```
In [15]: %pylab inline
         import gzip, cPickle
```

```
Welcome to pylab, a matplotlib-based Python environment
[backend: module://IPython.zmq.pylab.backend_inline].
For more information, type 'help(pylab)'.
```

Lab 2: Classification

Machine Learning and Pattern Recognition, September 2013

- The lab exercises should be made in groups of three people, or at most two people.
- The deadline is october 6th (sunday) 23:59.
- Assignment should be sent to N.Hu@uva.nl (Ninhang Hu). The subject line of your email should be "#lab_lastname1_lastname2_lastname3".
- Put your and your teammates' names in the body of the email
- Attach the .IPYNB (IPython Notebook) file containing your code and answers. Naming of the file follows the same rule as the subject line. For example, if the subject line is "lab01_Kingma_Hu", the attached file should be "lab01_Kingma_Hu.ipynb". Only use underscores ("_") to connect names, otherwise the files cannot be parsed.

Notes on implementation:

- You should write your code and answers in an IPython Notebook: http://ipython.org/notebook.html. If you have problems, please contact us.
- Among the first lines of your notebook should be "%pylab inline". This imports all required modules, and your plots will appear inline.
- For this lab, your regression solutions should be in closed form, i.e., should not perform iterative gradient-based optimization but find the exact optimum directly.
- NOTE: Make sure we can run your notebook / scripts!

Part 1. Handwritten digit classification

Scenario: you have a friend with one big problem: she's completely blind. You decided to help her: she has a special smartphone for blind people, and you are going to develop a mobile phone app that can do *machine vision* using the mobile camera: extracting a picture (from the camera) to the meaning of the image. You decide to start with an app that can read handwritten digits, i.e. convert an image of handwritten digits to text (e.g. it would enable her to read precious handwritten phone numbers).

A key building block for such an app would be a function `predict_digit(\bx)` that returns the digit class of an image patch $\mathbf{x}$. Since hand-coding this function is highly non-trivial, you decide to solve this problem using machine learning, such that the internal parameters of this function are automatically learned using machine learning techniques.

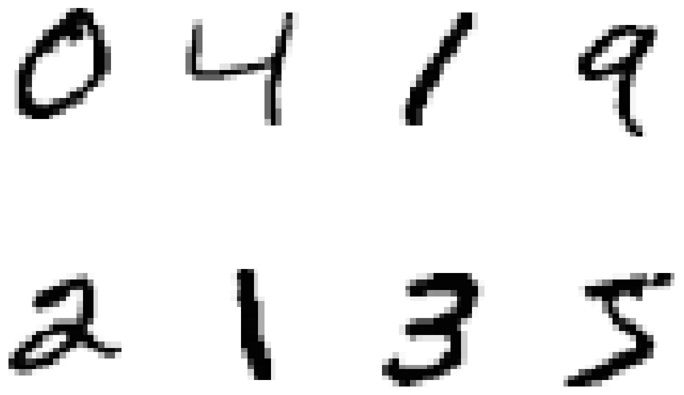
The dataset you're going to use for this is the MNIST handwritten digits dataset (`http://yann.lecun.com/exdb/mnist/`). You can load the data from `mnist.pkl.gz` we provided, using:

```
In [13]: def load_mnist():
             f = gzip.open('mnist.pkl.gz', 'rb')
             data = cPickle.load(f)
             f.close()
             return data

         (x_train, t_train), (x_valid, t_valid), (x_test, t_test) = load_
```

The tuples represent train, validation and test sets. The first element (`x_train`, `x_valid`, `x_test`) of each tuple is a $N \times M$ matrix, where N is the number of datapoints and $M = 28^2 = 784$ is the dimensionality of the data. The second element (`t_train`, `t_valid`, `t_test`) of each tuple is the corresponding N-dimensional vector of integers, containing the true class labels.

Here's a visualisation of the first 8 digits of the trainingset:

```
In [14]: def plot_digits(data, numcols, shape=(28,28)):
             numdigits = data.shape[0]
             numrows = int(numdigits/numcols)
             for i in range(numdigits):
                 plt.subplot(numrows, numcols, i)
                 plt.axis('off')
                 plt.imshow(data[i].reshape(shape), interpolation='neares
             plt.show()

         plot_digits(x_train[0:8], numcols=4)
```



In logistic regression, the conditional probability of class label t given the image $\mathbf{x}$ for some datapoint is given by:

$$\log p(t = j \mid \mathbf{x}, \mathbf{b}, \mathbf{W}) = \frac{q_j}{Z}$$

where $q_j = \exp(\mathbf{w}_j^T \mathbf{x} + b_j)$ (the unnormalized probability of the class j), and $Z = \sum_k q_k$ is the normalizing factor. $\mathbf{w}_j$ is the j-th column of $\mathbf{W}$ corresponding to the class label, b_j is the j-th element of $\mathbf{w}$.

The likelihood of some dataset $(\mathbf{T}, \mathbf{X})$ with N datapoints is given by:

$$\mathcal{L}(\mathbf{b}, \mathbf{W}) = \log p(\mathbf{T} \mid \mathbf{X}, \mathbf{b}, \mathbf{W}) = \sum_{i=1}^{N} \log p(t^{(i)} \mid \mathbf{x}^{(i)}, \mathbf{b}, \mathbf{W})$$

1.1 Gradient-based stochastic optimization

1.1.1 Derive gradient computation equations

First, repeatedly aply the chain rule to derive the equations for computing the gradients. The likelihood w.r.t. a single datapoint is computed by first computing the unnormalized probabilities q_j for each class, then the normalizing factor Z, and finally the normalized probability of the true class label. The gradient computations are in reverse order (as required by the equations resulting from chain rule).

1.1.2 Compute gradients of the log-likelihood

Write a function `logreg_gradient(x, t, w, b)` that returns the gradient (w.r.t. the parameters `w` and `b`) of the log-likelihood for a single datapoint (`x` and `t`).

1.1.3 Stochastic gradient descent

Write a function `sgd_iter(x_train, t_train, w, b)` that performs an iteration of stochastic gradient descent (SGD). It should go through the trainingset once in randomized order, call `logreg_gradient(x, t, w, b)` for each datapoint to get the gradients, and update the parameters using a small learning rate (e.g. `1E-4`). Note that in this case we're maximizing the function, so we should actually performing gradient *ascent* since we're *maximizing* the likelihood function.

1.4. Train

1.4.1 Train

Perform a handful of training iterations through the trainingset. Plot (in one graph) the conditional log-probability of the trainingset and validation set after each iteration.

1.4.2 Visualize weights

Visualize the resulting parameters $\mathbf{W}$ after a few iterations through the training set, by treating each column of $\mathbf{W}$ as an image. If you want, you can use or edit the `plot_digits(...)` above.

14.2. Visualize the 8 hardest and 8 easiest digits

Visualize the 8 digits in the validation set with the highest probability of the true class label under the model. Also plot the 8 digits that were assigned the lowest probability. Ask yourself of these results make sense.