Assignment 2

Submission deadline: classes on **3-6.11.2015**

Points: 13 + 3 bonus

Downloading the starter code

We have provided starter code, available at https://github.com/janchorowski/nn_assignments. Follow the instructions in its README for download instructions. Please email us about any problems with it - we will try to correct them quickly. Also, please do not hesitate to use GitHub's pull requests to send us corrections!

Problem 1. [**3p**] k-NN classifier for larger datasets

Apply the K-Nearest Neighbors (K-NN) algorithm to the MNIST and CIFAR10 datasets.

The MNIST (http://yann.lecun.com/exdb/mnist/) dataset consists of normalized (centered and stretched) scans of hand-written digits. Specifically, each element of the dataset is a 28×28 grayscale image, thus having 764 8-bit pixels.

The CIFAR10 (http://www.cs.toronto.edu/~kriz/cifar.html) dataset consists of small, 32 by 32 pixels, RGB images belonging to 10 categories.

(a) [**1p**] Download and load the MNIST and CIFAR10 datasets. For both datasets, display a few objects from each of the classes, paying attention to aesthetics and clarity of your presentation.

Note: the datasets are available on Lab computers. Please refer to the starter code for instructions on using them.

(b) [**2p**] Apply a k-NN classifier to the MNIST and CIFAR10 datasets.

First, divide the training set into two parts, which we will call training and validation. On MNIST use the first 50000 samples for training and the last 10000 for validation. On CIFAR10, use 40000 to train and 10000 for validation. Then find the optimal number of neighbors by assessing the accuracy on the validation set. You do not need to repeat this experiment multiple times. Finally, compute the accuracy on the test set obtained with the best previously chosen number of neighbors.

On MNIST you should get about 3% errors, while on CIFAR10 you should get about 70% errors. Why CIFAR10 is harder than MNIST? Pick a few mislabeled samples from the test dataset and plot them along with the correct ones.

Note: MNIST and CIFAR10 are much larger than the Iris dataset. A good implementation needs about XXX minutes on Lab computers. Please optimize your algorithm:

- Compute the distances only once, then test for different values of k.
- Use vectorized expressions to compute the distance. It is possible to compute all distances between the training and testing points in one expression. Hint: thing about the vectorized expression $\sqrt{(X-Y)^T \cdot (X-Y)}$.
- You can use single precision numbers in computation.
- If your code is taking a long time o execute, please save its results before the lab session.

Note: in NumPy, matrices have its own data type (`dtype`), which is retained during calculations. Please pay attention, i.e. do not subtract values of data types not having the sign bit, do not divide integers, etc. Results of such operations won't be automatically casted to types having the required precision.

Problem 2. [**3p**] Linear regression

(a) **[1p]** Implement a function generating a dataset of n points according to the following algorithm:

i. Draw n points $x \propto U(0;10)$ (uniformly distributed on $[0,10]$).

ii. Draw n points $y \propto \mathcal{N}(1+20x-1.3x^2, 7)$ (from a Gaussian distribution with $\mu = 1+20x-1.3x^2$ and $\sigma = 7$).

Prepare a dataset of 30 elements and make a scatterplot of the expected value y in function x.

(b) **[1p]** Use linear regression to fit polynomials to the generated dataset. Fit polynomials od degree zero (a constant line), one, two and three. An easy way to do it is to transform each data point x into a vector of its powers $[1, x, x^2, \ldots, x^m]$.

Plot the dataset and all regression curves on one figure.

Note: name "linear regression" signifies, that the hypothesis is linear with respect to parameters Θ. However, the relationship between x and y is not constrained to a linear one. In this exercise it is a polynomial one.

(c) **[1p]** Repeat the previous exercise, this time using the Gradient Descent algorithm to find the optimal Θ (please see the next problem for a description of Gradient Descent).

Problem 3. **[3p]** Gradient Descent (GD) algorithm

Note: the problem has all the information necessary to solve it. GD will be discussed during the 28.10.2015 lecture.

The Gradient Descent (GD) algorithm finds the minimum of a given function by taking small steps along the function's gradient. In pseudocode:

```
$\Theta \leftarrow \Theta_0$
while stop condition not met do
    $\Theta \leftarrow \Theta - \alpha\nabla_\Theta f(\Theta)$
end while
```

where f is the function to minimize, $\nabla_\Theta f(\Theta)$ denotes f's gradient at Θ and α is the step size, taking typically values from $10^{-4}, \ldots, 10^{-1}$.

(a) **[1p]** Implement the GD algorithm as a function:

$$\Theta_{opt} = \mathrm{GD}(f, \Theta_0, \alpha, \rho), \tag{1}$$

where f is a function returning the cost and the gradient of the cost with respect to parameter vector Θ, Θ_0 is the initial value, and α is the step size (a.k.a. the learning rate). You can assume that α remains constant throughout the optimization. Terminate when the function values will differ by less than ρ between subsequent iterations, eg. by 10^{-10}.

(b) **[1p]** Use the GD algorithm to find the optimum of the Rosenbrock (`https://en.wikipedia.org/wiki/Rosenbrock_function`) function. Set $(0,2)$ as the initial point. Try to set an appropriate learning rate α.

Plot the values found by GD at subsequent iterations. Set log scale for the Y axis.

Plot function contours and values of Θ at subsequent iterations.

Note: you can debug your implementation by using the gradient checking routines.

(c) **[1p]** Numerical optimization is of great importance, and many algorithms beside GD exists. Get familiar with the L-BFGS algorithm (for Python: `scipy.optimize.fmin_l_bfgs_b`). Use the L-BFGS algorithm to find the optimum of the Rosenbrock function and plot the contours and Θ's in subsequent iterations.

How many iterations do BGD and L-BFGS need to find a point, for which the Rosenbrock function value is lower than 10^{-10}?

Problem 4. **[2p]** Logistic regression

Note: the problem has all the information necessary to solve it. Logistic regression will be discussed during the 28.10.2015 lecture.

Linear regression is suitable for problems, where the forecasted values are real numbers. We use logistic regression, when we want to label the data with 0 and 1.

Let $x \in \mathbb{R}^n$ be a vector of n real numbers, and $y \in \{0, 1\}$ the given class label. Similarly to what was shown during the lecture, we add an additional element $x_0 = 1$ to vector x, to account for the bias term (and simplify the equations).

Similarly to linear regression, vector $\Theta \in \mathbb{R}^{n+1}$ parametrizes the model (n coefficients describes the data, the remaining one is the intercept). In logistic regression, we model conditional probability that sample x belongs to class 1 as:

$$p(\text{class} = 1|x, \Theta) = h_\Theta(x) = \sigma\left(\sum_{j=0}^{n} \Theta_j x_j\right) = \sigma\left(\Theta^T x\right), \tag{2}$$

where $\sigma(a) = \frac{1}{1+\exp(-a)}$ is being called the logistic sigmoid (a function, which plot is s-curved).

An unknown sample x is being labeled 1 if $h_\Theta(x) \geq 0.5$, or equivalently, $\Theta^T x \geq 0$.

Classification mismatch between the forecasted values and the data is being measured most of the time with cross-entropy:

$$J(\Theta) = -\sum_{i=1}^{m} y^{(i)} \log\left(h_\Theta(x^{(i)})\right) + (1 - y^{(i)}) \log\left(1 - h_\Theta(x^{(i)})\right), \tag{3}$$

assuming $0 \log(0) = 0$.

Exercieses

(a) **[0.5p]** Plot the function $\sigma(a) = \frac{1}{1+\exp(-a)}$.

(b) **[1.5p]** Use logistic regression to distinguish *Versicolor* and *Virginica* irises. Use only the `petal length` and `petal width` features. Use either Gradient Descent, or L-BFGS to solve for the optimal Θ. Prepare the scatterplot of the data and plot the class separation boundary found by logistic regression.

Problem 5. **[2p]** Locomotive problem

While on a walk, you notice that a locomotive has the serial number 50. Assuming, that all locomotives used by PKP (the polish railroad operator) are numbered using consecutive natural numbers, what is your estimate of N the total number of locomotives operated by PKP?

Tell why the Maximum Likelihood principle may not yield satisfactory results.

Use the Bayesian approach to find the posterior distribution over the number of locomotives. Then compute the expected count of locomotives. For the prior use the power-law:

$$p(N) = \frac{1}{N^\alpha} \frac{1}{\zeta(\alpha, 1)},$$

where the $\zeta(s, q) = \sum_{n=0}^{\infty} \frac{1}{(q+n)^s}$ is the Hurvitz Zeta function (`https://en.wikipedia.org/wiki/Hurwitz_zeta_function`) available in Python as `scipy.special.zeta`. The use of the power law is motivated by the observation that the frequency of occurrence of a company is inversely proportional to its size (see also: R.L. Axtell, Zipf distribution of US firm sizes `https://www.sciencemag.org/content/293/5536/1818`).

How would your estimate change after seeing 5 locomotives, with the biggest serial number among them being 50?

Note: during the Second World War, a similar problem was encountered while trying to estimate the total German tank production from the serial number of captured machines. The statistical estimates were the most precise!

Problem 6. [**3p bonus**] Quantile Regression

This exercise will be posted soon, sorry for the delay!