MACHINE LEARNING HOMEWORK SHEET 05

OPTIMIZATION

NAME: MAHALAKSHMI SABANAYAGAM

TUMID: ge73yuw

DATE: 25.11.2018

Problem 1:

i) $f(x,y) = x^2 + 2y + \cos(\sin\sqrt{\pi}) - \min\{-x^2, \log(y)\}$ and $D = (-100,100) \times (1,50)$

Looking at $f(x,y)$ as combination (sum) of functions

$x^2 \rightarrow$ convex

$2y \rightarrow$ affine

$\cos(\sin\sqrt{\pi}) \rightarrow$ constant - affine

$-\min\{x^2, \log(y)\} \rightarrow$ min is concave as $-x^2$ concave & $\log y$ is concave

$-\min$ is convex (min {concave fn} $\rightarrow$ concave)

$\therefore$ From the property sum of convex functions is convex, the given function $f(x,y)$ is convex.

ii.) $f(x) = \log(x) - x^3$ $D = (1, \infty)$

Lets find $f''(x)$ to check convexity,

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

$$f''(x) = -\frac{1}{x^2} - 6x$$

Domain of x is $(1,\infty) \Rightarrow f''(x) < 0$

$\therefore f(x)$ is not convex.

iii) $f(x) = -\min\{\log(3x+1), -x^4 - 3x^2 + 8x - 42\}$ $D = R^+$

min {concave fns} $\Rightarrow$ concave

a.) $\log(3x+1)$ - Concave $\rightarrow$ ①

b.) $f(x) = -x^4 - 3x^2 + 8x - 42$

$$f'(x) = -4x^3 - 6x + 8$$

$$f''(x) = -12x^2 - 6$$

In R^+, $x^2 \geq 0 \Rightarrow -12x^2 - 6 < 0$ $\therefore$ Concave $\rightarrow$ ②

From ①, ②

min {concave fns} is concave.

We are looking at $-\min\{\ldots\}$. This is convex.

iv.) $f(x,y) = yx^3 - yx^2 + y^2 + y + 4$ $\quad D = (-10,10) \times (-10,10)$

Computing Hessian matrix (H)

$$H = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix}$$

$$\frac{\partial^2 f}{\partial x^2} = \frac{\partial}{\partial x}(3yx^2 - 2yx) = 6xy - 2y$$

$$\frac{\partial^2 f}{\partial y^2} = \frac{\partial}{\partial y}(x^3 - x^2 + 2y + 1) = 2$$

$$\frac{\partial^2 f}{\partial x \partial y} = \frac{\partial}{\partial x}(x^3 - x^2 + 2y + 1) = 3x^2 - 2x = \frac{\partial^2 f}{\partial y \partial x}$$

$$H = \begin{bmatrix} 6xy - 2y & 3x^2 - 2x \\ 3x^2 - 2x & 2 \end{bmatrix}$$

For $f(x,y)$ to be convex, H has to be positive semidefinite

(ie) $v^T H v \geq 0$

Let v be $\begin{bmatrix} a \\ b \end{bmatrix}$

$$\begin{bmatrix} a & b \end{bmatrix} \begin{bmatrix} 6xy - 2y & 3x^2 - 2x \\ 3x^2 - 2x & 2 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix}$$

$$= \begin{bmatrix} a(6xy - 2y) + b(3x^2 - 2x) & a(3x^2 - 2x) + 2b \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix}$$

$$= a^2(6xy - 2y) + ab(3x^2 - 2x) + ab(3x^2 - 2x) + 2b^2$$

In the given domain D, for $(x = 0, y = 1)$ ~~there~~ exists

$$v^T H v < 0$$

$\therefore$ H is not positive semidefinite.

$\Rightarrow f(x,y)$ is not convex.

Problem 2: $f_1 : R^d \to R$, $f_2 : R^d \to R$ and convex functions.

$h(x) = \max\{f_1(x), f_2(x)\}$.

To prove: $h(x)$ - convex fn.

$f_1(x)$ and $f_2(x)$ are convex functions

$\Rightarrow f_1(t_1 x + (1-t_1)y) \leq t_1 f_1(x) + (1-t_1) f_1(y)$

& $f_2(t_2 x + (1-t_2)y) \leq t_2 f_2(x) + (1-t_2) f_2(y)$

For $x \in R^d$, $y \in R^d$

$tx + (1-t)y \leq t \max_i f_i(x) + (1-t) \max_i f_i(y)$

$\max(tx + (1-t)y) \leq t \max_i f_i(x) + (1-t) \max_i f_i(y)$

From the above equation, $\max\{f_1(x), f_2(x)\}$ is convex, given $f_1(x)$ and $f_2(x)$ are convex functions.

Problem 3: $f_1 : R \to R$, $f_2 : R \to R$ and convex functions.

To prove: $g(x) = f_1(f_2(x))$ convex.

Again as $f_1(x)$ and $f_2(x)$ are convex functions,

$f_1(t_1 x + (1-t_1)y) \leq t_1 f_1(x) + (1-t_1) f_1(y)$

$f_2(t_2 x + (1-t_2)y) \leq t_2 f_2(x) + (1-t_2) f_2(y)$

$g(x) = f_1(f_2(x))$

Lets look at $f_1(f_2(Tx + (1-T)y))$

As f_2 is convex,

$f_1(f_2(Tx + (1-T)y)) \leq f_1[\overbrace{T f_2(x)}^{Tx} + \overbrace{(1-T) f_2(y)}^{(1-T)y}]$

$\leq T f_1(f_2(x)) + (1-T) f_1(f_2(y))$ $\because f_1$ is also convex

$\Rightarrow f_1(f_2(x)) = g(x)$ is convex.

Problem 4: $f: R^N \to R$; f - convex.

To prove : $\nabla f(\theta^*) = 0$ then θ^* is a global minimum.

Let x be a local minimum.

Now, lets say y is also a local minimum and

$$f(y) < f(x).$$

As f is convex,

$$f(tx+(1-t)y) \leq tf(x) + (1-t)f(y) \quad \forall\, t \in R$$

$$f(tx+(1-t)y) < \underbrace{t f(x) + (1-t) f(x)}_{=} \quad \text{As } f(y)<f(x)$$

$$f(tx+(1-t)y) < f(x)$$

When $t \to 1$, $f(x) < f(x)$ which is not true.

So this implies all the local minimums should be equal and thus each local minimum will be global minimum.

Programming assignment 5: Optimization: Logistic regression

In [1]:
```python
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
```

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)

Exporting the results to PDF

Once you complete the assignments, export the entire notebook as PDF and attach it to your homework solutions. The best way of doing that is

1. Run all the cells of the notebook.
2. Download the notebook in HTML (click File > Download as > .html)
3. Convert the HTML to PDF using e.g. https://www.sejda.com/html-to-pdf or `wkhtmltopdf` for Linux (tutorial)
4. Concatenate your solutions for other tasks with the output of Step 3. On a Linux machine you can simply use `pdfunite`, there are similar tools for other platforms too. You can only upload a single PDF file to Moodle.

This way is preferred to using `nbconvert`, since `nbconvert` clips lines that exceed page width and makes your code harder to grade.

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 [2]:
```python
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)
```

Task 1: Implement the sigmoid function

In [3]:
```python
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 = np.array(t);
    t_sigmoid = 1 / (1 + np.exp(-t));
    return t_sigmoid;
```

Task 2: Implement the negative log likelihood

As defined in Eq. 33

In [4]:
```python
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
    p = sigmoid(np.matmul(X,w));
    y = np.array(y);
    NLL = -np.sum(y*np.log(p) + (1.0-y)*np.log(1.0-p));
    return NLL;
```

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

In [5]:
```python
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
```

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 [6]:
```python
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. w.

    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 the gradien
t.
        This includes the full batch gradient as well, if mini_batch_indices = np.arange(n_train).
    lmbda: float
        Regularization strentgh. lmbda = 0 means having no regularization.

    Returns
    -------
    dw : array, shape [D]
        Gradient w.r.t. w.
    """
    # TODO
    X_batch = [];
    y_batch = [];
    for i in mini_batch_indices:
        X_batch.append(X[i]);
        y_batch.append(y[i]);
    #err = np.array(y) - sigmoid(np.matmul(X,w));
    #dw = -(1/len(y))*np.matmul(np.transpose(X),err) + 2*lmbda*np.array(w);
    err = np.array(y_batch) - sigmoid(np.matmul(X_batch,w));
    dw = -(1/len(y_batch))*np.matmul(np.transpose(X_batch),err) + 2*lmbda*np.array(w);
    return dw;
```

Train the logistic regression model (nothing to do here)

In [7]:
```python
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 once
        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
```

Task 4: Implement the function to obtain the predictions

In [8]:
```python
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
    y_pred = sigmoid(np.matmul(X,w));
    for i in range(len(y_pred)):
        if y_pred[i] > 0.5:
            y_pred[i] = 1;
        else:
            y_pred[i] = 0;
    return y_pred;
```

Full batch gradient descent

In [9]:
```python
# Change this to True if you want to see loss values over iterations.
verbose = False
```

In [10]:
```python
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 [11]:
```python
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.

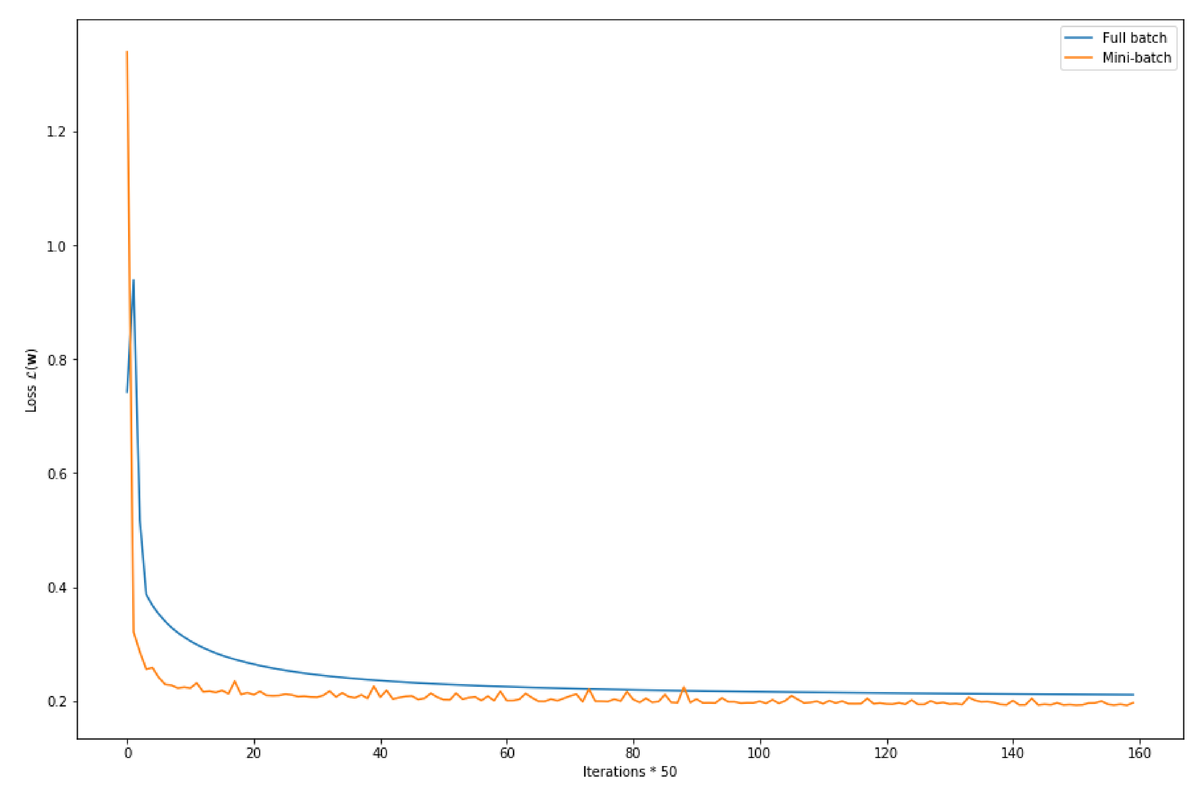
```
Full batch: accuracy: 0.9240, f1_score: 0.9384
Mini-batch: accuracy: 0.9415, f1_score: 0.9533
```

In [12]:
```python
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_minibatch)))
```

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

In [13]:
```python
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()
```



In [ ]: