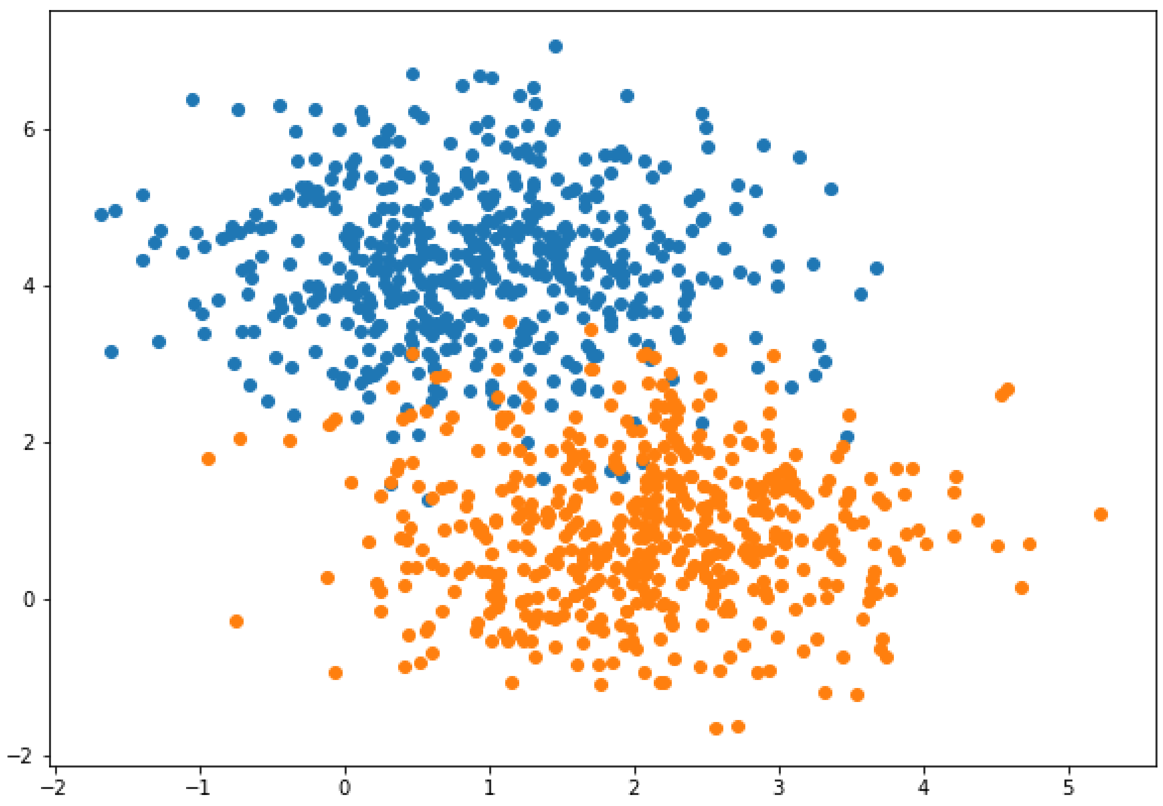
A toy example

In [23]:
```
from sklearn.datasets import make_blobs
```

In [24]:
```
X, y = make_blobs(n_samples=1000, centers=2, random_state=0)
X[:5], y[:5]
```

Out[24]:
```
(array([[0.4666179 , 3.86571303],
        [2.84382807, 3.32650945],
        [0.61121486, 2.51245978],
        [3.81653365, 1.65175932],
        [1.28097244, 0.62827388]]), array([0, 0, 0, 1, 1]))
```

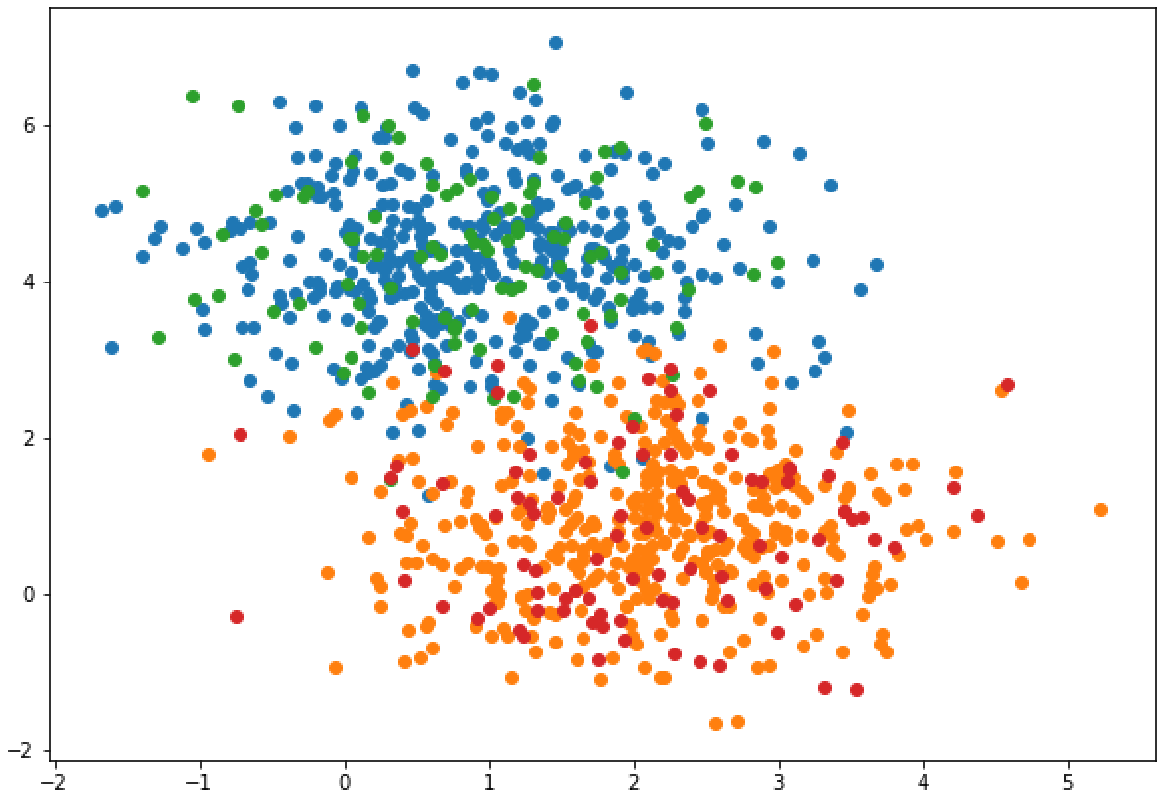
In [25]:
```
fig, ax = plt.subplots()
for label in [0, 1]:
    mask = (y == label)
    ax.scatter(X[mask, 0], X[mask, 1])
```



In [26]:
```
from sklearn.model_selection import train_test_split
```

In [27]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2
```

In [28]:
```
fig, ax = plt.subplots()
for label in [0, 1]:
    mask = (y_train == label)
    ax.scatter(X_train[mask, 0], X_train[mask, 1])
for label in [0, 1]:
    mask = (y_test == label)
    ax.scatter(X_test[mask, 0], X_test[mask, 1])
```



Sklearn uses a uniform and very consistent API, making it easy to switch algorithms

For instance, training and predicting with a perceptron.

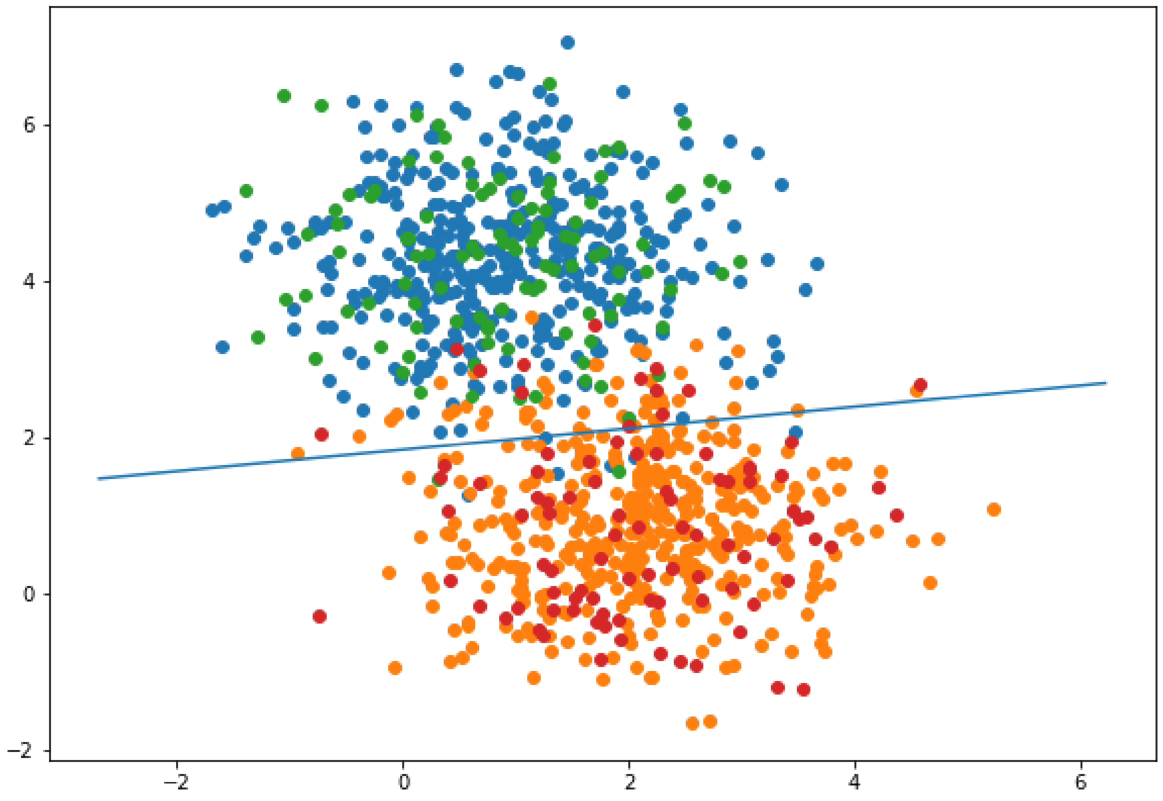
In [29]:
```
from sklearn.linear_model import Perceptron
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score
```

In [30]:
```
clf = Perceptron(max_iter=40, random_state=0)
# clf = LinearSVC(max_iter=40, random_state=0)
```

```
In [31]: clf.fit(X_train, y_train)
         y_pred = clf.predict(X_test)

         print('Test accuracy: %.4f' % accuracy_score(y_test, y_pred))

         Test accuracy: 0.9250
```

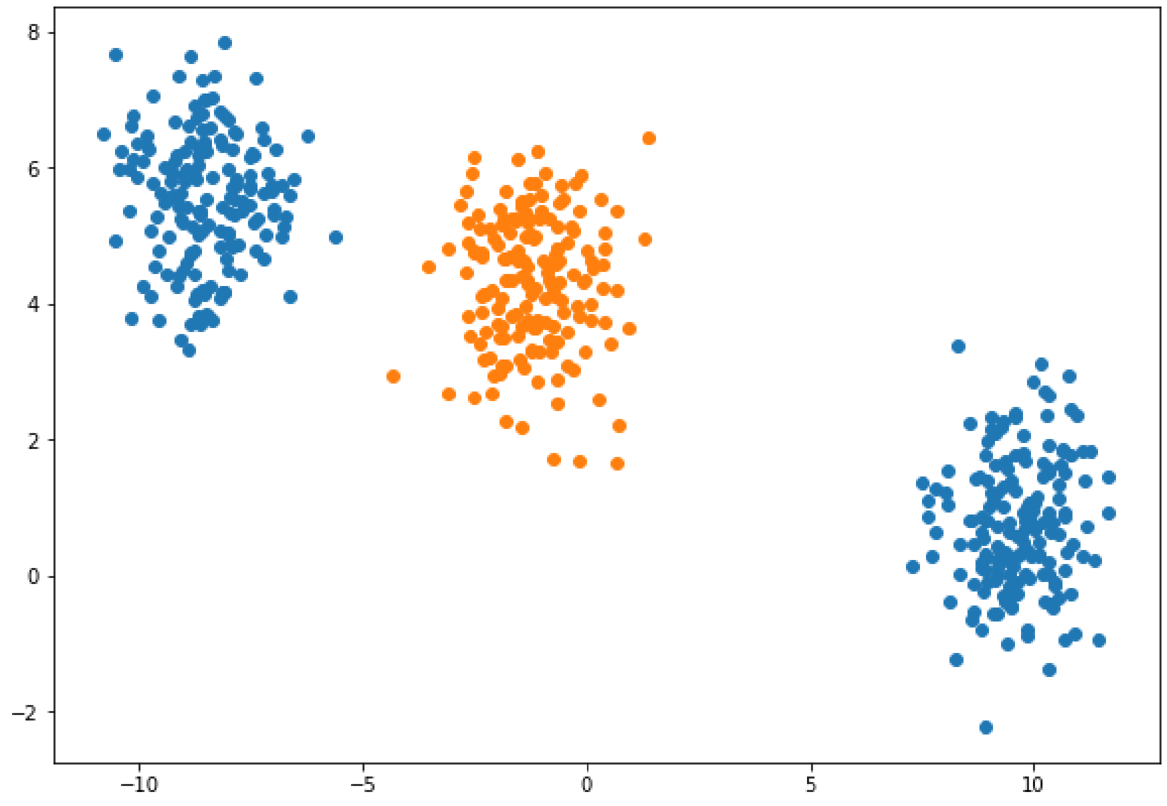
```
In [32]: theta = clf.coef_[0]
         theta_0 = clf.intercept_
```

```
In [33]: fig, ax = plt.subplots()
         for label in [0, 1]:
             mask = (y_train == label)
             ax.scatter(X_train[mask, 0], X_train[mask, 1])
         for label in [0, 1]:
             mask = (y_test == label)
             ax.scatter(X_test[mask, 0], X_test[mask, 1])
         x_bnd = np.linspace(X[:, 0].min() - 1, X[:, 0].max() + 1,  400)
         y_bnd = - x_bnd * (theta[0] /theta[1]) - (theta_0 / theta[1])
         ax.plot(x_bnd, y_bnd)

Out[33]: [<matplotlib.lines.Line2D at 0x11acccba8>]
```



Another toy example

In [34]:
```
X, y = make_blobs(n_samples=500, centers=3, random_state=7)
y[y==2] = 0
fig, ax = plt.subplots()
for label in [0, 1]:
    mask = (y == label)
    ax.scatter(X[mask, 0], X[mask, 1])
```



In [35]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
```

In [36]:
```
from sklearn.svm import SVC
# clf = SVC(kernel="linear", random_state=0)
clf = SVC(kernel="rbf", random_state=0)
clf.fit(X_train, y_train)
```

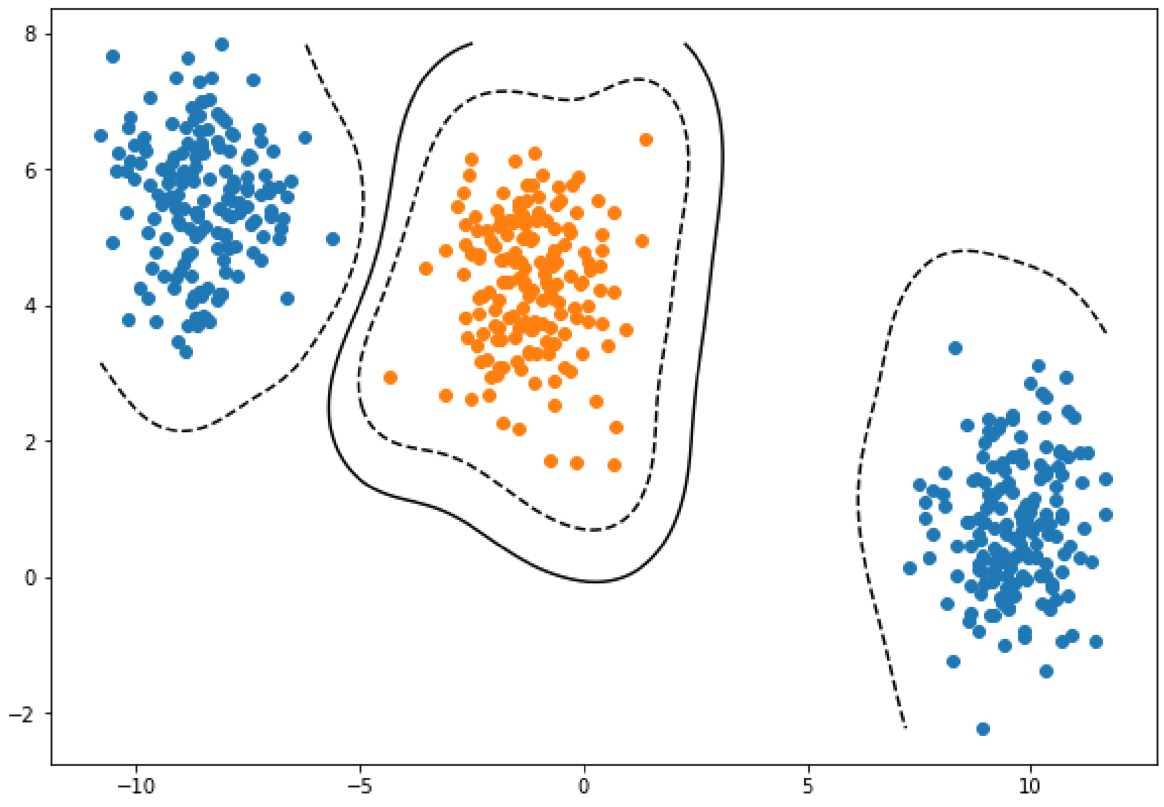
Out[36]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=0, shrinking=True,
  tol=0.001, verbose=False)
```

In [37]:
```
y_pred = clf.predict(X_test)

print('Test accuracy: %.4f' % accuracy_score(y_test, y_pred))
```

```
Test accuracy: 1.0000
```

In [38]:
```python
x_min = X[:, 0].min()
x_max = X[:, 0].max()
y_min = X[:, 1].min()
y_max = X[:, 1].max()

XX, YY = np.mgrid[x_min:x_max:200j, y_min:y_max:200j]
Z = clf.decision_function(np.c_[XX.ravel(), YY.ravel()])

fig, ax = plt.subplots()
for label in [0, 1]:
    mask = (y == label)
    ax.scatter(X[mask, 0], X[mask, 1])

Z = Z.reshape(XX.shape)
ax.contour(XX, YY, Z, colors="black",
    linestyles=['--', '-', '--'], levels=[-.5, 0, .5])
```

Out[38]:
```
<matplotlib.contour.QuadContourSet at 0x11ae7f390>
```



Classify digits

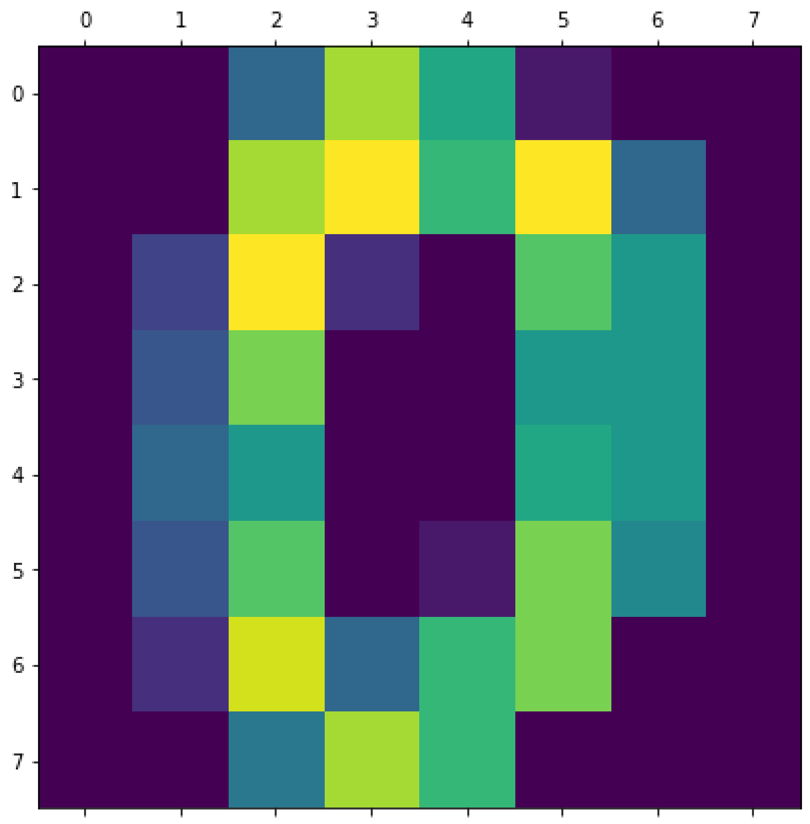
In [39]:
```python
# from sklearn.datasets import load_breast_cancer
# breast_cancer = load_breast_cancer()
# X, y = breast_cancer.data, breast_cancer.target
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=
```

In [40]:
```python
from sklearn.datasets import load_digits
```

In [41]:
```
digits = load_digits()
X, y = digits.data, digits.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2
```

In [42]:
```
fig, ax = plt.subplots()
ax.matshow(digits.images[0])
```

Out[42]:
```
<matplotlib.image.AxesImage at 0x11a634ac8>
```



In [43]:
```
X_train.shape
```

Out[43]:
```
(1437, 64)
```

In [44]:
```
clf = Perceptron(max_iter=40, random_state=0)
```

In [45]:
```
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
print('Accuracy: %.4f' % accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.9389
```

In [46]:
```
clf = LinearSVC(C=1, random_state=0)
```

In [47]:
```
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
print('Accuracy: %.4f' % accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.9361
```

In [48]:
```
from sklearn.metrics import confusion_matrix
```

In [49]:
```
confusion_matrix(y_test, clf.predict(X_test))
```

Out[49]:
```
array([[27,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0, 29,  0,  0,  0,  0,  1,  0,  5,  0],
       [ 0,  1, 33,  2,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0, 29,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  0, 30,  0,  0,  0,  0,  0],
       [ 0,  1,  0,  0,  0, 38,  1,  0,  0,  0],
       [ 0,  1,  0,  0,  0,  0, 43,  0,  0,  0],
       [ 0,  2,  0,  0,  0,  0,  0, 37,  0,  0],
       [ 0,  2,  1,  0,  0,  0,  0,  0, 35,  1],
       [ 0,  0,  0,  2,  0,  1,  0,  0,  2, 36]])
```

Scikit-learn also includes utilities to quickly compute a cross validation score...

In [50]:
```
clf = LinearSVC(C=1, random_state=0)
from sklearn.model_selection import cross_val_score
scores =  cross_val_score(clf, X_train, y_train, cv=5)
print("Mean: %.4f, Std: %.4f" % (np.mean(scores), np.std(scores)))
```

```
Mean: 0.9443, Std: 0.0127
```

In [51]:
```
clf = LinearSVC(C=0.1, random_state=0)
scores =  cross_val_score(clf, X_train, y_train, cv=5)
print("Mean: %.4f, Std: %.4f" % (np.mean(scores), np.std(scores)))
```

```
Mean: 0.9555, Std: 0.0101
```

... or to perform a grid search

In [52]:
```
from sklearn.model_selection import GridSearchCV
```

In [53]:
```
clf = LinearSVC(random_state=0)
param_grid = {'C': 10. ** np.arange(-6, 4)}
grid_search = GridSearchCV(clf, param_grid=param_grid, cv=5, verbose=3,
```

In [54]:
```
grid_search.fit(X_train, y_train);
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[CV] C=1e-06 ..........................................................
[CV] ................ C=1e-06, score=0.8424657534246576, total=   0.0s
[CV] C=1e-06 ..........................................................
[CV] ................ C=1e-06, score=0.8719723183391004, total=   0.0s
[CV] C=1e-06 ..........................................................
[CV] ................ C=1e-06, score=0.9090909090909091, total=   0.0s
[CV] C=1e-06 ..........................................................
[CV] ................ C=1e-06, score=0.8426573426573427, total=   0.0s
[CV] C=1e-06 ..........................................................
[CV] ................. C=1e-06, score=0.897887323943662, total=   0.0s
[CV] C=1e-05 ..........................................................
[CV] ................ C=1e-05, score=0.9075342465753424, total=   0.0s
[CV] C=1e-05 ..........................................................
[CV] ................ C=1e-05, score=0.9134948096885813, total=   0.0s
[CV] C=1e-05 ..........................................................
[CV] ................ C=1e-05, score=0.9440559440559441, total=   0.0s
[CV] C=1e-05 ..........................................................
[CV] ................ C=1e-05, score=0.9020979020979021, total=   0.0s
[CV] C=1e-05 ..........................................................
[CV] ................ C=1e-05, score=0.9330985915492958, total=   0.0s
[CV] C=0.0001 .........................................................
[CV] ................ C=0.0001, score=0.934931506849315, total=   0.0s
[CV] C=0.0001 .........................................................
[CV] ............... C=0.0001, score=0.9411764705882353, total=   0.0s
[CV] C=0.0001 .........................................................
[CV] ................ C=0.0001, score=0.965034965034965, total=   0.0s
[CV] C=0.0001 .........................................................
[CV] ............... C=0.0001, score=0.9405594405594405, total=   0.0s
[CV] C=0.0001 .........................................................
[CV] ............... C=0.0001, score=0.9683098591549296, total=   0.0s
[CV] C=0.001 ..........................................................

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s remaining

[CV] ................. C=0.001, score=0.958904109589041, total=   0.0s
[CV] C=0.001 ..........................................................
[CV] ................ C=0.001, score=0.9653979238754326, total=   0.0s
[CV] C=0.001 ..........................................................
[CV] ................. C=0.001, score=0.965034965034965, total=   0.0s
[CV] C=0.001 ..........................................................
[CV] ................. C=0.001, score=0.965034965034965, total=   0.0s
[CV] C=0.001 ..........................................................
[CV] ................ C=0.001, score=0.9788732394366197, total=   0.0s
[CV] C=0.01 ...........................................................
[CV] ................. C=0.01, score=0.9554794520547946, total=   0.1s
[CV] C=0.01 ...........................................................
[CV] ................. C=0.01, score=0.9619377162629758, total=   0.1s
[CV] C=0.01 ...........................................................
[CV] ................. C=0.01, score=0.9685314685314685, total=   0.1s
[CV] C=0.01 ...........................................................
[CV] ................. C=0.01, score=0.9615384615384616, total=   0.1s
[CV] C=0.01 ...........................................................
[CV] .................. C=0.01, score=0.971830985915493, total=   0.1s
```

```
[CV] C=0.1 ...........................................................
[CV] .................. C=0.1, score=0.9452054794520548, total=   0.1s
[CV] C=0.1 ...........................................................
[CV] ................... C=0.1, score=0.972318339100346, total=   0.1s
[CV] C=0.1 ...........................................................
[CV] .................. C=0.1, score=0.9475524475524476, total=   0.1s
[CV] C=0.1 ...........................................................
[CV] .................. C=0.1, score=0.9615384615384616, total=   0.1s
[CV] C=0.1 ...........................................................
[CV] .................. C=0.1, score=0.9507042253521126, total=   0.1s
[CV] C=1.0 ...........................................................
[CV] .................. C=1.0, score=0.9417808219178082, total=   0.1s
[CV] C=1.0 ...........................................................
[CV] .................. C=1.0, score=0.9619377162629758, total=   0.1s
[CV] C=1.0 ...........................................................
[CV] .................. C=1.0, score=0.9370629370629371, total=   0.1s
[CV] C=1.0 ...........................................................
[CV] .................. C=1.0, score=0.9545454545454546, total=   0.1s
[CV] C=1.0 ...........................................................
[CV] ................... C=1.0, score=0.926056338028169, total=   0.1s
[CV] C=10.0 ..........................................................
[CV] ................. C=10.0, score=0.9383561643835616, total=   0.1s
[CV] C=10.0 ..........................................................
[CV] ................. C=10.0, score=0.9653979238754326, total=   0.1s
[CV] C=10.0 ..........................................................
[CV] ................. C=10.0, score=0.9335664335664335, total=   0.1s
[CV] C=10.0 ..........................................................
[CV] .................. C=10.0, score=0.951048951048951, total=   0.1s
[CV] C=10.0 ..........................................................
[CV] .................. C=10.0, score=0.954225352112676, total=   0.1s
[CV] C=100.0 .........................................................
[CV] ................. C=100.0, score=0.928082191780822, total=   0.1s
[CV] C=100.0 .........................................................
[CV] ................ C=100.0, score=0.9446366782006921, total=   0.1s
[CV] C=100.0 .........................................................
[CV] ................ C=100.0, score=0.9335664335664335, total=   0.1s
[CV] C=100.0 .........................................................
[CV] ................ C=100.0, score=0.9475524475524476, total=   0.1s
[CV] C=100.0 .........................................................
[CV] ................ C=100.0, score=0.9401408450704225, total=   0.1s
[CV] C=1000.0 ........................................................
[CV] ............... C=1000.0, score=0.9383561643835616, total=   0.1s
[CV] C=1000.0 ........................................................
[CV] ............... C=1000.0, score=0.9550173010380623, total=   0.1s
[CV] C=1000.0 ........................................................
[CV] ............... C=1000.0, score=0.9300699300699301, total=   0.1s
[CV] C=1000.0 ........................................................
[CV] ............... C=1000.0, score=0.9545454545454546, total=   0.1s
[CV] C=1000.0 ........................................................
[CV] ............... C=1000.0, score=0.9436619718309859, total=   0.1s

[Parallel(n_jobs=1)]: Done  50 out of  50 | elapsed:    2.5s finished
```

In [55]:

```
print(grid_search.best_params_)
```

```
{'C': 0.001}
```

In [56]:
```
print(grid_search.best_score_)
```

```
0.9665970772442589
```

In [57]:
```
y_pred = grid_search.predict(X_test)
print('Accuracy: %.4f' % accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.9639
```

And a lot more features! We have only looked at some classification models and some model selection features, but sklearn can also be used for regression,

Pandas

In [58]:
```
import pandas as pd
```

Pandas (https://pandas.pydata.org) is a library that provides a set of tools for data analysis (Python Data Analysis Library).

Pandas dataframes can be created by importing a CSV file (or TSV, or JSON, or SQL, etc.)

In [59]:
```
# df = pd.read_csv("file.csv")
```

Pandas dataframes can also be created directly from a dictionary of arrays.

In [60]:
```
print(grid_search.cv_results_)
```

```
{'mean_fit_time': array([0.01142702, 0.01115093, 0.01050925, 0.0205419
       0.06751108, 0.06907425, 0.06755571, 0.07172804, 0.07606635]), '
       0.00261869, 0.00117984, 0.00326013, 0.00573768, 0.0030746 ]), '
       0.00030074, 0.00031796, 0.00030222, 0.00032935, 0.00035524]), '
       4.90331443e-06, 1.30740366e-05, 3.27037297e-05, 1.36716789e-05,
       3.10957060e-05, 4.44746199e-05]), 'param_C': masked_array(data=
                   100.0, 1000.0],
             mask=[False, False, False, False, False, False, False, Fa
                   False, False],
       fill_value='?',
            dtype=object), 'params': [{'C': 1e-06}, {'C': 1e-05}, {'C'
       0.94520548, 0.94178082, 0.93835616, 0.92808219, 0.93835616]), '
       0.97231834, 0.96193772, 0.96539792, 0.94463668, 0.9550173 ]), '
       0.94755245, 0.93706294, 0.93356643, 0.93356643, 0.93006993]), '
       0.96153846, 0.95454545, 0.95104895, 0.94755245, 0.95454545]), '
       0.95070423, 0.92605634, 0.95422535, 0.94014085, 0.94366197]), '
       0.95546277, 0.94432846, 0.94850383, 0.93876131, 0.94432846]), '
       0.0101387 , 0.0126865 , 0.01142328, 0.00715855, 0.00956827]), '
       0.99825328, 0.99737991, 0.99475983, 0.99563319, 0.99650655]), '
       0.99651568, 0.99738676, 0.99216028, 0.98954704, 0.98432056]), '
       0.99826238, 0.99739357, 0.99478714, 0.99565595, 0.99565595]), '
       0.99652476, 0.99565595, 0.9913119 , 0.98783666, 0.99391833]), '
       0.99566349, 0.98091934, 0.99045967, 0.99132697, 0.98785776]), '
       0.99704392, 0.99374711, 0.99269576, 0.99199996, 0.99165183]), '
       0.00103936, 0.00644882, 0.00177967, 0.00317394, 0.00475144])}
```

In [61]:

```
df = pd.DataFrame(grid_search.cv_results_)
df
```

Out[61]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	params	split0_test_
0	0.011427	0.000539	0.000306	0.000049	1e-06	{'C': 1e-06}	0.84
1	0.011151	0.000381	0.000299	0.000110	1e-05	{'C': 1e-05}	0.90
2	0.010509	0.000301	0.000258	0.000038	0.0001	{'C': 0.0001}	0.93
3	0.020542	0.000409	0.000291	0.000038	0.001	{'C': 0.001}	0.95
4	0.071609	0.002812	0.000298	0.000005	0.01	{'C': 0.01}	0.95
5	0.067511	0.002619	0.000301	0.000013	0.1	{'C': 0.1}	0.94
6	0.069074	0.001180	0.000318	0.000033	1	{'C': 1.0}	0.94
7	0.067556	0.003260	0.000302	0.000014	10	{'C': 10.0}	0.93
8	0.071728	0.005738	0.000329	0.000031	100	{'C': 100.0}	0.92
9	0.076066	0.003075	0.000355	0.000044	1000	{'C': 1000.0}	0.93

10 rows × 21 columns

Pandas columns are also Numpy arrays, so they obey to the same indexing magic

In [62]:

```
df[df['param_C'] < 0.01]
```

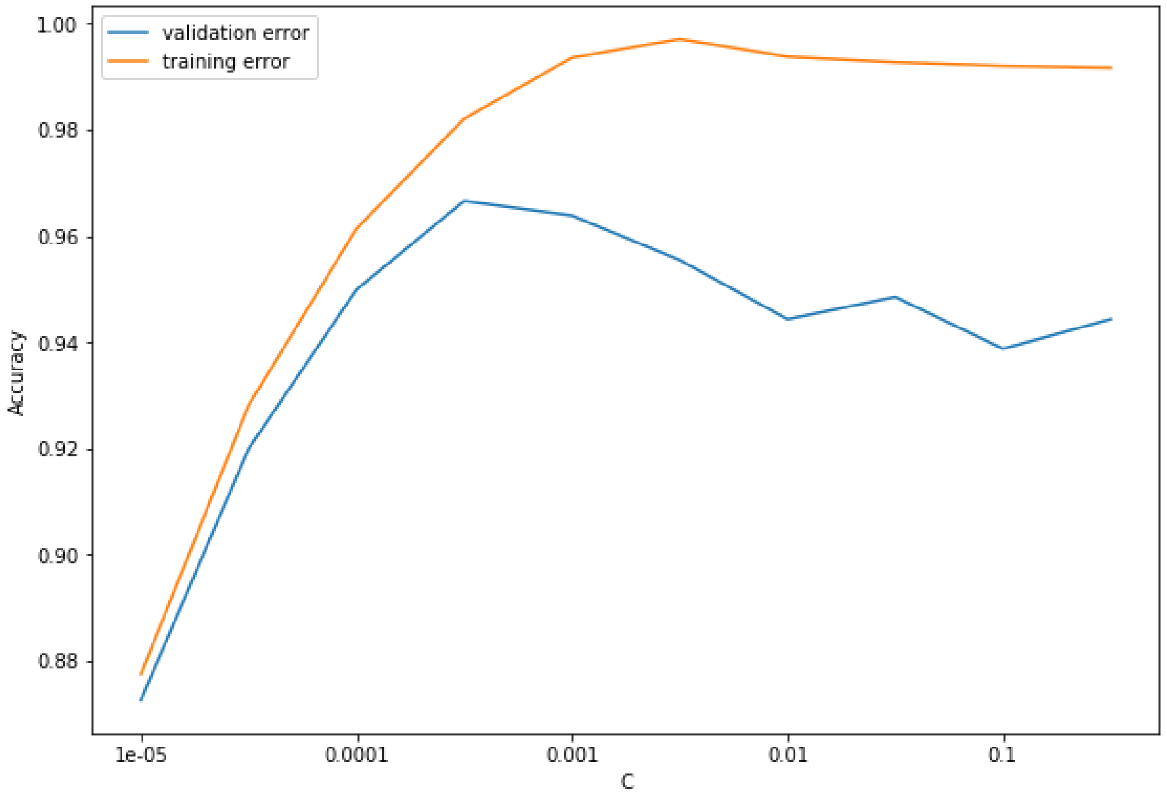
Out[62]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	params	split0_test_
0	0.011427	0.000539	0.000306	0.000049	1e-06	{'C': 1e-06}	0.84
1	0.011151	0.000381	0.000299	0.000110	1e-05	{'C': 1e-05}	0.90
2	0.010509	0.000301	0.000258	0.000038	0.0001	{'C': 0.0001}	0.93
3	0.020542	0.000409	0.000291	0.000038	0.001	{'C': 0.001}	0.95

4 rows × 21 columns

They also provide most functionality you would expect as database user (`df.sort_values` , `df.groupby` , `df.join` , `df.concat` , etc.)

In [63]:
```
fig, ax = plt.subplots()
ax.plot(df['mean_test_score'], label="validation error")
ax.plot(df['mean_train_score'], label="training error")
ax.set_xticklabels(df['param_C'])
ax.set_xlabel("C")
ax.set_ylabel("Accuracy")
ax.legend(loc='best');
```



Other packages

Other packages that didn't make the cut:

- Plotly (https://plot.ly) and Seaborn (https://seaborn.pydata.org): two other plotting libraries
- Scipy (https://www.scipy.org): a science library built on top of Numpy
- Scrapy (https://www.scipy.org): a web crawling library
- pdb (https://docs.python.org/3/library/pdb.html): a debugger for python (not ML-specific but terribly useful)
- tqdm (https://github.com/tqdm/tqdm): a progress bar (not ML-specific)

Next time:

- Pytorch (https://pytorch.org)