ex1_sol

April 18, 2018

1 Exercise 1 - Machine Learning Basics

This exercise is based on https://github.com/rasbt/pydata-chicago2016-ml-tutorial

2 Table of Contents

- Section **??**
  - Section 3.1
  - Section 3.2
  - Section 3.3
  - Section 3.4
- Section **??**
  - Section 4.1
  - Section 4.2
  - Section 4.3
  - Section 4.4
  - Section 4.5
  - Section 4.6

3 1 Linear Regression

3.1 Loading the dataset

We will use a dataset of an old publication which studied the relation of the brain weight to the head size for different gender and age ranges.

Source: R.J. Gladstone (1905). "A Study of the Relations of the Brain to to the Size of the Head", Biometrika, Vol. 4, pp105-123

The dataset is stored in a file called `dataset_brain.txt`

Description: Brain weight (grams) and head size (cubic cm) for 237 adults classified by gender and age group.

Variables/Columns - Gender (1=Male, 2=Female) - Age Range (1=20-46, 2=46+) - Head size (cm^3) - Brain weight (grams)

3.1.1 Task 1: Print the first 30 lines of the dataset

```
In [1]: with open("dataset_brain.txt") as ds_brain:
            firstNlines=ds_brain.readlines()[0:30] #put here the interval you want
        print firstNlines
```

```
['# Source: R.J. Gladstone (1905). "A Study of the Relations of the Brain to \n', '# to the Size
```

We will use pandas to read in the dataset.

https://pandas.pydata.org/pandas-docs/stable/

'pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with "relational" or "labeled" data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python. Additionally, it has the broader goal of becoming the most powerful and flexible open source data analysis / manipulation tool available in any language. It is already well on its way toward this goal.' (quoted from web page)

```
In [2]: import pandas as pd
```

The file contains 'comma separated values' (CSV) and we will use pandas `DataFrame` to handle the data.

https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html#pandas.DataFrame

```
In [3]: df = pd.read_csv('dataset_brain.txt',
                         encoding='utf-8',
                         comment='#',
                         sep='\s+')
        df.head(10)
```

Out[3]:

	gender	age-group	head-size	brain-weight
0	1	1	4512	1530
1	1	1	3738	1297
2	1	1	4261	1335
3	1	1	3777	1282
4	1	1	4177	1590
5	1	1	3585	1300
6	1	1	3785	1400
7	1	1	3559	1255
8	1	1	3613	1355
9	1	1	3982	1375

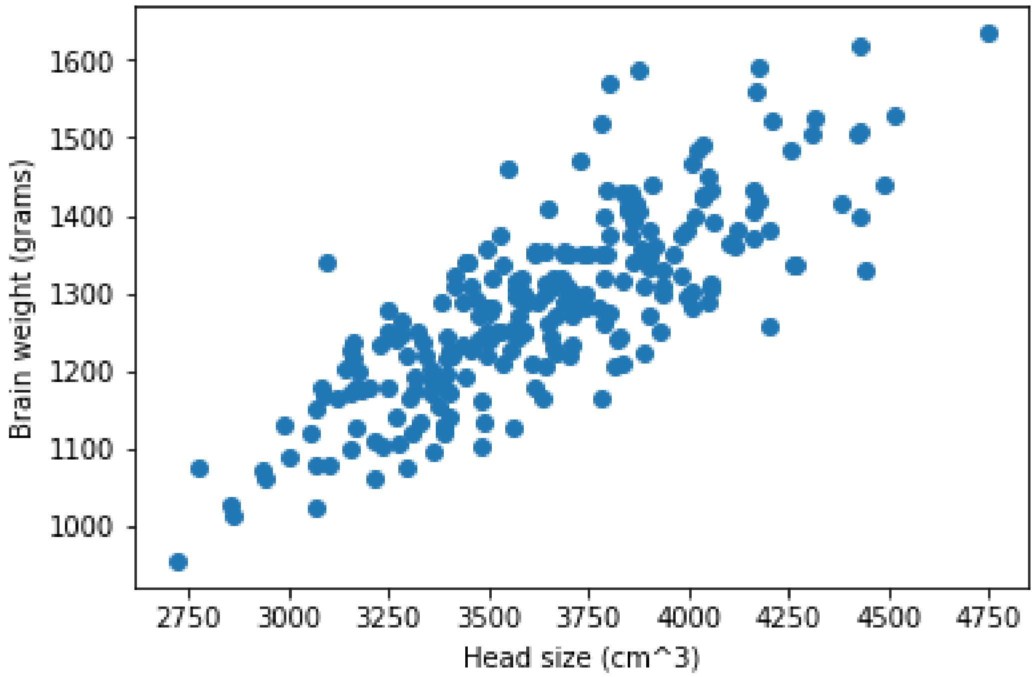
Let's look at the relation of the brain weight to the head size by plotting them in a 2D scatter plot. We will use `matplotlib` for that.

https://matplotlib.org/

```
In [4]: %matplotlib inline
        import matplotlib.pyplot as plt
```

We can call the columns of the pandas DataFrame simply by using the keys.

```
In [5]: plt.scatter(df['head-size'], df['brain-weight'])
        plt.xlabel('Head size (cm^3)')
        plt.ylabel('Brain weight (grams)');
```



3.2 Preparing the dataset

In order to use the dataset, we need to retrieve a numpy array containing only the values. http://www.numpy.org/

```
In [6]: import numpy as np
```

```
In [7]: y = df['brain-weight'].values
        print y
```

```
[1530 1297 1335 1282 1590 1300 1400 1255 1355 1375 1340 1380 1355 1522 1208
 1405 1358 1292 1340 1400 1357 1287 1275 1270 1635 1505 1490 1485 1310 1420
 1318 1432 1364 1405 1432 1207 1375 1350 1236 1250 1350 1320 1525 1570 1340
 1422 1506 1215 1311 1300 1224 1350 1335 1390 1400 1225 1310 1560 1330 1222
 1415 1175 1330 1485 1470 1135 1310 1154 1510 1415 1468 1390 1380 1432 1240
 1195 1225 1188 1252 1315 1245 1430 1279 1245 1309 1412 1120 1220 1280 1440
 1370 1192 1230 1346 1290 1165 1240 1132 1242 1270 1218 1430 1588 1320 1290
 1260 1425 1226 1360 1620 1310 1250 1295 1290 1290 1275 1250 1270 1362 1300
 1173 1256 1440 1180 1306 1350 1125 1165 1312 1300 1270 1335 1450 1310 1027
 1235 1260 1165 1080 1127 1270 1252 1200 1290 1334 1380 1140 1243 1340 1168
 1322 1249 1321 1192 1373 1170 1265 1235 1302 1241 1078 1520 1460 1075 1280
```

```
 1180 1250 1190 1374 1306 1202 1240 1316 1280 1350 1180 1210 1127 1324 1210
 1290 1100 1280 1175 1160 1205 1163 1022 1243 1350 1237 1204 1090 1355 1250
 1076 1120 1220 1240 1220 1095 1235 1105 1405 1150 1305 1220 1296 1175  955
 1070 1320 1060 1130 1250 1225 1180 1178 1142 1130 1185 1012 1280 1103 1408
 1300 1246 1380 1350 1060 1350 1220 1110 1215 1104 1170 1120]
```

How many data points do we have?

```
In [8]: y.shape
```

```
Out[8]: (237,)
```

The same with the head size:

```
In [9]: X = df['head-size'].values
        print X
        X.shape
```

```
[4512 3738 4261 3777 4177 3585 3785 3559 3613 3982 3443 3993 3640 4208 3832
 3876 3497 3466 3095 4424 3878 4046 3804 3710 4747 4423 4036 4022 3454 4175
 3787 3796 4103 4161 4158 3814 3527 3748 3334 3492 3962 3505 4315 3804 3863
 4034 4308 3165 3641 3644 3891 3793 4270 4063 4012 3458 3890 4166 3935 3669
 3866 3393 4442 4253 3727 3329 3415 3372 4430 4381 4008 3858 4121 4057 3824
 3394 3558 3362 3930 3835 3830 3856 3249 3577 3933 3850 3309 3406 3506 3907
 4160 3318 3662 3899 3700 3779 3473 3490 3654 3478 3495 3834 3876 3661 3618
 3648 4032 3399 3916 4430 3695 3524 3571 3594 3383 3499 3589 3900 4114 3937
 3399 4200 4488 3614 4051 3782 3391 3124 4053 3582 3666 3532 4046 3667 2857
 3436 3791 3302 3104 3171 3572 3530 3175 3438 3903 3899 3401 3267 3451 3090
 3413 3323 3680 3439 3853 3156 3279 3707 4006 3269 3071 3779 3548 3292 3497
 3082 3248 3358 3803 3566 3145 3503 3571 3724 3615 3203 3609 3561 3979 3533
 3689 3158 4005 3181 3479 3642 3632 3069 3394 3703 3165 3354 3000 3687 3556
 2773 3058 3344 3493 3297 3360 3228 3277 3851 3067 3692 3402 3995 3318 2720
 2937 3580 2939 2989 3586 3156 3246 3170 3268 3389 3381 2864 3740 3479 3647
 3716 3284 4204 3735 3218 3685 3704 3214 3394 3233 3352 3391]
```

```
Out[9]: (237,)
```

Instead of an array, we would like to have n arrays containing one value:

```
In [10]: X = X[:, None]
         print X
```

```
[[4512]
 [3738]
 [4261]
 [3777]
 [4177]
```

[3585]
[3785]
[3559]
[3613]
[3982]
[3443]
[3993]
[3640]
[4208]
[3832]
[3876]
[3497]
[3466]
[3095]
[4424]
[3878]
[4046]
[3804]
[3710]
[4747]
[4423]
[4036]
[4022]
[3454]
[4175]
[3787]
[3796]
[4103]
[4161]
[4158]
[3814]
[3527]
[3748]
[3334]
[3492]
[3962]
[3505]
[4315]
[3804]
[3863]
[4034]
[4308]
[3165]
[3641]
[3644]
[3891]
[3793]
[4270]

[4063]
[4012]
[3458]
[3890]
[4166]
[3935]
[3669]
[3866]
[3393]
[4442]
[4253]
[3727]
[3329]
[3415]
[3372]
[4430]
[4381]
[4008]
[3858]
[4121]
[4057]
[3824]
[3394]
[3558]
[3362]
[3930]
[3835]
[3830]
[3856]
[3249]
[3577]
[3933]
[3850]
[3309]
[3406]
[3506]
[3907]
[4160]
[3318]
[3662]
[3899]
[3700]
[3779]
[3473]
[3490]
[3654]
[3478]
[3495]

[3834]
[3876]
[3661]
[3618]
[3648]
[4032]
[3399]
[3916]
[4430]
[3695]
[3524]
[3571]
[3594]
[3383]
[3499]
[3589]
[3900]
[4114]
[3937]
[3399]
[4200]
[4488]
[3614]
[4051]
[3782]
[3391]
[3124]
[4053]
[3582]
[3666]
[3532]
[4046]
[3667]
[2857]
[3436]
[3791]
[3302]
[3104]
[3171]
[3572]
[3530]
[3175]
[3438]
[3903]
[3899]
[3401]
[3267]
[3451]

[3090]
[3413]
[3323]
[3680]
[3439]
[3853]
[3156]
[3279]
[3707]
[4006]
[3269]
[3071]
[3779]
[3548]
[3292]
[3497]
[3082]
[3248]
[3358]
[3803]
[3566]
[3145]
[3503]
[3571]
[3724]
[3615]
[3203]
[3609]
[3561]
[3979]
[3533]
[3689]
[3158]
[4005]
[3181]
[3479]
[3642]
[3632]
[3069]
[3394]
[3703]
[3165]
[3354]
[3000]
[3687]
[3556]
[2773]
[3058]

```
 [3344]
 [3493]
 [3297]
 [3360]
 [3228]
 [3277]
 [3851]
 [3067]
 [3692]
 [3402]
 [3995]
 [3318]
 [2720]
 [2937]
 [3580]
 [2939]
 [2989]
 [3586]
 [3156]
 [3246]
 [3170]
 [3268]
 [3389]
 [3381]
 [2864]
 [3740]
 [3479]
 [3647]
 [3716]
 [3284]
 [4204]
 [3735]
 [3218]
 [3685]
 [3704]
 [3214]
 [3394]
 [3233]
 [3352]
 [3391]]
```

We will use the machine learning tool and library `scikit-learn` in the following.
http://scikit-learn.org/stable/

A very useful functionality of scikit learn is to easily split the dataset into training and testing dataset. The dataset is split randomly with seed 123 and the test size is 30%, train size 70%:

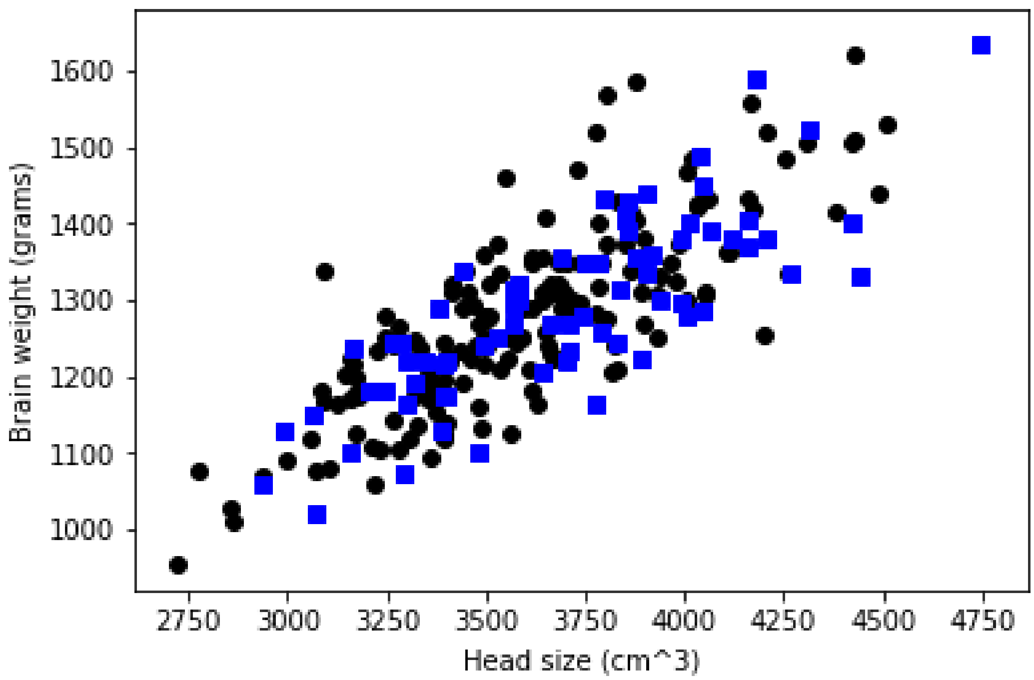
```
In [11]: from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.3, random_state=123)
```

3.2.1 Task 2: Plot the training and testing dataset separately again in a 2D scatter plot including axis label. Use different colors (option c(olor)='blue') and different marker (option marker='o')

https://matplotlib.org/api/colors_api.html
https://matplotlib.org/api/markers_api.html

```
In [12]: plt.scatter(X_train, y_train, c='black', marker='o')
         plt.scatter(X_test, y_test, c='blue', marker='s')
         plt.xlabel('Head size (cm^3)')
         plt.ylabel('Brain weight (grams)');
```



3.3 Fitting the model

We would like to fit the training data now using the LinearRegression model of scikit-learn:
http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html
Which uses a linear function and the ordinary least squares method.

```
In [13]: from sklearn.linear_model import LinearRegression

         lr = LinearRegression()
         lr.fit(X_train, y_train)
         y_pred = lr.predict(X_test)
```

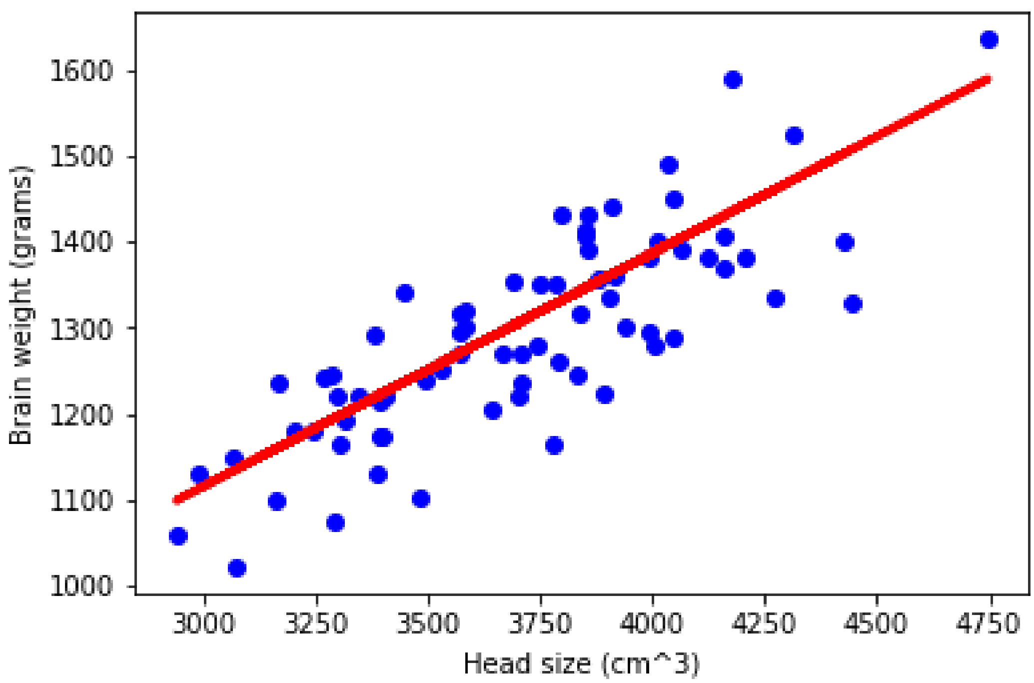
OK, what is the result of the fit?

```
In [14]: # The coefficients
         print 'Coefficients: \n', lr.coef_
         # The intercept
         print 'Intercept: \n', lr.intercept_
```

```
Coefficients:
[ 0.271117]
Intercept:
302.030331961
```

OK, let's plot this linear function.

```
In [15]: plt.scatter(X_test, y_test,  color='blue')
         plt.plot(X_test, y_pred, color='red', linewidth=3)
         plt.xlabel('Head size (cm^3)')
         plt.ylabel('Brain weight (grams)');
```



3.4 Evaluating the model

How do we know if the fit was good? We need to define a performance measure. One way is to calculate the **Coefficient of determination**, denoted R^2. It is the proportion of the variance in the dependent variable that is predictable from the independent variables. It is calculated the following way:

```
In [16]: sum_of_squares = ((y_test - y_pred) ** 2).sum()
         res_sum_of_squares = ((y_test - y_test.mean()) ** 2).sum()
         r2_score = 1 - (sum_of_squares / res_sum_of_squares)
         print('R2 score: %.2f' % r2_score)
```

```
R2 score: 0.63
```

It ranges from 0 to 1 and values close to 1 means a good agreement. Luckily, scikit-learn has several performance measures for regression (metrics) already included:
http://scikit-learn.org/stable/modules/model_evaluation.html#regression-metrics

```
In [17]: from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error

         # Explained variance score: 1 is perfect prediction
         print('Coefficient of determination: %.2f' % r2_score(y_test, y_pred))
         # The mean squared error
         print("Mean squared error: %.2f" % mean_squared_error(y_test, y_pred))
         # The mean squared error
         print("Mean absolute error: %.2f" % mean_absolute_error(y_test, y_pred))
```

```
Coefficient of determination: 0.63
Mean squared error: 5068.22
Mean absolute error: 57.08
```

4 2 Classification

4.1 The Iris dataset

4.1.1 Task 3: The Iris flower dataset is stored in file `dataset_iris.txt`. Read in the dataset using a pandas DatafFrame and have a look at the first entries.

```
In [18]: with open("dataset_iris.txt") as ds_brain:
             firstNlines=ds_brain.readlines()[:100] #put here the interval you want
         print firstNlines

         df = pd.read_csv('dataset_iris.txt',
                          encoding='utf-8',
                          comment='#',
                          sep=',')
         df.head()
```

```
['# Download source: https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data\n'
```

Out[18]:

	sepal_length	sepal_width	petal_length	petal_width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa

2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

We now need to create a 150x4 design matrix containing only our feature values. In order to do that, we need to strip the class column from the dataset. We use the `iloc` function for that:

`DataFrame.iloc Purely integer-location based indexing for selection by position.`
https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.iloc.html

```
In [19]: X = df.iloc[:, :4]
         X
```

Out[19]:

	sepal_length	sepal_width	petal_length	petal_width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
5	5.4	3.9	1.7	0.4
6	4.6	3.4	1.4	0.3
7	5.0	3.4	1.5	0.2
8	4.4	2.9	1.4	0.2
9	4.9	3.1	1.5	0.1
10	5.4	3.7	1.5	0.2
11	4.8	3.4	1.6	0.2
12	4.8	3.0	1.4	0.1
13	4.3	3.0	1.1	0.1
14	5.8	4.0	1.2	0.2
15	5.7	4.4	1.5	0.4
16	5.4	3.9	1.3	0.4
17	5.1	3.5	1.4	0.3
18	5.7	3.8	1.7	0.3
19	5.1	3.8	1.5	0.3
20	5.4	3.4	1.7	0.2
21	5.1	3.7	1.5	0.4
22	4.6	3.6	1.0	0.2
23	5.1	3.3	1.7	0.5
24	4.8	3.4	1.9	0.2
25	5.0	3.0	1.6	0.2
26	5.0	3.4	1.6	0.4
27	5.2	3.5	1.5	0.2
28	5.2	3.4	1.4	0.2
29	4.7	3.2	1.6	0.2
..	...	...	...	...
120	6.9	3.2	5.7	2.3
121	5.6	2.8	4.9	2.0
122	7.7	2.8	6.7	2.0
123	6.3	2.7	4.9	1.8

```
124           6.7          3.3           5.7          2.1
125           7.2          3.2           6.0          1.8
126           6.2          2.8           4.8          1.8
127           6.1          3.0           4.9          1.8
128           6.4          2.8           5.6          2.1
129           7.2          3.0           5.8          1.6
130           7.4          2.8           6.1          1.9
131           7.9          3.8           6.4          2.0
132           6.4          2.8           5.6          2.2
133           6.3          2.8           5.1          1.5
134           6.1          2.6           5.6          1.4
135           7.7          3.0           6.1          2.3
136           6.3          3.4           5.6          2.4
137           6.4          3.1           5.5          1.8
138           6.0          3.0           4.8          1.8
139           6.9          3.1           5.4          2.1
140           6.7          3.1           5.6          2.4
141           6.9          3.1           5.1          2.3
142           5.8          2.7           5.1          1.9
143           6.8          3.2           5.9          2.3
144           6.7          3.3           5.7          2.5
145           6.7          3.0           5.2          2.3
146           6.3          2.5           5.0          1.9
147           6.5          3.0           5.2          2.0
148           6.2          3.4           5.4          2.3
149           5.9          3.0           5.1          1.8

[150 rows x 4 columns]
```

And now we get 150x4 numpy array (design matrix) by using the values function:

```
In [20]: X = X.values
         X

Out[20]: array([[ 5.1,  3.5,  1.4,  0.2],
                [ 4.9,  3. ,  1.4,  0.2],
                [ 4.7,  3.2,  1.3,  0.2],
                [ 4.6,  3.1,  1.5,  0.2],
                [ 5. ,  3.6,  1.4,  0.2],
                [ 5.4,  3.9,  1.7,  0.4],
                [ 4.6,  3.4,  1.4,  0.3],
                [ 5. ,  3.4,  1.5,  0.2],
                [ 4.4,  2.9,  1.4,  0.2],
                [ 4.9,  3.1,  1.5,  0.1],
                [ 5.4,  3.7,  1.5,  0.2],
                [ 4.8,  3.4,  1.6,  0.2],
                [ 4.8,  3. ,  1.4,  0.1],
                [ 4.3,  3. ,  1.1,  0.1],
```

```
[ 5.8,  4. ,  1.2,  0.2],
[ 5.7,  4.4,  1.5,  0.4],
[ 5.4,  3.9,  1.3,  0.4],
[ 5.1,  3.5,  1.4,  0.3],
[ 5.7,  3.8,  1.7,  0.3],
[ 5.1,  3.8,  1.5,  0.3],
[ 5.4,  3.4,  1.7,  0.2],
[ 5.1,  3.7,  1.5,  0.4],
[ 4.6,  3.6,  1. ,  0.2],
[ 5.1,  3.3,  1.7,  0.5],
[ 4.8,  3.4,  1.9,  0.2],
[ 5. ,  3. ,  1.6,  0.2],
[ 5. ,  3.4,  1.6,  0.4],
[ 5.2,  3.5,  1.5,  0.2],
[ 5.2,  3.4,  1.4,  0.2],
[ 4.7,  3.2,  1.6,  0.2],
[ 4.8,  3.1,  1.6,  0.2],
[ 5.4,  3.4,  1.5,  0.4],
[ 5.2,  4.1,  1.5,  0.1],
[ 5.5,  4.2,  1.4,  0.2],
[ 4.9,  3.1,  1.5,  0.1],
[ 5. ,  3.2,  1.2,  0.2],
[ 5.5,  3.5,  1.3,  0.2],
[ 4.9,  3.1,  1.5,  0.1],
[ 4.4,  3. ,  1.3,  0.2],
[ 5.1,  3.4,  1.5,  0.2],
[ 5. ,  3.5,  1.3,  0.3],
[ 4.5,  2.3,  1.3,  0.3],
[ 4.4,  3.2,  1.3,  0.2],
[ 5. ,  3.5,  1.6,  0.6],
[ 5.1,  3.8,  1.9,  0.4],
[ 4.8,  3. ,  1.4,  0.3],
[ 5.1,  3.8,  1.6,  0.2],
[ 4.6,  3.2,  1.4,  0.2],
[ 5.3,  3.7,  1.5,  0.2],
[ 5. ,  3.3,  1.4,  0.2],
[ 7. ,  3.2,  4.7,  1.4],
[ 6.4,  3.2,  4.5,  1.5],
[ 6.9,  3.1,  4.9,  1.5],
[ 5.5,  2.3,  4. ,  1.3],
[ 6.5,  2.8,  4.6,  1.5],
[ 5.7,  2.8,  4.5,  1.3],
[ 6.3,  3.3,  4.7,  1.6],
[ 4.9,  2.4,  3.3,  1. ],
[ 6.6,  2.9,  4.6,  1.3],
[ 5.2,  2.7,  3.9,  1.4],
[ 5. ,  2. ,  3.5,  1. ],
[ 5.9,  3. ,  4.2,  1.5],
```

```
[ 6. ,  2.2,  4. ,  1. ],
[ 6.1,  2.9,  4.7,  1.4],
[ 5.6,  2.9,  3.6,  1.3],
[ 6.7,  3.1,  4.4,  1.4],
[ 5.6,  3. ,  4.5,  1.5],
[ 5.8,  2.7,  4.1,  1. ],
[ 6.2,  2.2,  4.5,  1.5],
[ 5.6,  2.5,  3.9,  1.1],
[ 5.9,  3.2,  4.8,  1.8],
[ 6.1,  2.8,  4. ,  1.3],
[ 6.3,  2.5,  4.9,  1.5],
[ 6.1,  2.8,  4.7,  1.2],
[ 6.4,  2.9,  4.3,  1.3],
[ 6.6,  3. ,  4.4,  1.4],
[ 6.8,  2.8,  4.8,  1.4],
[ 6.7,  3. ,  5. ,  1.7],
[ 6. ,  2.9,  4.5,  1.5],
[ 5.7,  2.6,  3.5,  1. ],
[ 5.5,  2.4,  3.8,  1.1],
[ 5.5,  2.4,  3.7,  1. ],
[ 5.8,  2.7,  3.9,  1.2],
[ 6. ,  2.7,  5.1,  1.6],
[ 5.4,  3. ,  4.5,  1.5],
[ 6. ,  3.4,  4.5,  1.6],
[ 6.7,  3.1,  4.7,  1.5],
[ 6.3,  2.3,  4.4,  1.3],
[ 5.6,  3. ,  4.1,  1.3],
[ 5.5,  2.5,  4. ,  1.3],
[ 5.5,  2.6,  4.4,  1.2],
[ 6.1,  3. ,  4.6,  1.4],
[ 5.8,  2.6,  4. ,  1.2],
[ 5. ,  2.3,  3.3,  1. ],
[ 5.6,  2.7,  4.2,  1.3],
[ 5.7,  3. ,  4.2,  1.2],
[ 5.7,  2.9,  4.2,  1.3],
[ 6.2,  2.9,  4.3,  1.3],
[ 5.1,  2.5,  3. ,  1.1],
[ 5.7,  2.8,  4.1,  1.3],
[ 6.3,  3.3,  6. ,  2.5],
[ 5.8,  2.7,  5.1,  1.9],
[ 7.1,  3. ,  5.9,  2.1],
[ 6.3,  2.9,  5.6,  1.8],
[ 6.5,  3. ,  5.8,  2.2],
[ 7.6,  3. ,  6.6,  2.1],
[ 4.9,  2.5,  4.5,  1.7],
[ 7.3,  2.9,  6.3,  1.8],
[ 6.7,  2.5,  5.8,  1.8],
[ 7.2,  3.6,  6.1,  2.5],
```

```
       [ 6.5,  3.2,  5.1,  2. ],
       [ 6.4,  2.7,  5.3,  1.9],
       [ 6.8,  3. ,  5.5,  2.1],
       [ 5.7,  2.5,  5. ,  2. ],
       [ 5.8,  2.8,  5.1,  2.4],
       [ 6.4,  3.2,  5.3,  2.3],
       [ 6.5,  3. ,  5.5,  1.8],
       [ 7.7,  3.8,  6.7,  2.2],
       [ 7.7,  2.6,  6.9,  2.3],
       [ 6. ,  2.2,  5. ,  1.5],
       [ 6.9,  3.2,  5.7,  2.3],
       [ 5.6,  2.8,  4.9,  2. ],
       [ 7.7,  2.8,  6.7,  2. ],
       [ 6.3,  2.7,  4.9,  1.8],
       [ 6.7,  3.3,  5.7,  2.1],
       [ 7.2,  3.2,  6. ,  1.8],
       [ 6.2,  2.8,  4.8,  1.8],
       [ 6.1,  3. ,  4.9,  1.8],
       [ 6.4,  2.8,  5.6,  2.1],
       [ 7.2,  3. ,  5.8,  1.6],
       [ 7.4,  2.8,  6.1,  1.9],
       [ 7.9,  3.8,  6.4,  2. ],
       [ 6.4,  2.8,  5.6,  2.2],
       [ 6.3,  2.8,  5.1,  1.5],
       [ 6.1,  2.6,  5.6,  1.4],
       [ 7.7,  3. ,  6.1,  2.3],
       [ 6.3,  3.4,  5.6,  2.4],
       [ 6.4,  3.1,  5.5,  1.8],
       [ 6. ,  3. ,  4.8,  1.8],
       [ 6.9,  3.1,  5.4,  2.1],
       [ 6.7,  3.1,  5.6,  2.4],
       [ 6.9,  3.1,  5.1,  2.3],
       [ 5.8,  2.7,  5.1,  1.9],
       [ 6.8,  3.2,  5.9,  2.3],
       [ 6.7,  3.3,  5.7,  2.5],
       [ 6.7,  3. ,  5.2,  2.3],
       [ 6.3,  2.5,  5. ,  1.9],
       [ 6.5,  3. ,  5.2,  2. ],
       [ 6.2,  3.4,  5.4,  2.3],
       [ 5.9,  3. ,  5.1,  1.8]])
```

However, we also need a numpy array containing the class labels in order to classify. Let's get the class column and create a numpy array out of it:

```
In [21]: y = df['class'].values
         y

Out[21]: array([u'Iris-setosa', u'Iris-setosa', u'Iris-setosa', u'Iris-setosa',
                u'Iris-setosa', u'Iris-setosa', u'Iris-setosa', u'Iris-setosa',
```

```
u'Iris-setosa', u'Iris-setosa', u'Iris-setosa', u'Iris-setosa',
u'Iris-setosa', u'Iris-setosa', u'Iris-setosa', u'Iris-setosa',
u'Iris-setosa', u'Iris-setosa', u'Iris-setosa', u'Iris-setosa',
u'Iris-setosa', u'Iris-setosa', u'Iris-setosa', u'Iris-setosa',
u'Iris-setosa', u'Iris-setosa', u'Iris-setosa', u'Iris-setosa',
u'Iris-setosa', u'Iris-setosa', u'Iris-setosa', u'Iris-setosa',
u'Iris-setosa', u'Iris-setosa', u'Iris-setosa', u'Iris-setosa',
u'Iris-setosa', u'Iris-setosa', u'Iris-setosa', u'Iris-setosa',
u'Iris-setosa', u'Iris-setosa', u'Iris-setosa', u'Iris-setosa',
u'Iris-setosa', u'Iris-setosa', u'Iris-setosa', u'Iris-setosa',
u'Iris-setosa', u'Iris-setosa', u'Iris-versicolor',
u'Iris-versicolor', u'Iris-versicolor', u'Iris-versicolor',
u'Iris-versicolor', u'Iris-versicolor', u'Iris-versicolor',
u'Iris-versicolor', u'Iris-versicolor', u'Iris-versicolor',
u'Iris-versicolor', u'Iris-versicolor', u'Iris-versicolor',
u'Iris-versicolor', u'Iris-versicolor', u'Iris-versicolor',
u'Iris-versicolor', u'Iris-versicolor', u'Iris-versicolor',
u'Iris-versicolor', u'Iris-versicolor', u'Iris-versicolor',
u'Iris-versicolor', u'Iris-versicolor', u'Iris-versicolor',
u'Iris-versicolor', u'Iris-versicolor', u'Iris-versicolor',
u'Iris-versicolor', u'Iris-versicolor', u'Iris-versicolor',
u'Iris-versicolor', u'Iris-versicolor', u'Iris-versicolor',
u'Iris-versicolor', u'Iris-versicolor', u'Iris-versicolor',
u'Iris-versicolor', u'Iris-versicolor', u'Iris-versicolor',
u'Iris-versicolor', u'Iris-versicolor', u'Iris-versicolor',
u'Iris-versicolor', u'Iris-versicolor', u'Iris-versicolor',
u'Iris-versicolor', u'Iris-virginica', u'Iris-virginica',
u'Iris-virginica', u'Iris-virginica', u'Iris-virginica',
u'Iris-virginica', u'Iris-virginica', u'Iris-virginica',
u'Iris-virginica', u'Iris-virginica', u'Iris-virginica',
u'Iris-virginica', u'Iris-virginica', u'Iris-virginica',
u'Iris-virginica', u'Iris-virginica', u'Iris-virginica',
u'Iris-virginica', u'Iris-virginica', u'Iris-virginica',
u'Iris-virginica', u'Iris-virginica', u'Iris-virginica',
u'Iris-virginica', u'Iris-virginica', u'Iris-virginica',
u'Iris-virginica', u'Iris-virginica', u'Iris-virginica',
u'Iris-virginica', u'Iris-virginica', u'Iris-virginica',
u'Iris-virginica', u'Iris-virginica', u'Iris-virginica',
u'Iris-virginica', u'Iris-virginica', u'Iris-virginica',
u'Iris-virginica', u'Iris-virginica', u'Iris-virginica',
u'Iris-virginica', u'Iris-virginica', u'Iris-virginica',
u'Iris-virginica', u'Iris-virginica', u'Iris-virginica'], dtype=object)
```

We could also just inspect the targets by only looking at unique values:

```
In [22]: np.unique(y)
```

```
Out[22]: array([u'Iris-setosa', u'Iris-versicolor', u'Iris-virginica'], dtype=object)
```

4.2 Class label encoding

We will now use the `LabelEncoder` class to convert the class labels into numerical labels:

```
In [23]: from sklearn.preprocessing import LabelEncoder

         l_encoder = LabelEncoder()
         l_encoder.fit(y)
         l_encoder.classes_

Out[23]: array([u'Iris-setosa', u'Iris-versicolor', u'Iris-virginica'], dtype=object)
```

Simply, by using `transform`, we can convert it into numerical targets

```
In [24]: y_enc = l_encoder.transform(y)
         y_enc

Out[24]: array([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, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 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, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
                2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
                2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

Or just the unique values:

```
In [25]: np.unique(y_enc)

Out[25]: array([0, 1, 2])
```

We can also convert it back by using `inverse_transform`:

```
In [26]: np.unique(l_encoder.inverse_transform(y_enc))

Out[26]: array([u'Iris-setosa', u'Iris-versicolor', u'Iris-virginica'], dtype=object)
```

4.3 Scikit-learn's in-build datasets

Scikit-learn has also a couple of in-build datasets:
http://scikit-learn.org/stable/datasets/index.html
The iris dataset is part of it, which you can simply load:

```
In [27]: from sklearn.datasets import load_iris

         iris = load_iris()
         print(iris['DESCR'])
```

```
Iris Plants Database
====================

Notes
-----
Data Set Characteristics:
    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica
    :Summary Statistics:

    ============== ==== ==== ======= ===== ====================
                    Min  Max   Mean    SD   Class Correlation
    ============== ==== ==== ======= ===== ====================
    sepal length:   4.3  7.9   5.84   0.83    0.7826
    sepal width:    2.0  4.4   3.05   0.43   -0.4194
    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
    petal width:    0.1  2.5   1.20  0.76     0.9565  (high!)
    ============== ==== ==== ======= ===== ====================

    :Missing Attribute Values: None
    :Class Distribution: 33.3% for each of 3 classes.
    :Creator: R.A. Fisher
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
    :Date: July, 1988

This is a copy of UCI ML iris datasets.
http://archive.ics.uci.edu/ml/datasets/Iris

The famous Iris database, first used by Sir R.A Fisher

This is perhaps the best known database to be found in the
pattern recognition literature.  Fisher's paper is a classic in the field and
is referenced frequently to this day.  (See Duda & Hart, for example.)  The
data set contains 3 classes of 50 instances each, where each class refers to a
type of iris plant.  One class is linearly separable from the other 2; the
latter are NOT linearly separable from each other.

References
----------
```

```
- Fisher,R.A. "The use of multiple measurements in taxonomic problems"
  Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to
  Mathematical Statistics" (John Wiley, NY, 1950).
- Duda,R.O., & Hart,P.E. (1973) Pattern Classification and Scene Analysis.
  (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.
- Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System
  Structure and Classification Rule for Recognition in Partially Exposed
  Environments".  IEEE Transactions on Pattern Analysis and Machine
  Intelligence, Vol. PAMI-2, No. 1, 67-71.
- Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transactions
  on Information Theory, May 1972, 431-433.
- See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II
  conceptual clustering system finds 3 classes in the data.
- Many, many more ...
```

We get the feature design matrix by calling data:

In [28]:
```
iris.data
```

Out[28]:
```
array([[ 5.1,  3.5,  1.4,  0.2],
       [ 4.9,  3. ,  1.4,  0.2],
       [ 4.7,  3.2,  1.3,  0.2],
       [ 4.6,  3.1,  1.5,  0.2],
       [ 5. ,  3.6,  1.4,  0.2],
       [ 5.4,  3.9,  1.7,  0.4],
       [ 4.6,  3.4,  1.4,  0.3],
       [ 5. ,  3.4,  1.5,  0.2],
       [ 4.4,  2.9,  1.4,  0.2],
       [ 4.9,  3.1,  1.5,  0.1],
       [ 5.4,  3.7,  1.5,  0.2],
       [ 4.8,  3.4,  1.6,  0.2],
       [ 4.8,  3. ,  1.4,  0.1],
       [ 4.3,  3. ,  1.1,  0.1],
       [ 5.8,  4. ,  1.2,  0.2],
       [ 5.7,  4.4,  1.5,  0.4],
       [ 5.4,  3.9,  1.3,  0.4],
       [ 5.1,  3.5,  1.4,  0.3],
       [ 5.7,  3.8,  1.7,  0.3],
       [ 5.1,  3.8,  1.5,  0.3],
       [ 5.4,  3.4,  1.7,  0.2],
       [ 5.1,  3.7,  1.5,  0.4],
       [ 4.6,  3.6,  1. ,  0.2],
       [ 5.1,  3.3,  1.7,  0.5],
       [ 4.8,  3.4,  1.9,  0.2],
       [ 5. ,  3. ,  1.6,  0.2],
       [ 5. ,  3.4,  1.6,  0.4],
```

```
[ 5.2,  3.5,  1.5,  0.2],
[ 5.2,  3.4,  1.4,  0.2],
[ 4.7,  3.2,  1.6,  0.2],
[ 4.8,  3.1,  1.6,  0.2],
[ 5.4,  3.4,  1.5,  0.4],
[ 5.2,  4.1,  1.5,  0.1],
[ 5.5,  4.2,  1.4,  0.2],
[ 4.9,  3.1,  1.5,  0.1],
[ 5. ,  3.2,  1.2,  0.2],
[ 5.5,  3.5,  1.3,  0.2],
[ 4.9,  3.1,  1.5,  0.1],
[ 4.4,  3. ,  1.3,  0.2],
[ 5.1,  3.4,  1.5,  0.2],
[ 5. ,  3.5,  1.3,  0.3],
[ 4.5,  2.3,  1.3,  0.3],
[ 4.4,  3.2,  1.3,  0.2],
[ 5. ,  3.5,  1.6,  0.6],
[ 5.1,  3.8,  1.9,  0.4],
[ 4.8,  3. ,  1.4,  0.3],
[ 5.1,  3.8,  1.6,  0.2],
[ 4.6,  3.2,  1.4,  0.2],
[ 5.3,  3.7,  1.5,  0.2],
[ 5. ,  3.3,  1.4,  0.2],
[ 7. ,  3.2,  4.7,  1.4],
[ 6.4,  3.2,  4.5,  1.5],
[ 6.9,  3.1,  4.9,  1.5],
[ 5.5,  2.3,  4. ,  1.3],
[ 6.5,  2.8,  4.6,  1.5],
[ 5.7,  2.8,  4.5,  1.3],
[ 6.3,  3.3,  4.7,  1.6],
[ 4.9,  2.4,  3.3,  1. ],
[ 6.6,  2.9,  4.6,  1.3],
[ 5.2,  2.7,  3.9,  1.4],
[ 5. ,  2. ,  3.5,  1. ],
[ 5.9,  3. ,  4.2,  1.5],
[ 6. ,  2.2,  4. ,  1. ],
[ 6.1,  2.9,  4.7,  1.4],
[ 5.6,  2.9,  3.6,  1.3],
[ 6.7,  3.1,  4.4,  1.4],
[ 5.6,  3. ,  4.5,  1.5],
[ 5.8,  2.7,  4.1,  1. ],
[ 6.2,  2.2,  4.5,  1.5],
[ 5.6,  2.5,  3.9,  1.1],
[ 5.9,  3.2,  4.8,  1.8],
[ 6.1,  2.8,  4. ,  1.3],
[ 6.3,  2.5,  4.9,  1.5],
[ 6.1,  2.8,  4.7,  1.2],
[ 6.4,  2.9,  4.3,  1.3],
```

```
[ 6.6,  3. ,  4.4,  1.4],
[ 6.8,  2.8,  4.8,  1.4],
[ 6.7,  3. ,  5. ,  1.7],
[ 6. ,  2.9,  4.5,  1.5],
[ 5.7,  2.6,  3.5,  1. ],
[ 5.5,  2.4,  3.8,  1.1],
[ 5.5,  2.4,  3.7,  1. ],
[ 5.8,  2.7,  3.9,  1.2],
[ 6. ,  2.7,  5.1,  1.6],
[ 5.4,  3. ,  4.5,  1.5],
[ 6. ,  3.4,  4.5,  1.6],
[ 6.7,  3.1,  4.7,  1.5],
[ 6.3,  2.3,  4.4,  1.3],
[ 5.6,  3. ,  4.1,  1.3],
[ 5.5,  2.5,  4. ,  1.3],
[ 5.5,  2.6,  4.4,  1.2],
[ 6.1,  3. ,  4.6,  1.4],
[ 5.8,  2.6,  4. ,  1.2],
[ 5. ,  2.3,  3.3,  1. ],
[ 5.6,  2.7,  4.2,  1.3],
[ 5.7,  3. ,  4.2,  1.2],
[ 5.7,  2.9,  4.2,  1.3],
[ 6.2,  2.9,  4.3,  1.3],
[ 5.1,  2.5,  3. ,  1.1],
[ 5.7,  2.8,  4.1,  1.3],
[ 6.3,  3.3,  6. ,  2.5],
[ 5.8,  2.7,  5.1,  1.9],
[ 7.1,  3. ,  5.9,  2.1],
[ 6.3,  2.9,  5.6,  1.8],
[ 6.5,  3. ,  5.8,  2.2],
[ 7.6,  3. ,  6.6,  2.1],
[ 4.9,  2.5,  4.5,  1.7],
[ 7.3,  2.9,  6.3,  1.8],
[ 6.7,  2.5,  5.8,  1.8],
[ 7.2,  3.6,  6.1,  2.5],
[ 6.5,  3.2,  5.1,  2. ],
[ 6.4,  2.7,  5.3,  1.9],
[ 6.8,  3. ,  5.5,  2.1],
[ 5.7,  2.5,  5. ,  2. ],
[ 5.8,  2.8,  5.1,  2.4],
[ 6.4,  3.2,  5.3,  2.3],
[ 6.5,  3. ,  5.5,  1.8],
[ 7.7,  3.8,  6.7,  2.2],
[ 7.7,  2.6,  6.9,  2.3],
[ 6. ,  2.2,  5. ,  1.5],
[ 6.9,  3.2,  5.7,  2.3],
[ 5.6,  2.8,  4.9,  2. ],
[ 7.7,  2.8,  6.7,  2. ],
```

```
       [ 6.3,  2.7,  4.9,  1.8],
       [ 6.7,  3.3,  5.7,  2.1],
       [ 7.2,  3.2,  6. ,  1.8],
       [ 6.2,  2.8,  4.8,  1.8],
       [ 6.1,  3. ,  4.9,  1.8],
       [ 6.4,  2.8,  5.6,  2.1],
       [ 7.2,  3. ,  5.8,  1.6],
       [ 7.4,  2.8,  6.1,  1.9],
       [ 7.9,  3.8,  6.4,  2. ],
       [ 6.4,  2.8,  5.6,  2.2],
       [ 6.3,  2.8,  5.1,  1.5],
       [ 6.1,  2.6,  5.6,  1.4],
       [ 7.7,  3. ,  6.1,  2.3],
       [ 6.3,  3.4,  5.6,  2.4],
       [ 6.4,  3.1,  5.5,  1.8],
       [ 6. ,  3. ,  4.8,  1.8],
       [ 6.9,  3.1,  5.4,  2.1],
       [ 6.7,  3.1,  5.6,  2.4],
       [ 6.9,  3.1,  5.1,  2.3],
       [ 5.8,  2.7,  5.1,  1.9],
       [ 6.8,  3.2,  5.9,  2.3],
       [ 6.7,  3.3,  5.7,  2.5],
       [ 6.7,  3. ,  5.2,  2.3],
       [ 6.3,  2.5,  5. ,  1.9],
       [ 6.5,  3. ,  5.2,  2. ],
       [ 6.2,  3.4,  5.4,  2.3],
       [ 5.9,  3. ,  5.1,  1.8]])
```

And the target array:

```
In [29]: iris.target
```

```
Out[29]: array([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, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 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, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

4.4 Test/train splits

OK, now we need to split the dataset again in training and testing. Let's first assign the design matrix to X and the target to y:

```
In [30]: X, y = iris.data[:, :2], iris.target
         # ! We only use 2 features for visual purposes
```

How many example do we have of each class?

```
In [31]: print('Class labels:', np.unique(y))
         print('Class proportions:', np.bincount(y))
```

```
('Class labels:', array([0, 1, 2]))
('Class proportions:', array([50, 50, 50]))
```

4.4.1 Task 4: Split the dataset in 40% testing and 60% training sets. How many examples of each class do you expect in the training set? How many are there? What happened?

```
In [32]: from sklearn.model_selection import train_test_split

         X_train, X_test, y_train, y_test = train_test_split(
                 X, y, test_size=0.4, random_state=42)

         print('Class labels:', np.unique(y_train))
         print('Class proportions:', np.bincount(y_train))
```

```
('Class labels:', array([0, 1, 2]))
('Class proportions:', array([27, 31, 32]))
```

By default, the dataset is shuffled. What happens if we don't shuffle?

```
In [33]: from sklearn.model_selection import train_test_split

         X_train, X_test, y_train, y_test = train_test_split(
                 X, y, test_size=0.4, random_state=42, shuffle=False)

         print('Class labels:', np.unique(y_train))
         print('Class proportions:', np.bincount(y_train))
```

```
('Class labels:', array([0, 1]))
('Class proportions:', array([50, 40]))
```

OK, we want to shuffle, but we want equal portions of each class. We can achieve that by using the `stratify` option:

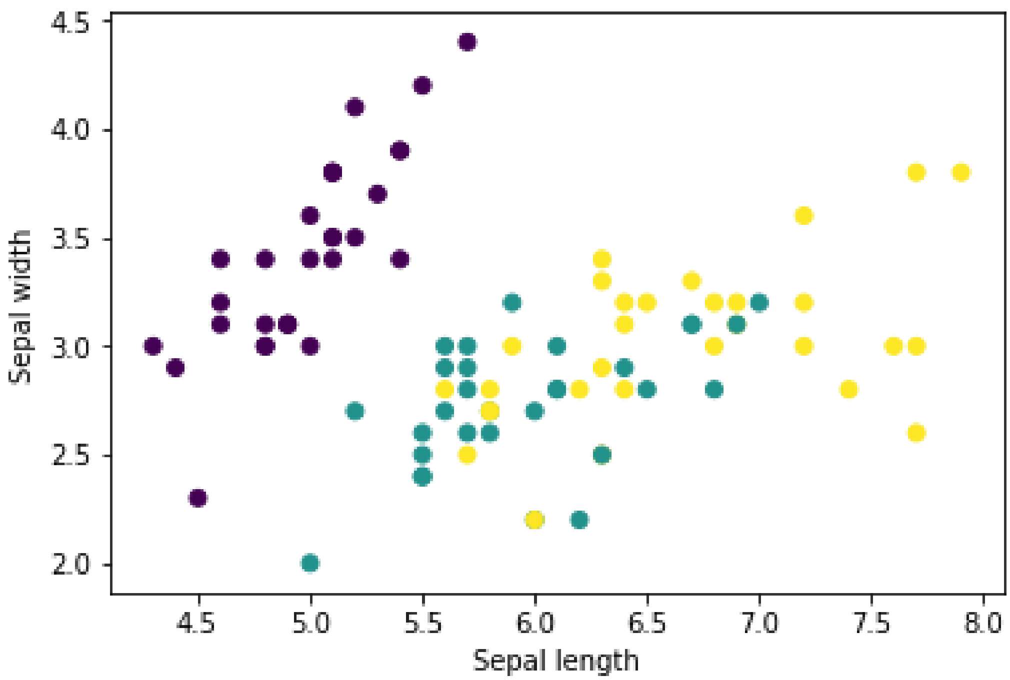
```
In [34]: from sklearn.model_selection import train_test_split

         X_train, X_test, y_train, y_test = train_test_split(
                 X, y, test_size=0.4, random_state=42,
                 stratify=y)

         print('Class labels:', np.unique(y_train))
         print('Class proportions:', np.bincount(y_train))
```

```
('Class labels:', array([0, 1, 2]))
('Class proportions:', array([30, 30, 30]))
```

4.4.2 Task 5: Plot the sepal length vs the sepal width of the training set for the different classes in a scatter plot. You can set different colors for the classes with c=y_train

In [35]:
```
# Plot the training points
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
```

Out[35]:
```
Text(0,0.5,u'Sepal width')
```



4.5 Logistic Regression

Let's perform a classification using logistic regression:

In [36]:
```
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression(solver='newton-cg',
                        multi_class='multinomial',
                        random_state=42)

lr.fit(X_train, y_train)
y_pred = lr.predict(X_test)
```

OK, how do we evaluate the classification? We can chose one of the classification performance measures:
http://scikit-learn.org/stable/modules/model_evaluation.html#classification-metrics

```
In [37]: from sklearn.metrics import accuracy_score, precision_score, recall_score, classificati
         print('Accuracy: %.2f' % accuracy_score(y_test, y_pred))
         print("Precision: %.2f" % precision_score(y_test, y_pred, average='weighted'))
         print("Recall: %.2f" % recall_score(y_test, y_pred, average='weighted'))
```

```
Accuracy: 0.77
Precision: 0.76
Recall: 0.77
```

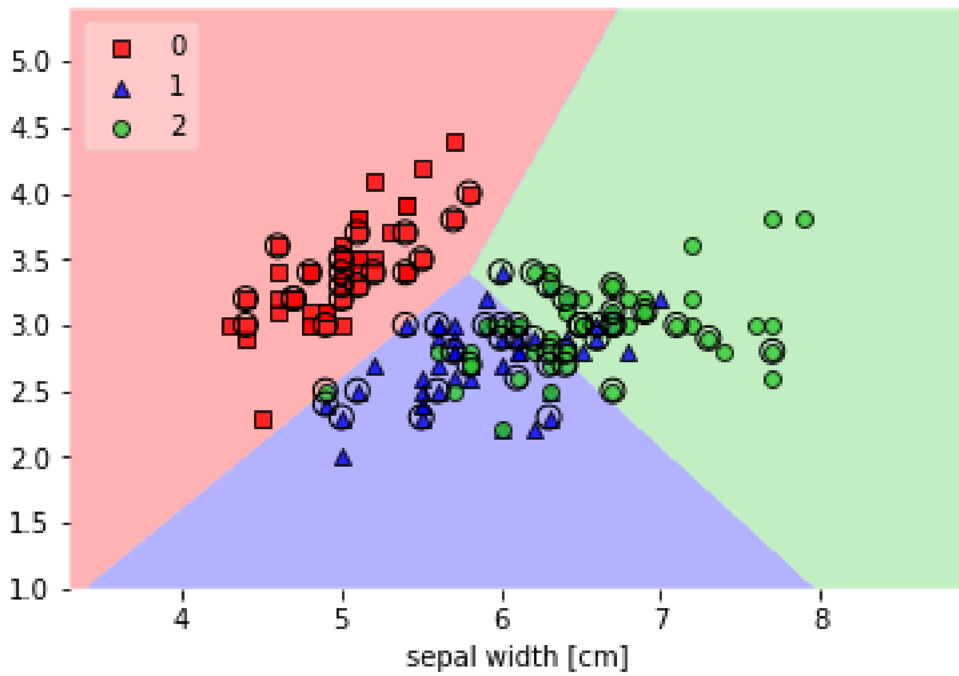
Or we use the classification report function:

```
In [38]: print 'Classification Report:\n', classification_report(y_test, y_pred)
```

```
Classification Report:
             precision    recall  f1-score   support

          0       0.91      1.00      0.95        20
          1       0.68      0.65      0.67        20
          2       0.68      0.65      0.67        20

avg / total       0.76      0.77      0.76        60
```

Finally, we would like to plot the decision regions and our data in order to see how the classifier categorized the events.

```
In [39]: from mlxtend.plotting import plot_decision_regions

         plot_decision_regions(X=X, y=y, clf=lr, X_highlight=X_test, legend=2)
         plt.xlabel('sepal length [cm]')
         plt.xlabel('sepal width [cm]');
```



4.6 K-Nearest Neighbors

4.6.1 Task 6 (Bonus): Perform a classification using K-nearest neighbors classifier, evaluate the performance and show the decision regions.

http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

```
In [40]: from sklearn.neighbors import KNeighborsClassifier

         kn = KNeighborsClassifier(n_neighbors=5)

         kn.fit(X_train, y_train)
         y_pred = kn.predict(X_test)

         print('Accuracy: %.2f' % accuracy_score(y_test, y_pred))
         print("Precision: %.2f" % precision_score(y_test, y_pred, average='weighted'))
         print("Recall: %.2f" % recall_score(y_test, y_pred, average='weighted'))
         print 'Classification Report:\n', classification_report(y_test, y_pred)

         plot_decision_regions(X=X, y=y, clf=kn, X_highlight=X_test, legend=2)
         plt.xlabel('sepal length [cm]')
         plt.xlabel('sepal width [cm]');
```

```
Accuracy: 0.82
Precision: 0.81
```

```
Recall: 0.82
Classification Report:
             precision    recall  f1-score   support

          0       0.95      1.00      0.98        20
          1       0.75      0.75      0.75        20
          2       0.74      0.70      0.72        20

avg / total       0.81      0.82      0.81        60
```