EE-UY/CS-UY 4563: Introduction to Machine Learning
Midterm 2, Fall 2017

Answer all THREE questions. Exam is closed book. No electronic aids. But, you are permitted a limited number of cheat sheets. Part marks are given. If you do not remember a particular python command or its syntax, use pseudo-code and state what syntax you are assuming.

Best of luck!

1. Given a time-sequence x_i, $i = 0, 1, \ldots, T-1$, suppose we try to fit a model,

$$x_i \approx \hat{x}_i = \phi\left(\sum_{j=1}^{p} w_j x_{i-j}\right), \quad \phi(z_i) = \frac{1}{1+e^{-z_i}},$$

where $\mathbf{w} = (w_1, \ldots, w_p)$ are the unknown parameters. Assume $x_k = 0$ for $k < 0$. To fit the model, we use a loss function,

$$J(\mathbf{w}) = \sum_{i=1}^{T-1} (\widehat{x}_i - x_i)^2.$$

(a) Find a matrix $\mathbf{A}$ and vector $\mathbf{z}$ such that $\mathbf{z} = \mathbf{A}\mathbf{w}$ and

$$\hat{x}_i = \phi(z_i), \quad i = 1, \ldots, T-1.$$

(b) What is the gradient $\nabla_{\mathbf{w}} J(\mathbf{w})$?

(c) Suppose that $\mathbf{w}^0 = (w_1^0, \ldots, w_p^0)$ and $\mathbf{w}^1 = (w_1^1, \ldots, w_p^1)$ are two parameter vectors such that,

$$w_1^1 = w_1^0 + \delta,$$
$$w_j^1 = w_j^0, \text{ for } j = 2, 3, \ldots, p.$$

That is, the two parameters are equal except for the first coordinate. Approximately, what is $J(\mathbf{w}^1) - J(\mathbf{w}^0)$ when δ is small? Use a linear approximation and leave your answer in terms of the coefficients of the gradient $\nabla J(\mathbf{w})$.

(d) Write a short python function to implement the following variant of gradient descent:

$$\mathbf{v}^k = \mathbf{w}^k - \alpha \nabla_{\mathbf{w}} J(\mathbf{w}^k),$$
$$\mathbf{w}^{k+1} = \max\{0, \mathbf{v}^k\}.$$

In the second step, the maximum is applied elementwise, $\hat{w}_j^{k+1} = \max\{0, v_j^k\}$. This algorithm is called *projected gradient descent*.

Your function should work for an *arbitrary* loss function $J(\mathbf{w})$ – not necessarily one of the loss functions above. To do this, use the format,

```python
def proj_grad(feval,...):
    ...
```

which takes a function argument `feval` that returns the loss function and gradient (i.e. `J, Jgrad = feval(w)`). Add any other arguments and make any other assumptions as necessary. The function should return the final estimate $\mathbf{w}^k$ and loss function $J(\mathbf{w}^k)$.

2. You are given four data points $\mathbf{x}_i$ with binary class labels $y_i = \pm 1$:

x_{i1}	0	0	2	3
x_{i2}	0	2	2	0
y_i	-1	1	-1	1

(a) Draw a scatter plot of the four data points indicating the two classes in different markers.

(b) Find a weight, $\mathbf{w} = (w_1, w_2)$, and bias, b, such that the linear classifier,

$$\hat{y} = \begin{cases} 1, & \text{if } z > 0 \\ -1 & \text{if } z < 0. \end{cases} \qquad z = b + w_1 x_1 + w_2 x_2.$$

makes a minimum number of errors on the training data.

(c) Consider the SVM loss,

$$J(\mathbf{w}, b) := C \sum_{i=1}^{N} \epsilon_i + \frac{1}{2}\|\mathbf{w}\|^2, \quad \epsilon_i = \max\{0, 1 - y_i z_i\},$$

for some $C > 0$. For your classifier, which sample $(\mathbf{x}_i, y_i)$ has the largest ϵ_i? What is the value of ϵ_i?

Note: You do not need to compute ϵ_i for all the samples. Think about the sample that will have the highest value.

(d) Now, consider an SVM classifier,

$$\hat{y} = \begin{cases} 1, & \text{if } z > 0 \\ -1 & \text{if } z < 0, \end{cases} \qquad z = \sum_{i=1}^{N} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}),$$

using a radial basis function,

$$K(\mathbf{x}_i, \mathbf{x}) = e^{-\gamma\|\mathbf{x}_i - \mathbf{x}\|^2}, \quad \|\mathbf{x}_i - \mathbf{x}\|^2 = \sum_{j=1}^{d}(x_{ij} - x_j)^2.$$

Write a python method `predict`, that outputs $\hat{y}$ for a single data point $\mathbf{x}$:

```python
def predict(x,...):
   return yhat
```

You will need to supply your method `predict` any arguments in addition to $\mathbf{x}$ that you will need. You can assume the input dimension is $d = 2$.

3. Consider a neural network used for binary classification of the form,

$$z_j^{\mathrm{H}} = \sum_{k=1}^{N_i} W_{jk}^{\mathrm{H}} x_k + b_j^{\mathrm{H}}, \quad u_j^{\mathrm{H}} = \begin{cases} 1, & \text{if } z_j^{\mathrm{H}} > 0, \\ 0, & \text{if } z_j^{\mathrm{H}} < 0. \end{cases}, \quad j = 1, \ldots, N_h$$

$$z^{\mathrm{O}} = \sum_{k=1}^{N_h} W_k^{\mathrm{O}} u_k^{\mathrm{H}} + b^{\mathrm{O}}, \quad P(y=1|\mathbf{x}) = \frac{1}{1+e^{-z^{\mathrm{O}}}}.$$

The hidden weights and biases are:

$$\mathbf{W}^{\mathrm{H}} = \begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \mathbf{b}^{\mathrm{H}} = \begin{bmatrix} -2 \\ 4 \\ -1 \end{bmatrix}.$$

(a) Draw the region on the (x_1, x_2) plane where $u_j^{\mathrm{H}} = 1$ for all j.

(b) Suppose that the output weight vector is $\mathbf{W}^{\mathrm{O}} = [1, 1, 1]$. For what range of values b^{O} is $\hat{y} = 1$ when $\mathbf{x} = (3, 2)$.

(c) Consider the problem of computing the gradient of a loss function J on a mini-batch $(\mathbf{x}_i, y_i)$, $i = 1, \ldots, N$. In back-propagation, suppose that we have computed the gradients, $\partial J/\partial z_{ij}^{\mathrm{H}}$. Show how to compute the gradients $\partial J/\partial W_{jk}^{\mathrm{H}}$.

(d) Write a few lines of python code to implement the gradient calculation in part (c). State your assumptions on how you represent the inputs and outputs for the calculation.