Advanced Machine Learning (GR5242)
Fall 2017

Homework 1

Due: Thurday 28 September, at 4pm (for both sections of the class)

Homework submission: Please submit your homework by publishing a notebook that cleanly displays your code, results, and plots to pdf or html.

For conciseness, we assume that you have imported the `numpy` package as `np`, and the `matplotlib.pyplot` package as `plt`.

Problem 1 (Sampling a d-dimensional Gaussian distribution)

We consider the problem of sampling a multivariate normal (Gaussian) distribution. The `np.random` library provides a function called `randn`, which by default produces pseudo-random samples of an i.i.d normal. We wish to produce samples from a Gaussian $N(\mu, \Sigma)$ with arbitrary parameters μ and Σ, so we have to transform the sample in a suitable manner.

Our approach is based on the eigenvalue structure of symmetric matrices: The eigenvectors of a full-rank symmetric matrix form an orthogonal basis of the underlying vector space. With respect to this basis, the matrix is diagonal, with the eigenvalues as diagonal entries. Denote this diagonal matrix of eigenvalues D and the matrix describing the change of basis V. Then

$$\Sigma = VDV^{-1}$$

V is orthogonal (since it describes a change of basis between two orthonormal bases), so $V^{-1} = V^T$ and $\Sigma = VDV^T$. This representation of Σ is called the Schur decomposition. We can produce a sample from a normal distribution with parameters (μ, Σ) by drawing a sample vector g from $(0, 1)$ using multivariate_normal, changing basis, and adding the expectation vector:

$$\tilde{g} = V\sqrt{D}g + \mu$$

As you will recall from linear algebra, $\sqrt{D} = \text{diag}(\sqrt{D_{ii}})$.

a) Implement a function to produce `n` draws from a d-dimensional Gaussian.

```
def GSAMPLE(mu, Sigma, n):
# Your code here
return(G)  # an n by d matrix of d-dim gaussian vectors
```

The ouput should be an `np.ndarray` of size `(n, d)`. (The dimension d is implicitly specified by the size of the arguments `mu` and `Sigma`.)

b) Verify that your function works as intended by applying the functions `np.mean` and `np.cov` to a sufficiently large sample generated by your program. They should approximately reproduce your input parameters.

c) Produce 100 samples each in two and three dimensions, using the parameter values $\mu = (1, 1)^T$, $\Sigma = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$ and $\mu = (1, 1, 1)^T$, $\Sigma = \begin{pmatrix} 5 & 2 & 0 \\ 2 & 3 & 1 \\ 0 & 1 & 1 \end{pmatrix}$, respectively. Plot your results using the `plt.scatter` function.

Hint: For the 3D plot you will find it helpful to include the line

```
from mpl_toolkits.mplot3d import axes3d
```

with the rest of your imports and create your plot axes with the line

```
ax = fig.add_subplot(111, projection='3d').
```

Problem 2 (Sample size)

This problem visualizes the effects of small sample size. We will draw a number n of sample points from a one-dimensional Gaussian, sort them into a histogram, and see how stable the result is with respect to different samples. The basic procedure is the following:

- Choose a number n of sample points and a number N_{bins} of histogram bins.
- Use the python function `np.random.randn` to draw n samples from a one-dimensional Gaussian distribution ($\mu = 0, \sigma = 1$).
- Turn the complete data sample into a histogram using the function `plt.hist`

Please complete the following steps:

a) Produce four histograms with $n = 100$ and $N_{\text{bins}} = 10$. Plot the histograms using the `plt.hist` function.

b) Repeat the procedure with $n = 100000$.

c) For $n = 100000$, plot one histogram each for $N_{\text{bins}} \in \{10, 100, 1000\}$. (matplotlib lets you display multiple plots within a single figure using the `plt.subplot` function.)

d) Finally, choose $n = 100$ and $N_{\text{bins}} = 1000$.

e) Give a brief discussion of the results. Remember, this is about sample size and reliability of estimates.

Please submit your plots and discussion. Make sure that plots are adequately labeled.

Problem 3 (Gradient descent for logistic regression)

In this problem, we solve a logistic regression problem. Recall that we discussed the computation of a maximum likelihood solution in class, where this solution was obtained using Newton's algorithm. Here, we will simplify things a little and use gradient descent instead.

a) First, generate some data: Draw 100 samples from a bivariate normal with mean $(0, -1)$ and covariance $\begin{pmatrix} 1 & 0.5 \\ 0.5 & 1 \end{pmatrix}$, and label such points 0. Generate an additional 100 samples from a bivariate normal with mean $(0, 1)$ and covariance $\begin{pmatrix} 1 & -0.5 \\ -0.5 & 1 \end{pmatrix}$. Label these points 1. Create your design matrix X and labels y. Visualize the generated data on a scatter plot by coloring points by their label.

b) Write a function `loss(X, y, w)` that computes the negative log-likelihood, and a function `grad(X, y, w)` that computes the gradient of the negative log-likelihood at the parameter w. You may find the function `scipy.special.expit` helpful (it computes the sigmoid function).

c) Run gradient descent starting with the point $w = (0, 0)$, and step size 0.001. Stop when the norm of the gradient is less than 0.001. Verify that your final loss is less than your initial loss.