Datasets with PyTorch

In this section we'll show how to:

- load data from outside files
- build random batches using PyTorch's **data** (https://pytorch.org/docs/stable/data.html) utilities

At the end we'll briefly mention **torchvision** (https://pytorch.org/docs/stable/torchvision/index.html).

Perform standard imports

In [1]:

```
import torch
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

Loading data from files

We've seen how to load NumPy arrays into PyTorch, and anyone familiar with `pandas.read_csv()` can use it to prepare data before forming tensors. Here we'll load the iris flower dataset (https://en.wikipedia.org/wiki/Iris_flower_data_set) saved as a .csv file.

In [2]:

```
df = pd.read_csv('../Data/iris.csv')
df.head()
```

Out[2]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0.0
1	4.9	3.0	1.4	0.2	0.0
2	4.7	3.2	1.3	0.2	0.0
3	4.6	3.1	1.5	0.2	0.0
4	5.0	3.6	1.4	0.2	0.0

In [3]:

```
df.shape
```

Out[3]:

```
(150, 5)
```

Plot the data

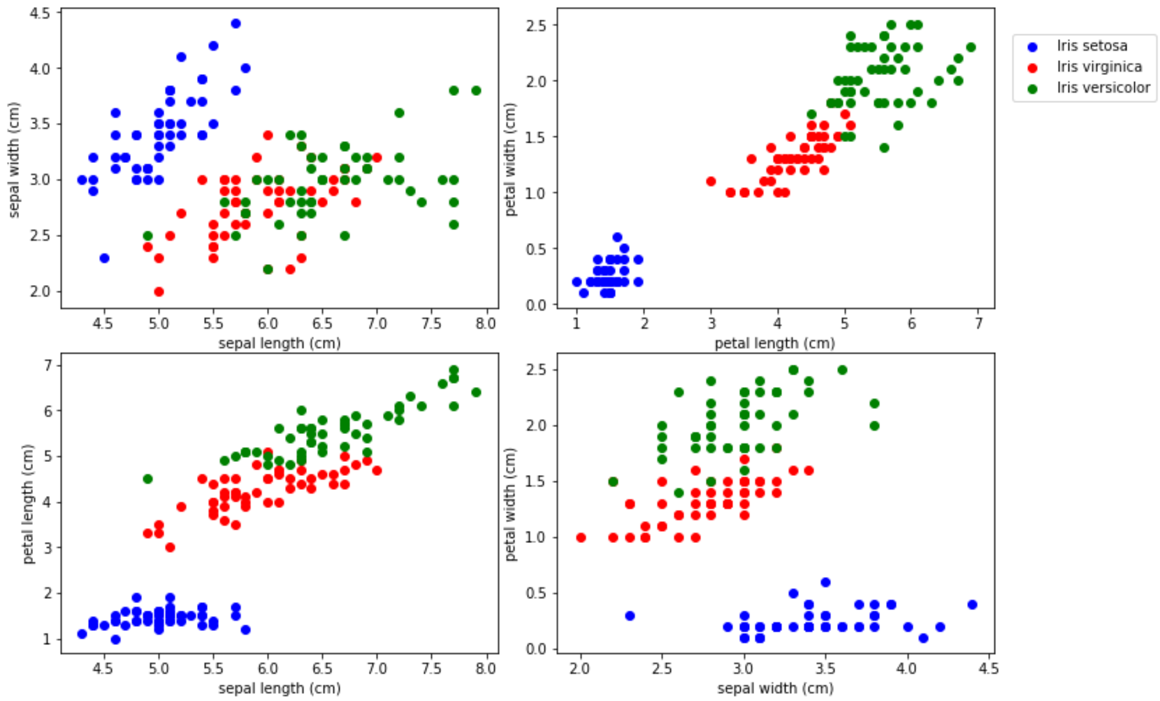
In [4]:

```
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(10,7))
fig.tight_layout()

plots = [(0,1),(2,3),(0,2),(1,3)]
colors = ['b', 'r', 'g']
labels = ['Iris setosa','Iris virginica','Iris versicolor']

for i, ax in enumerate(axes.flat):
    for j in range(3):
        x = df.columns[plots[i][0]]
        y = df.columns[plots[i][1]]
        ax.scatter(df[df['target']==j][x], df[df['target']==j][y], color=colors[j])
        ax.set(xlabel=x, ylabel=y)

fig.legend(labels=labels, loc=3, bbox_to_anchor=(1.0,0.85))
plt.show()
```



The iris dataset consists of 50 samples each from three species of Iris (*Iris setosa*, *Iris virginica* and *Iris versicolor*), for 150 total samples. We have four features (sepal length & width, petal length & width) and three unique labels:

0. *Iris setosa*
1. *Iris virginica*

2. *Iris versicolor*

The classic method for building train/test split tensors

Before introducing PyTorch's Dataset and DataLoader classes, we'll take a quick look at the alternative.

In [5]:

```
from sklearn.model_selection import train_test_split

train_X, test_X, train_y, test_y = train_test_split(df.drop('target',axis=1).values
                                                    df['target'].values, test_size=
                                                    random_state=33)

X_train = torch.FloatTensor(train_X)
X_test = torch.FloatTensor(test_X)
y_train = torch.LongTensor(train_y).reshape(-1, 1)
y_test = torch.LongTensor(test_y).reshape(-1, 1)
```

In [6]:

```
print(f'Training size: {len(y_train)}')
labels, counts = y_train.unique(return_counts=True)
print(f'Labels: {labels}\nCounts: {counts}')
```

```
Training size: 120
Labels: tensor([0, 1, 2])
Counts: tensor([42, 42, 36])
```

NOTE: The importance of a balanced training set is discussed in *A systematic study of the class imbalance problem in convolutional neural networks* by Mateusz Buda, Atsuto Maki, Maciej A. Mazurowski (10/15/17, latest rev 10/13/18) [https://arxiv.org/abs/1710.05381 (https://arxiv.org/abs/1710.05381)](https://arxiv.org/abs/1710.05381)
For example, the authors show that oversampling a less common class so that it matches the more common classes is always the preferred choice.

In [7]:

```
X_train.size()
```

Out[7]:

```
torch.Size([120, 4])
```

In [8]:

```
y_train.size()
```

Out[8]:

```
torch.Size([120, 1])
```

NOTE: It's up to us to remember which columns correspond to which features.

Using PyTorch's Dataset and DataLoader classes

A far better alternative is to leverage PyTorch's **`Dataset`** [(https://pytorch.org/docs/stable/data.html)](https://pytorch.org/docs/stable/data.html) and **`DataLoader`** [(https://pytorch.org/docs/stable/data.html#torch.utils.data.DataLoader)](https://pytorch.org/docs/stable/data.html#torch.utils.data.DataLoader) classes.

Usually, to set up a Dataset specific to our investigation we would define our own custom class that inherits from `torch.utils.data.Dataset` (we'll do this in the CNN section). For now, we can use the built-in **`TensorDataset`** [(https://pytorch.org/docs/stable/data.html#torch.utils.data.TensorDataset)](https://pytorch.org/docs/stable/data.html#torch.utils.data.TensorDataset) class.

In [9]:

```
from torch.utils.data import TensorDataset, DataLoader

data = df.drop('target',axis=1).values
labels = df['target'].values

iris = TensorDataset(torch.FloatTensor(data),torch.LongTensor(labels))
```

In [10]:

```
len(iris)
```

Out[10]:

```
150
```

In [11]:

```
type(iris)
```

Out[11]:

```
torch.utils.data.dataset.TensorDataset
```

In [12]:

```
for i in iris:
    print(i)
```

```
(tensor([5.1000, 3.5000, 1.4000, 0.2000]), tensor(0))
(tensor([4.9000, 3.0000, 1.4000, 0.2000]), tensor(0))
(tensor([4.7000, 3.2000, 1.3000, 0.2000]), tensor(0))
(tensor([4.6000, 3.1000, 1.5000, 0.2000]), tensor(0))
(tensor([5.0000, 3.6000, 1.4000, 0.2000]), tensor(0))
(tensor([5.4000, 3.9000, 1.7000, 0.4000]), tensor(0))
(tensor([4.6000, 3.4000, 1.4000, 0.3000]), tensor(0))
(tensor([5.0000, 3.4000, 1.5000, 0.2000]), tensor(0))
(tensor([4.4000, 2.9000, 1.4000, 0.2000]), tensor(0))
(tensor([4.9000, 3.1000, 1.5000, 0.1000]), tensor(0))
(tensor([5.4000, 3.7000, 1.5000, 0.2000]), tensor(0))
(tensor([4.8000, 3.4000, 1.6000, 0.2000]), tensor(0))
(tensor([4.8000, 3.0000, 1.4000, 0.1000]), tensor(0))
(tensor([4.3000, 3.0000, 1.1000, 0.1000]), tensor(0))
(tensor([5.8000, 4.0000, 1.2000, 0.2000]), tensor(0))
(tensor([5.7000, 4.4000, 1.5000, 0.4000]), tensor(0))
(tensor([5.4000, 3.9000, 1.3000, 0.4000]), tensor(0))
(tensor([5.1000, 3.5000, 1.4000, 0.3000]), tensor(0))
(tensor([5.7000, 3.8000, 1.7000, 0.3000]), tensor(0))
```

Once we have a dataset we can wrap it with a DataLoader. This gives us a powerful sampler that provides single- or multi-process iterators over the dataset.

In [14]:

```
iris_loader = DataLoader(iris, batch_size=105, shuffle=True)
```

In [15]:

```
for i_batch, sample_batched in enumerate(iris_loader):
    print(i_batch, sample_batched)
```

```
0 [tensor([[6.7000, 3.1000, 4.4000, 1.4000],
        [4.8000, 3.4000, 1.6000, 0.2000],
        [4.8000, 3.4000, 1.9000, 0.2000],
        [5.6000, 2.9000, 3.6000, 1.3000],
        [6.7000, 3.3000, 5.7000, 2.1000],
        [6.9000, 3.1000, 4.9000, 1.5000],
        [6.7000, 2.5000, 5.8000, 1.8000],
        [6.4000, 3.1000, 5.5000, 1.8000],
        [6.7000, 3.1000, 5.6000, 2.4000],
        [5.5000, 2.5000, 4.0000, 1.3000],
        [7.0000, 3.2000, 4.7000, 1.4000],
        [5.5000, 4.2000, 1.4000, 0.2000],
        [7.3000, 2.9000, 6.3000, 1.8000],
        [7.7000, 2.8000, 6.7000, 2.0000],
        [4.9000, 3.1000, 1.5000, 0.1000],
        [7.2000, 3.0000, 5.8000, 1.6000],
        [6.7000, 3.1000, 4.7000, 1.5000],
        [5.8000, 2.7000, 3.9000, 1.2000],
        [5.5000, 2.4000, 3.8000, 1.1000],
        [5.0000, 3.5000, 1.6000, 0.6000],
        [5.1000, 3.8000, 1.6000, 0.2000],
        [4.8000, 3.0000, 1.4000, 0.1000],
        [6.5000, 3.0000, 5.5000, 1.8000],
        [6.7000, 3.0000, 5.2000, 2.3000],
        [6.8000, 2.8000, 4.8000, 1.4000],
        [7.4000, 2.8000, 6.1000, 1.9000],
        [5.0000, 3.4000, 1.6000, 0.4000],
        [6.3000, 3.3000, 6.0000, 2.5000],
        [5.7000, 2.8000, 4.1000, 1.3000],
        [5.1000, 3.8000, 1.9000, 0.4000],
        [6.6000, 2.9000, 4.6000, 1.3000],
        [6.3000, 3.4000, 5.6000, 2.4000],
        [5.0000, 3.2000, 1.2000, 0.2000],
        [5.9000, 3.2000, 4.8000, 1.8000],
        [4.7000, 3.2000, 1.6000, 0.2000],
        [5.1000, 3.8000, 1.5000, 0.3000],
        [5.7000, 2.6000, 3.5000, 1.0000],
        [5.7000, 4.4000, 1.5000, 0.4000],
        [5.0000, 2.0000, 3.5000, 1.0000],
        [4.4000, 3.2000, 1.3000, 0.2000],
        [5.2000, 3.4000, 1.4000, 0.2000],
        [5.5000, 2.3000, 4.0000, 1.3000],
        [7.6000, 3.0000, 6.6000, 2.1000],
        [4.4000, 2.9000, 1.4000, 0.2000],
        [5.7000, 3.8000, 1.7000, 0.3000],
        [7.7000, 3.0000, 6.1000, 2.3000],
        [4.9000, 2.5000, 4.5000, 1.7000],
        [5.9000, 3.0000, 5.1000, 1.8000],
        [7.2000, 3.6000, 6.1000, 2.5000],
        [5.8000, 2.8000, 5.1000, 2.4000],
        [4.7000, 3.2000, 1.3000, 0.2000],
        [6.2000, 3.4000, 5.4000, 2.3000],
        [5.7000, 3.0000, 4.2000, 1.2000],
        [5.6000, 2.7000, 4.2000, 1.3000],
        [5.2000, 4.1000, 1.5000, 0.1000],
        [5.1000, 3.5000, 1.4000, 0.3000],
```

```
        [5.0000, 3.0000, 1.6000, 0.2000],
        [6.3000, 2.3000, 4.4000, 1.3000],
        [6.5000, 3.2000, 5.1000, 2.0000],
        [5.6000, 2.8000, 4.9000, 2.0000],
        [5.4000, 3.4000, 1.7000, 0.2000],
        [5.9000, 3.0000, 4.2000, 1.5000],
        [6.2000, 2.2000, 4.5000, 1.5000],
        [5.1000, 3.4000, 1.5000, 0.2000],
        [6.9000, 3.1000, 5.4000, 2.1000],
        [4.6000, 3.2000, 1.4000, 0.2000],
        [5.8000, 2.7000, 4.1000, 1.0000],
        [5.8000, 2.7000, 5.1000, 1.9000],
        [6.0000, 2.2000, 4.0000, 1.0000],
        [6.3000, 2.7000, 4.9000, 1.8000],
        [7.1000, 3.0000, 5.9000, 2.1000],
        [6.3000, 2.9000, 5.6000, 1.8000],
        [4.6000, 3.1000, 1.5000, 0.2000],
        [4.4000, 3.0000, 1.3000, 0.2000],
        [5.5000, 2.6000, 4.4000, 1.2000],
        [5.4000, 3.4000, 1.5000, 0.4000],
        [4.9000, 2.4000, 3.3000, 1.0000],
        [6.2000, 2.8000, 4.8000, 1.8000],
        [7.2000, 3.2000, 6.0000, 1.8000],
        [6.3000, 3.3000, 4.7000, 1.6000],
        [5.6000, 3.0000, 4.5000, 1.5000],
        [6.0000, 2.7000, 5.1000, 1.6000],
        [6.0000, 2.2000, 5.0000, 1.5000],
        [6.4000, 2.9000, 4.3000, 1.3000],
        [5.8000, 2.6000, 4.0000, 1.2000],
        [6.9000, 3.1000, 5.1000, 2.3000],
        [5.6000, 3.0000, 4.1000, 1.3000],
        [5.4000, 3.9000, 1.3000, 0.4000],
        [5.3000, 3.7000, 1.5000, 0.2000],
        [6.3000, 2.5000, 4.9000, 1.5000],
        [5.0000, 3.6000, 1.4000, 0.2000],
        [5.1000, 3.3000, 1.7000, 0.5000],
        [6.1000, 2.8000, 4.7000, 1.2000],
        [6.2000, 2.9000, 4.3000, 1.3000],
        [6.7000, 3.0000, 5.0000, 1.7000],
        [6.1000, 2.6000, 5.6000, 1.4000],
        [6.4000, 2.7000, 5.3000, 1.9000],
        [4.5000, 2.3000, 1.3000, 0.3000],
        [6.1000, 2.8000, 4.0000, 1.3000],
        [5.4000, 3.0000, 4.5000, 1.5000],
        [6.5000, 3.0000, 5.2000, 2.0000],
        [6.0000, 3.0000, 4.8000, 1.8000],
        [5.0000, 3.5000, 1.3000, 0.3000],
        [6.5000, 3.0000, 5.8000, 2.2000],
        [5.0000, 3.3000, 1.4000, 0.2000]]), tensor([1, 0, 0, 1, 2, 1,
2, 2, 2, 1, 1, 0, 2, 2, 0, 2, 1, 1, 1, 0, 0, 0, 2, 2,
        1, 2, 0, 2, 1, 0, 1, 2, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 2, 0, 0,
2, 2, 2,
        2, 2, 0, 2, 1, 1, 0, 0, 0, 1, 2, 2, 0, 1, 1, 0, 2, 0, 1, 2, 1,
2, 2, 2,
        0, 0, 1, 0, 1, 2, 2, 1, 1, 1, 2, 1, 1, 2, 1, 0, 0, 1, 0, 0, 1,
1, 1, 2,
        2, 0, 1, 1, 2, 2, 0, 2, 0])]
1 [tensor([[5.5000, 2.4000, 3.7000, 1.0000],
        [6.4000, 3.2000, 4.5000, 1.5000],
        [5.6000, 2.5000, 3.9000, 1.1000],
        [5.1000, 3.7000, 1.5000, 0.4000],
```

```
        [5.0000, 2.3000, 3.3000, 1.0000],
        [5.0000, 3.4000, 1.5000, 0.2000],
        [5.7000, 2.9000, 4.2000, 1.3000],
        [6.1000, 3.0000, 4.9000, 1.8000],
        [5.1000, 2.5000, 3.0000, 1.1000],
        [6.0000, 2.9000, 4.5000, 1.5000],
        [4.3000, 3.0000, 1.1000, 0.1000],
        [6.4000, 2.8000, 5.6000, 2.1000],
        [5.1000, 3.5000, 1.4000, 0.2000],
        [5.7000, 2.8000, 4.5000, 1.3000],
        [4.9000, 3.1000, 1.5000, 0.1000],
        [7.7000, 2.6000, 6.9000, 2.3000],
        [6.6000, 3.0000, 4.4000, 1.4000],
        [4.8000, 3.0000, 1.4000, 0.3000],
        [6.4000, 2.8000, 5.6000, 2.2000],
        [6.8000, 3.2000, 5.9000, 2.3000],
        [5.4000, 3.7000, 1.5000, 0.2000],
        [4.9000, 3.1000, 1.5000, 0.1000],
        [6.9000, 3.2000, 5.7000, 2.3000],
        [5.8000, 2.7000, 5.1000, 1.9000],
        [6.1000, 2.9000, 4.7000, 1.4000],
        [4.6000, 3.6000, 1.0000, 0.2000],
        [5.2000, 3.5000, 1.5000, 0.2000],
        [6.8000, 3.0000, 5.5000, 2.1000],
        [7.7000, 3.8000, 6.7000, 2.2000],
        [6.0000, 3.4000, 4.5000, 1.6000],
        [5.7000, 2.5000, 5.0000, 2.0000],
        [6.5000, 2.8000, 4.6000, 1.5000],
        [4.6000, 3.4000, 1.4000, 0.3000],
        [5.2000, 2.7000, 3.9000, 1.4000],
        [5.5000, 3.5000, 1.3000, 0.2000],
        [4.9000, 3.0000, 1.4000, 0.2000],
        [6.3000, 2.5000, 5.0000, 1.9000],
        [6.1000, 3.0000, 4.6000, 1.4000],
        [6.4000, 3.2000, 5.3000, 2.3000],
        [5.8000, 4.0000, 1.2000, 0.2000],
        [6.3000, 2.8000, 5.1000, 1.5000],
        [4.8000, 3.1000, 1.6000, 0.2000],
        [6.7000, 3.3000, 5.7000, 2.5000],
        [5.4000, 3.9000, 1.7000, 0.4000],
        [7.9000, 3.8000, 6.4000, 2.0000]]), tensor([1, 1, 1, 0, 1, 0,
1, 2, 1, 1, 0, 2, 0, 1, 0, 2, 1, 0, 2, 2, 0, 0, 2, 2,
        1, 0, 0, 2, 2, 1, 2, 1, 0, 1, 0, 0, 2, 1, 2, 0, 2, 0, 2, 0,
2])]
```

In [16]:

```
list(iris_loader)[0][1].bincount()
```

Out[16]:

```
tensor([30, 36, 39])
```

In [17]:

```
next(iter(iris_loader))
```

Out[17]:

```
[tensor([[5.4000, 3.7000, 1.5000, 0.2000],
         [4.7000, 3.2000, 1.3000, 0.2000],
         [6.1000, 3.0000, 4.6000, 1.4000],
         [4.3000, 3.0000, 1.1000, 0.1000],
         [5.0000, 3.5000, 1.3000, 0.3000],
         [7.2000, 3.2000, 6.0000, 1.8000],
         [4.8000, 3.4000, 1.9000, 0.2000],
         [6.4000, 3.1000, 5.5000, 1.8000],
         [6.6000, 3.0000, 4.4000, 1.4000],
         [6.8000, 3.2000, 5.9000, 2.3000],
         [6.4000, 3.2000, 4.5000, 1.5000],
         [5.0000, 2.3000, 3.3000, 1.0000],
         [6.0000, 2.2000, 4.0000, 1.0000],
         [6.7000, 3.1000, 5.6000, 2.4000],
         [6.0000, 2.7000, 5.1000, 1.6000],
         [6.2000, 2.8000, 4.8000, 1.8000],
         [5.4000, 3.4000, 1.7000, 0.2000],
         [5.4000, 3.9000, 1.7000, 0.4000],
```

A Quick Note on Torchvision

PyTorch offers another powerful dataset tool called **`torchvision`** (https://pytorch.org/docs/stable/torchvision/index.html), which is useful when working with image data. We'll go into a lot more detail in the Convolutional Neural Network (CNN) section. For now, just know that torchvision offers built-in image datasets like MNIST (https://en.wikipedia.org/wiki/MNIST_database) and CIFAR-10 (https://en.wikipedia.org/wiki/CIFAR-10), as well as tools for transforming images into tensors.