# Machine Learning Assignment 6

Shivangi Aneja

4-December-2017

**Problem 1**

**(1)** $f(x,y,z) = 3x + e^{y+z} - min\{-x^2, log(y)\}$ and $D = (-100,100) \times (1,50) \times (10,20)$
$\Rightarrow f(x,y,z) = 3x + e^{y+z} + max\{x^2, -log(y)\}$ $\because \{min(a,b) = -max(-a,-b)$
Split the function into 3 parts :
(a)$f_1(x) = 3x$ and $D = (-100,100)$
$f_1(x)$ is a linear function and a linear function is both convex and concave function
(b)$f_2(y,z) = e^{y+z}$ and $D = (1,50) \times (10,20)$
The function has min $= e^{1+10} = e^{11}$ and max $= e^{50+20} = e^{70}$. It takes no negative value.The functions $e^{y+z}$ is a convex , thus $f_2(y,z)$ is a convex function
(c)$f_3(x,y) = max\{x^2, -log(y)\}$ and $D = (-100,100) \times (1,50)$
Check convexity for $g(x) = x^2$.
For convexity we know that, $g(\lambda x_1 + (1-\lambda)x_2) \leq \lambda g(x_1) + (1-\lambda)g(x_2)$ for $x_1, x_2 \in (dom\ g)\ and\ 0 \leq \lambda \leq 1$
Substituting values in function , we get :
$(\lambda x_1 + (1-\lambda)x_2)^2 = \lambda^2 x_1^2 + 2\lambda(1-\lambda)x_1x_2 + (1-\lambda)^2 x_2^2 \leq \lambda x_1^2 + (1-\lambda)x_2^2$
$\Rightarrow \lambda x_1^2(\lambda - 1) + 2\lambda(1-\lambda)x_1x_2 - (1-\lambda)\lambda x_2^2 \leq 0$
$\lambda(1-\lambda)(2x_1x_2 - x_1^2 - y_1^2) \leq 0$
$-\lambda(1-\lambda)(x_1 - x_2)^2 \leq 0$ and this is true. Hence $g(x) = x^2$ is a convex function.
Also we know that $-log(y)$ is a convex function.
Thus $f_3(x,y) = max\{x^2, -log(y)\}$ is convex $\because \{max(f_a, f_b) = Convex$ if $f_a$ and $f_b$ are convex To summarize, we have

| Function | Convexity |
|---|---|
| $f_1(x) = 3x$ | Convex |
| $f_2(y,z) = e^{y+z}$ | Convex |
| $f_3(x,y) = max\{x^2, -log(y)\}$ | Convex |

$\because f(x,y,z) = f_1(x) + f_2(y,z) + f_3(x,y) = ConvexFunction$ if $f_1(x), f_2(y,z)$ and $f_3(x,y)$ are Convex Function
$\boxed{Thus\ f(x,y,z)\ is\ a\ Convex\ Function}$

**(2)** $f(x,y) = yx^3 - 2yx^2 + y + 4$ and $D = (-10,10) \times (-10,10)$
$\frac{\partial f}{\partial x} = 3x^2y - 4xy$
$\frac{\partial f}{\partial y} = x^3 - 2x^2 + 1$

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix} = \begin{bmatrix} 6xy - 4y & 3x^2 - 4x \\ 3x^2 - 4x & 0 \end{bmatrix}$$

The Hessian Matrix of $f(x, y)$ is not positive semidefinite.

Hence, $f(x, y)$ is not a Convex function

**(3)** $f(x) = log(x) + x^3$ and $D = (1, \infty)$

Splitting into 2 parts , we get :

(a) $f_1(x) = log(x)$. We know that $log(x)$ is a concave function.

(b) $f_2(x) = x^3 \Rightarrow f_2'(x) = 3x^2$ , $f_2''(x) = 6x$. The second derivative is positive in the domain of $f(x)$. Thus , it is a convex function

| Function | Convexity |
|---|---|
| $f_1(x) = log(x)$ | Concave |
| $f_2(x) = x^3$ | Convex |

Sum of convex and concave function could either be convex or concave. $f(x) = log(x) + x^3$

$f'(x) = \frac{1}{x} + 3x^2$

$f''(x) = \frac{-1}{x^2} + 6x$. Minimum value = 5 and maximum value = $\infty$. Thus , $f''(x) \geq 0$. Thus , it is a convex function

This is Convex Function

**4)** $f(x) = -min\{2log(2x), -x^2 + 4x - 32\}$ and $D = \mathbb{R}^+$

$\Rightarrow f(x) = max\{-2log(2x), x^2 - 4x + 32\}$

Splitting into 2 parts , we get :

(a) $f_1(x) = -2log(2x)$. We know that $-log(x)$ is a convex function.

(b) $f_2(x) = x^2 - 4x + 32$

$\Rightarrow f_2'(x) = 2x - 4$ , $f_2''(x) = 2$. The second derivative is positive in the domain of $f(x)$. Thus , it is a convex function

| Function | Convexity |
|---|---|
| $f_1(x) = -2log(2x)$ | Convex |
| $f_2(x) = x^2 - 4x + 32$ | Convex |

*Thus $f(x)$ is a Convex Function*

**Problem 2**
Given , $f_1 : \mathbb{R}^d \to \mathbb{R};\ f_2 : \mathbb{R}^d \to \mathbb{R}$ are convex functions
To prove, $h(x) := f_1(x) + f_2(x)$ is a Convex function
We know for a Convex function we have ,
$\forall x, y \in \mathbb{R}^d$ and $0 \leq \lambda \leq 1$,
$f_1(\lambda x + (1-\lambda)y) \leq \lambda f_1(x) + (1-\lambda) f_1(y)$.....(1)
$f_2(\lambda x + (1-\lambda)y) \leq \lambda f_2(x) + (1-\lambda) f_2(y)$.....(2)
Given , $h(x) := f_1(x) + f_2(x)$ .....(3)
Adding (1) and (2), we get,
$f_1(\lambda x + (1-\lambda)y) + f_2(\lambda x + (1-\lambda)y) \leq \lambda f_1(x) + (1-\lambda) f_1(y) + \lambda f_2(x) + (1-\lambda) f_2(y)$
$\Rightarrow f_1(\lambda x + (1-\lambda)y) + f_2(\lambda x + (1-\lambda)y) \leq \lambda\{f_1(x) + f_2(x)\} + (1-\lambda)\{f_1(y) + f_2(y)\}$
Using (3), we have ,
$\Rightarrow \boxed{h(\lambda x + (1-\lambda)y) \leq \lambda h(x) + (1-\lambda) h(y)}$. This is the required condition for convexity of a function.
Hence $h(x) := f_1(x) + f_2(x)$ is a convex function.
Hence Proved

**Problem 3**
Given , $f_1 : \mathbb{R} \to \mathbb{R};\ f_2 : \mathbb{R} \to \mathbb{R}$ are convex functions
Check , $h(x) := f_1(x) f_2(x)$ is a Convex function or not ?

$h(x) = f_1(x) f_2(x)$
$\Rightarrow h'(x) = f_1(x) f_2'(x) + f_1'(x) f_2(x)$
$\Rightarrow h''(x) = f_1(x) f_2''(x) + f_1''(x) f_2(x) + 2 f_1'(x) f_2'(x)$

$f_1''(x), f_2''(x) \geq 0$ because both $f_1(x), f_2(x)$ are convex functions.

**Case (a)**: When $f_1(x)$ and $f_2(x)$ are positive and non decreasing functions
$f_1(x) \geq 0, f_1'(x) \geq 0$ and $f_2(x) \geq 0, f_2'(x) \geq 0$
Also, this is a Convex function, so we have $f_1''(x) \geq 0$ and $f_2''(x) \geq 0$
$\Rightarrow f_1(x) f_2''(x) \geq 0$ ; $f_1''(x) f_2(x) \geq 0$ ; $2 f_1'(x) f_2'(x) \geq 0$
$\Rightarrow h''(x) \geq 0$
It is a convex function in this case

**Case (b)**: When $f_1(x)$ and $f_2(x)$ are not positive
$f_1(x) \leq 0$ and $f_2(x) \leq 0$
Also, this is a Convex function, so we have $f_1''(x) \geq 0$ and $f_2''(x) \geq 0$
But $f_1'(x)$ and $f_2'(x)$ can be positive or negative.

So $h''(x)$ can be positive or negative depending on the value of $f_1'(x)$ and $f_2'(x)$
It may or may not be convex function in this case

Thus, to conclude $h(x) = f_1(x)f_2(x)$ is not always a Convex Function

**Problem 4**

Let $f : \mathbb{R}^N \to \mathbb{R}$ be a convex function.

To prove : If $\nabla f(\theta) = 0$ then $\theta$ is global minima

From the first order characterization of convexity, we have
$f(y) \geq f(x) + \nabla f^T(x)(y - x), \forall x, y$

Specifically, substitute $x = \theta$ , we have
$f(y) \geq f(\theta) + \nabla f^T(\theta)(y - \theta), \forall y$
Since $\nabla f^T(\theta) = 0$, we get
$f(y) \geq f(\theta), \forall y$
Thus, for a convex function if $\nabla f(\theta) = 0$, then $\theta$ is global minima.
Hence Proved

**Problem 5**

Python file is attached on the next page

December 3, 2017

# 1 Programming assignment 6: Optimization: Logistic regression

```
In [3]: import numpy as np
        import matplotlib.pyplot as plt
        %matplotlib inline

        from sklearn.datasets import load_breast_cancer
        from sklearn.model_selection import train_test_split
        from sklearn.metrics import accuracy_score, f1_score
```

## 1.1 Your task

In this notebook code skeleton for performing logistic regression with gradient descent is given. Your task is to complete the functions where required. You are only allowed to use built-in Python functions, as well as any `numpy` functions. No other libraries / imports are allowed.

For numerical reasons, we actually minimize the following loss function

$$\mathcal{L}(\mathbf{w}) = \frac{1}{N} NLL(\mathbf{w}) + \frac{1}{2}\lambda ||\mathbf{w}||_2^2$$

where $NLL(\mathbf{w})$ is the negative log-likelihood function, as defined in the lecture (Eq. 33)

## 1.2 Load and preprocess the data

In this assignment we will work with the UCI ML Breast Cancer Wisconsin (Diagnostic) dataset https://goo.gl/U2Uwz2.

Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image. There are 212 malignant examples and 357 benign examples.

```
In [4]: X, y = load_breast_cancer(return_X_y=True)

        # Add a vector of ones to the data matrix to absorb the bias term
        X = np.hstack([np.ones([X.shape[0], 1]), X])

        # Set the random seed so that we have reproducible experiments
        np.random.seed(123)

        # Split into train and test
        test_size = 0.3
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size)
```

### 1.3 Task 1: Implement the sigmoid function

```
In [5]: def sigmoid(t):
            """
            Applies the sigmoid function elementwise to the input data.

            Parameters
            ----------
            t : array, arbitrary shape
                Input data.

            Returns
            -------
            t_sigmoid : array, arbitrary shape.
                Data after applying the sigmoid function.
            """
            # TODO
            t_sigmoid = 1 / (1 + np.exp(-t))
            return t_sigmoid
```

### 1.4 Task 2: Implement the negative log likelihood

As defined in Eq. 33

```
In [6]: def negative_log_likelihood(X, y, w):
            """
            Negative Log Likelihood of the Logistic Regression.

            Parameters
            ----------
            X : array, shape [N, D]
                (Augmented) feature matrix.
            y : array, shape [N]
                Classification targets.
            w : array, shape [D]
                Regression coefficients (w[0] is the bias term).

            Returns
            -------
            nll : float
                The negative log likelihood.
            """
            # TODO
            a = (w*X).sum(axis=1)
            eps = 1e-15
            sig = sigmoid(a)
            ln1 = np.log2(sig + eps)
            ln2 = np.log2(1-sig+eps)
```

```
    nll = -((y*ln1 + (1-y)*ln2).sum(axis=0))
    return nll
```

#### 1.4.1 Computing the loss function $\mathcal{L}(\mathbf{w})$ (nothing to do here)

```
In [7]: def compute_loss(X, y, w, lmbda):
    """
    Negative Log Likelihood of the Logistic Regression.

    Parameters
    ----------
    X : array, shape [N, D]
        (Augmented) feature matrix.
    y : array, shape [N]
        Classification targets.
    w : array, shape [D]
        Regression coefficients (w[0] is the bias term).
    lmbda : float
        L2 regularization strength.

    Returns
    -------
    loss : float
        Loss of the regularized logistic regression model.
    """
    # The bias term w[0] is not regularized by convention
    return negative_log_likelihood(X, y, w) / len(y) + lmbda * np.linalg.norm(w[1:])**2
```

### 1.5 Task 3: Implement the gradient $\nabla_{\mathbf{w}}\mathcal{L}(\mathbf{w})$

Make sure that you compute the gradient of the loss function $\mathcal{L}(\mathbf{w})$ (not simply the NLL!)

```
In [8]: def get_gradient(X, y, w, mini_batch_indices, lmbda):
    """
    Calculates the gradient (full or mini-batch) of the negative log likelilhood w.r.t.

    Parameters
    ----------
    X : array, shape [N, D]
        (Augmented) feature matrix.
    y : array, shape [N]
        Classification targets.
    w : array, shape [D]
        Regression coefficients (w[0] is the bias term).
    mini_batch_indices: array, shape [mini_batch_size]
        The indices of the data points to be included in the (stochastic) calculation of
        This includes the full batch gradient as well, if mini_batch_indices = np.arange
    lmbda: float
```

```
        Regularization strentgh. lmbda = 0 means having no regularization.

    Returns
    -------
    dw : array, shape [D]
        Gradient w.r.t. w.
    """
    # TODO
    y_pred = predict(X,w)
    X_req = X[mini_batch_indices,:]
    y_req = y[mini_batch_indices]
    y_pred_req = y_pred[mini_batch_indices]
    c = (y_pred_req - y_req)
    g1 = ((X_req.transpose() * c).transpose()).sum(axis=0)/len(y_req)
    g2 = lmbda*w
    return g1+g2
```

### 1.5.1 Train the logistic regression model (nothing to do here)

```
In [9]: def logistic_regression(X, y, num_steps, learning_rate, mini_batch_size, lmbda, verbose)
    """
    Performs logistic regression with (stochastic) gradient descent.

    Parameters
    ----------
    X : array, shape [N, D]
        (Augmented) feature matrix.
    y : array, shape [N]
        Classification targets.
    num_steps : int
        Number of steps of gradient descent to perform.
    learning_rate: float
        The learning rate to use when updating the parameters w.
    mini_batch_size: int
        The number of examples in each mini-batch.
        If mini_batch_size=n_train we perform full batch gradient descent.
    lmbda: float
        Regularization strentgh. lmbda = 0 means having no regularization.
    verbose : bool
        Whether to print the loss during optimization.

    Returns
    -------
    w : array, shape [D]
        Optimal regression coefficients (w[0] is the bias term).
    trace: list
        Trace of the loss function after each step of gradient descent.
    """
```

```
    trace = [] # saves the value of loss every 50 iterations to be able to plot it later
    n_train = X.shape[0] # number of training instances

    w = np.zeros(X.shape[1]) # initialize the parameters to zeros

    # run gradient descent for a given number of steps
    for step in range(num_steps):
        permuted_idx = np.random.permutation(n_train) # shuffle the data

        # go over each mini-batch and update the paramters
        # if mini_batch_size = n_train we perform full batch GD and this loop runs only
        for idx in range(0, n_train, mini_batch_size):
            # get the random indices to be included in the mini batch
            mini_batch_indices = permuted_idx[idx:idx+mini_batch_size]
            gradient = get_gradient(X, y, w, mini_batch_indices, lmbda)

            # update the parameters
            w = w - learning_rate * gradient

        # calculate and save the current loss value every 50 iterations
        if step % 50 == 0:
            loss = compute_loss(X, y, w, lmbda)
            trace.append(loss)
            # print loss to monitor the progress
            if verbose:
                print('Step {0}, loss = {1:.4f}'.format(step, loss))
    return w, trace
```

### 1.6 Task 4: Implement the function to obtain the predictions

```
In [10]: def predict(X, w):
             """
             Parameters
             ----------
             X : array, shape [N_test, D]
                 (Augmented) feature matrix.
             w : array, shape [D]
                 Regression coefficients (w[0] is the bias term).

             Returns
             -------
             y_pred : array, shape [N_test]
                 A binary array of predictions.
             """
             # TODO
             pred = sigmoid(np.dot(X,np.transpose(w)))
             pred[pred >= 0.5] = 1
```

```
        pred[pred < 0.5] = 0
        return pred
```

### 1.6.1 Full batch gradient descent

```
In [11]: # Change this to True if you want to see loss values over iterations.
         verbose = False

In [12]: n_train = X_train.shape[0]
         w_full, trace_full = logistic_regression(X_train,
                                                  y_train,
                                                  num_steps=8000,
                                                  learning_rate=1e-5,
                                                  mini_batch_size=n_train,
                                                  lmbda=0.1,
                                                  verbose=verbose)

In [13]: n_train = X_train.shape[0]
         w_minibatch, trace_minibatch = logistic_regression(X_train,
                                                            y_train,
                                                            num_steps=8000,
                                                            learning_rate=1e-5,
                                                            mini_batch_size=50,
                                                            lmbda=0.1,
                                                            verbose=verbose)
```

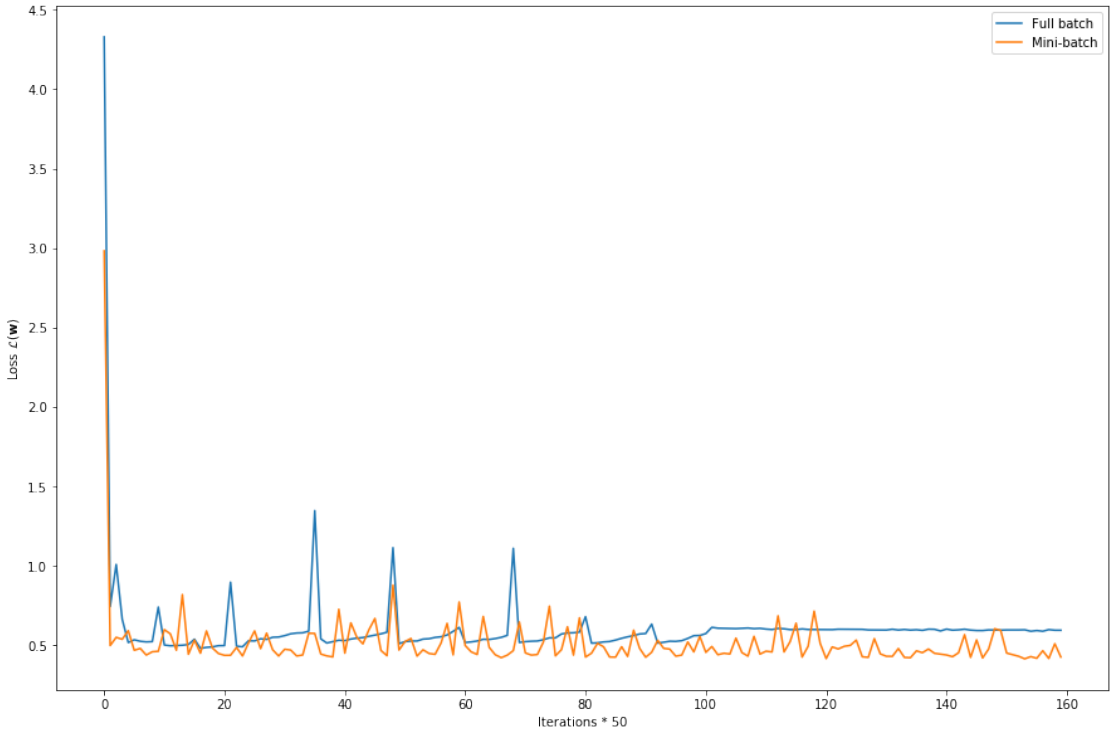
Our reference solution produces, but don't worry if yours is not exactly the same.

```
Full batch: accuracy: 0.9240, f1_score: 0.9384
Mini-batch: accuracy: 0.9415, f1_score: 0.9533
```

```
In [14]: y_pred_full = predict(X_test, w_full)
         y_pred_minibatch = predict(X_test, w_minibatch)
         print('Full batch: accuracy: {:.4f}, f1_score: {:.4f}'
               .format(accuracy_score(y_test, y_pred_full), f1_score(y_test, y_pred_full)))
         print('Mini-batch: accuracy: {:.4f}, f1_score: {:.4f}'
               .format(accuracy_score(y_test, y_pred_minibatch), f1_score(y_test, y_pred_minibat
```

```
Full batch: accuracy: 0.9126, f1_score: 0.9238
Mini-batch: accuracy: 0.9415, f1_score: 0.9524
```

```
In [257]: plt.figure(figsize=[15, 10])
          plt.plot(trace_full, label='Full batch')
          plt.plot(trace_minibatch, label='Mini-batch')
          plt.xlabel('Iterations * 50')
          plt.ylabel('Loss $\mathcal{L}(\mathbf{w})$')
          plt.legend()
          plt.show()
```


Full batch
Mini-batch
Loss $\mathcal{L}(\mathbf{w})$
Iterations * 50
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0
20
40
60
80
100
120
140
160