Linear regression

Import all libraries

```
import numpy as np
from sklearn import linear_model, datasets, tree
import matplotlib.pyplot as plt
%matplotlib inline
```

Step 1: Prepare data

First we will prepare some data for demonstrating linear regression. To keep things simple we will assume we have a single input feature. Let us use the following function to generate our data:

$$y = \frac{x}{2} + sin(x) + \epsilon$$

Where $\epsilon \sim \mathcal{N}(0,1)$ is Gaussian noise.

```
number_of_samples = 100
x = np.linspace(-np.pi, np.pi, number_of_samples)
y = 0.5*x+np.sin(x)+np.random.random(x.shape)
plt.scatter(x,y,color='black') #Plot y-vs-x in dots
plt.xlabel('x-input feature')
plt.ylabel('y-target values')
plt.title('Fig 1: Data for linear regression')
plt.show()
```

Step 2: Split the dataset into training, validation and test sets

It is always encouraged in machine learning to split the available data into *training*, *validation* and *test* sets. The training set is supposed to be used to train the model. The model is evaluated on the validation set after every episode of training. The performance on the validation set gives a measure of how good the model *generalizes*. Various hyperparameters of the model are tuned to improve performance on the validation set. Finally when the model is completely optimized and ready for deployment, it is evaluated on the *test data* and the performance is reported in the final description of the model.

In this example we do a 70%−15%−15% random split of the data between the training, validation and test sets respectively.

```
random_indices = np.random.permutation(number_of_samples)
#Training set
x_train = x[random_indices[:70]]
y_train = y[random_indices[:70]]
#Validation set
x_val = x[random_indices[70:85]]
y_val = y[random_indices[70:85]]
#Test set
x_test = x[random_indices[85:]]
y_test = y[random_indices[85:]]
```

Step 3: Fit a line to the data

Linear regression learns to fit a hyperplane to our data in the feature space. For one dimensional data, the hyperplane reduces to a straight line. We will fit a line to our data using **sklearn.linear_model.LinearRegression**.

```
model = linear_model.LinearRegression() #Create a least squared error linear
regression object

#sklearn takes the inputs as matrices. Hence we reshpae the arrays into
column matrices
x_train_for_line_fitting = np.matrix(x_train.reshape(len(x_train),1))
y_train_for_line_fitting = np.matrix(y_train.reshape(len(y_train),1))

#Fit the line to the training data
model.fit(x_train_for_line_fitting, y_train_for_line_fitting)

#Plot the line
plt.scatter(x_train, y_train, color='black')
plt.plot(x.reshape((len(x),1)),model.predict(x.reshape((len(x),1))),color='blue')
plt.xlabel('x-input feature')
plt.ylabel('y-target values')
plt.title('Fig 2: Line fit to training data')
plt.show()
```

Step 4: Evaluate the model

Now that we have our model ready, we must evaluate our model. In a linear regression scenario, its common to evaluate the model in terms of the *mean squared error* on the validation and test sets.

```
mean_val_error = np.mean( (y_val -
model.predict(x_val.reshape(len(x_val),1)))**2 )
mean_test_error = np.mean( (y_test -
model.predict(x_test.reshape(len(x_test),1)))**2 )

print 'Validation MSE: ', mean_val_error, '\nTest MSE: ', mean_test_error
```