582744 Advanced Course in Machine Learning

Exercise 2 **Due April 5, 10:00 AM**

Rules:

1. Your submission is composed of two parts: (a) A single PDF file containing your answers to all questions, including both the pen and paper questions and the written answers and various plots for the programming questions. (b) a single compressed file (zip or tar.gz) containing your code (and nothing else). If your code is in a single file, it can be sent also a plain source code.
2. The submission should be sent directly to BOTH Arto (`arto.klami@cs.helsinki.fi`) and Aditya (`aditya.jitta@cs.helsinki.fi`).
3. All material must be renamed to your student ID. Always mention your name and student ID also in the written report.
4. The subject field of your email should be [AML][your student ID][exercise 2].
5. Please typeset your work using appropriate software such as LaTeX. However, there is no need to typeset the pen and paper answers – you can also include a scanned hand-written version.
6. The pen and paper exercises can alternatively be returned in paper form during the Tuesday lecture.

This set of exercises is due on Tuesday April 5th, before 10:00 AM.

1 Derivatives, gradients and all that (6pts)

Logistic regression is a model for binary classification. It states that the probability of the positive class label is given by $s(\boldsymbol{\theta}^T\mathbf{x})$, where $s(z)$ is the *logistic function* (or sigmoid function):

$$s(z) = \frac{1}{1+e^{-z}}.$$

What does it look like? Sketch the curve by hand.

The logarithmic loss for a single data point (input $\mathbf{x} \in \mathbb{R}^D$ and output $y \in [0,1]$) is

$$L((\mathbf{x}, y), \boldsymbol{\theta}) = y \log s(\boldsymbol{\theta}^T\mathbf{x}) + (1-y)\log(1 - s(\boldsymbol{\theta}^T\mathbf{x})).$$

Where does this come from?

Given the loss function above, compute

(a) The gradient $\nabla L((x,y),\boldsymbol{\theta}) = [\frac{\partial L}{\partial \theta_1}, \dots, \frac{\partial L}{\partial \theta_d}]^T$.

(b) The Hessian matrix containing all second derivatives

$$H(L((\mathbf{x},y),\boldsymbol{\theta})) = \begin{bmatrix} \frac{\partial^2 L}{\partial\theta_1\partial\theta_1} & \frac{\partial^2 L}{\partial\theta_1\partial\theta_2} & \cdots & \frac{\partial^2 L}{\partial\theta_1\partial\theta_D} \\ \frac{\partial^2 L}{\partial\theta_2\partial\theta_1} & \frac{\partial^2 L}{\partial\theta_2\partial\theta_2} & \cdots & \frac{\partial^2 L}{\partial\theta_2\partial\theta_D} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 L}{\partial\theta_D\partial\theta_1} & \frac{\partial^2 L}{\partial\theta_D\partial\theta_2} & \cdots & \frac{\partial^2 L}{\partial\theta_D\partial\theta_D} \end{bmatrix}.$$

(c) What are the numerical values for the gradient and Hessian if $\boldsymbol{\theta} = [2,4]$, $\mathbf{x} = [-1,3]$ and $y = 1$?

Hints: Remember the chain rule of derivation. It probably helps if you first compute the derivative $\frac{ds(z)}{dz}$ of the sigmoid function for an arbitrary input and simplify the result as much as possible.

2 Training and generalization error (programming, 9 pts)

Read in the data matrix $\mathbf{X}$ given in http://www.cs.helsinki.fi/u/aklami/teaching/AML/exercise_2_2_train.csv. Each row is a data point of $D = 20$ dimensions and there are in total $N = 200$ data points. The last element on each row is a scalar output value.

In this exercise we will study a simple linear regression model on this data. The model is parameterized by $\boldsymbol{\theta} \in \mathbb{R}^D$ and the loss function is given by $L(\boldsymbol{\theta}) = \frac{1}{N}\sum_{n=1}^N(\boldsymbol{\theta}^T\mathbf{x}_n - y_n)^2$. We will also consider regularizers of the form $g(\boldsymbol{\theta}) = \boldsymbol{\theta}^T\boldsymbol{\theta}$. The gradients of these two are given by

$$\nabla L = \frac{2}{N}\sum_n(\boldsymbol{\theta}^T\mathbf{x}_n - y_n)\mathbf{x}_n,$$
$$\nabla g = 2\boldsymbol{\theta}.$$

Start by implementing the standard gradient descent method (Section 8.3.2) for this problem. For this you need to write a function that computes the loss for a given set of data points and the parameters of the model, a function that evaluates the gradient for given parameters, and the actual gradient descent loop. For this exercise, simply use the fixed step-size 0.5 and start with the initial solution of $\boldsymbol{\theta} = 0$.

We will now study alternative techniques for avoiding overfitting. You will produce three alternative estimates for the parameters and compare the resulting errors on a separate test set not available during training.

(a) *Empirical risk minimization*: Minimize the empirical risk for the first 20 samples of the data set. Run the algorithm until convergence (the gradient is close enough to zero). How many iterations did this take? Store the resulting parameter vector $\boldsymbol{\theta}_{\text{ERM}}$.

(b) *Early-stopping*: Minimize the empirical risk for the same subset of 20 samples, but after each iteration compute also the validation loss on the remaining $N - 20 = 180$ samples. Stop the iteration when the validation loss no longer decreases. How many iterations did this take? Store the resulting parameter vector $\boldsymbol{\theta}_{\text{early}}$.

(c) *Regularization*: Minimize the regularized risk for the same set of samples, now including also the regularization term $\lambda g(\boldsymbol{\theta})$ as part of the cost function. Learn the model until convergence (of the regularized training loss) for different values of $\lambda \in \{0.001, 0.003, 0.01, 0.03, 0.1\}$ and plot the validation loss (without the regularization part) as a function of λ. Which value gives the best accuracy? Store the parameter vector $\boldsymbol{\theta}_{\text{RRM}}$ corresponding to that choice.

Now you have three alternative solutions for the model. Evaluate their accuracy on the test data provided in file http://www.cs.helsinki.fi/u/aklami/teaching/AML/exercise_2_2_test.csv. The data is in the same format as the training data, but has 1000 samples. Which of the three solutions is the best? Are the other solutions good as well?

Above we used 10% of the available data for training and 90% for validation. Repeat the same process with 50/50 split and 90/10 split (that is, use the first 100 and 180 samples for training, respectively). How do the results change? Which of these splits would you use in future? Why?

3 Comparison of optimization methods (9 pts)

We use the same model as in Problem 2, but now fit it into a simple two-dimensional data set provided in http://www.cs.helsinki.fi/u/aklami/teaching/AML/exercise_2_3.csv. Again the last element on each row is the output value. No regularization is used in this case.

We will now compare three optimization algorithms for minimizing the empirical risk. For each method start from the parameter values $\boldsymbol{\theta} = [-0.3, 1.5]$ and compare the following algorithms:

1. *Gradient descent with fixed step size*: Use the algorithm you implemented in Problem 2, trying out alternative values for the step-size. Can you find a good one?

2. *Newton's method (Section 8.3.3)*: The Hessian matrix for this loss is constant $H = \frac{2}{N} X^T X$ (a 2×2 matrix). Instead of the regular gradient updates, use the update scheme $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - H^{-1} \nabla L(\boldsymbol{\theta})$. How many iterations do we need for convergence? Why?

3. *Stochastic gradient descent (Section 8.5.2)*: The stochastic gradient descent computes the gradient only over a subset of the data points. Furthermore, we will try out an adaptive step-size algorithm called *adaGrad* that works as follows:

   - For each iteration t, store the gradient $g_t = \nabla L$
   - For each dimension d of the parameter vector, compute $s_t(d) = \sum_{i=1}^{t} g_i(d)^2$
   - The step-size at iteration t for each dimension is given by $\frac{\eta}{\tau + \sqrt{s_t(d)}}$, where you can use $\eta = 1$ and $\tau = 1$ (or some other values if you wish to play around).

   Try the algorithm so that you always pick just one sample for estimating the gradient. Try it out also with *mini-batch* of 50 randomly chosen samples. How do these differ?

To understand what the algorithms do you can plot the cost function as a function of the iteration, but since the problem is two-dimensional we can also track how the actual parameter values change. The following python code produces a contour plot of the loss surface (assuming you have implemented the cost function). You can overlay the progress of the algorithm on top of this plot.

```
import numpy as np
import matplotlib.pyplot as plt
I = np.arange(-2.0, 2, 0.05)
J = np.arange(-2.0, 2, 0.05)

Z = np.zeros((len(I),len(J)))

for i in range(len(I)):
    for j in range(len(J)):
        theta = [I[i], J[j]]
        Z[i,j] = cost_function(theta, X, Y)

levels=np.arange(0.1,1.2,0.3)
I1, J1 = np.meshgrid(I, J)
plt.contour(I1, J1, Z.T, levels)

#
# Plot the progress of the algorithm here
#

plt.show()
```