Optimization

Machine Learning - Prof. Dr. Stephan Günnemann

Leonardo Freiherr von Lerchenfeld

December 3, 2017

Contents

1 Convexity

1.1 Problem 1

i) $f(x,y,z) = 3x + e^{y+z} - min\{-x^2, log(y)\}$ and $D = (-100,100) \times (1,50) \times (10,20)$

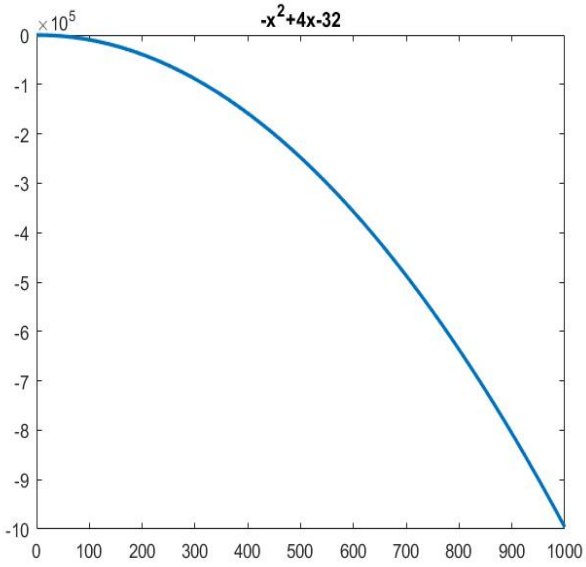
- $min\{-x^2, log(y)\} = -x^2$ for the given set
- **$\mathbf{h(x) = m(f_1(x))}$ is convex if $\mathbf{m : R \to R}$ is convex and non-decreasing** with $m(a) = e^a \to e^{y+z}$ is convex
- **$\mathbf{h(x) = f_1(x) + f_2(x)}$ is convex** with $f_1(x) = x^2$ and $f_2(x) = 3x$

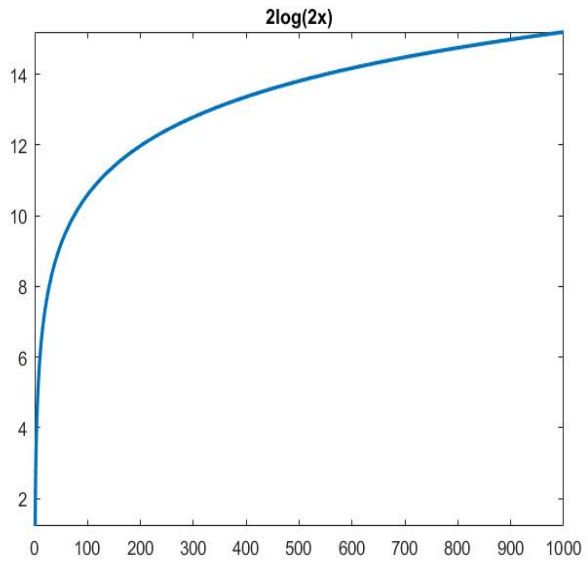
$\Rightarrow f(x,y,z)$ is convex

ii) $f(x,y) = yx^3 - 2yx^2 + y + 4$ and $D = (-10,10) \times (-10,10)$
f(x) is a convex function on a convex set X iff for all $\mathbf{p} \in \mathbf{X}$ & $\lambda \in [\mathbf{0},\mathbf{1}]$:
$\lambda \mathbf{f(p_1)} + (\mathbf{1} - \lambda)\mathbf{f(p_2)} \geq \mathbf{f}(\lambda \mathbf{p_1} + (\mathbf{1} - \lambda)\mathbf{p_2})$
with $p_1 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$, $p_2 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ and $\lambda = 0.5$
$0.5(-1)^3 - 2(-1)^2 + 1 + 4 + 0.5 * 4 = 4.5$
$0.5(0.5)^3 - 2 \times 0.5 \times (-0.5)^2 + 1 + 4 = 4.6875$
$\Rightarrow f(x,y)$ is concave, because the condition is not fulfilled

iii)$f(x) = log(x) + x^3$ and $D = (1,\infty)$
A twice differentiable function of one variable is convex on an interval iff its second-derivative is non-negative on this interval
$f''(x) = 6x - \frac{1}{x^2}$
$\Rightarrow f(x)$ is convex

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





When we look at the functions, we see that f(x) becomes most of the times the parabola. However, close to zero the log function becomes smaller than the parabola and goes to $-\infty$. Nevertheless, since the negative log is a convex function for $x > 0$
$\Rightarrow f(x)$ is convex

1.2 Problem 2

f_1 and f_2 are convex functions. Is $h(x) := f_1(x) + f_2(y)$ a convex function?
Convexity: Region above a convex function is convex

$$\lambda \mathbf{f}(\mathbf{x}) + (\mathbf{1} - \lambda)\mathbf{f}(\mathbf{y}) \in \mathbf{X} \text{ for } \mathbf{x}, \mathbf{y} \in \mathbf{X}$$

$$\begin{aligned} \mathbf{f}(\lambda \mathbf{x} + (\mathbf{1} - \lambda)\mathbf{y}) &\leq \lambda \mathbf{f}(\mathbf{x}) + (\mathbf{1} - \lambda)\mathbf{f}(\mathbf{y}) \\ f_1(\lambda x + (1-\lambda)y) + f_2(\lambda x + (1-\lambda)y) &\leq \lambda(f_1(x) + f_2(x)) + (1-\lambda)(f_1(y) + f_2(y)) \\ h(\lambda x + (1-\lambda)y) &\leq \lambda(f_1(x) + f_2(x)) + (1-\lambda)(f_1(y) + f_2(y)) \\ h(1x) &= f_1(x) + f_2(x) \end{aligned}$$

1.3 Problem 3

Two convex functions that are multiplied with each other do not result necessarily in a convex function. For example,

$$f_1(x) = x^2 \text{ and } f_2(x) = -x$$

are convex functions on the domain $D = (-10, 10)$. However,

$$g(x) = f_1(x) \times f_2(x) = -x^3$$

is not a convex function on the domain $D = (-10, 10)$.

2 Minimization of convex functions

2.1 Problem 4

First order convexity condition

$$\begin{gathered} \mathbf{f}(\mathbf{y}) \geq \mathbf{f}(\mathbf{x}) + (\mathbf{y} - \mathbf{x})^{\mathbf{T}} \nabla \mathbf{f}(\mathbf{x}) \\ \text{For } x = \theta^* \text{ and } \nabla f(\theta^*) = 0 \\ f(y) \geq f(\theta^*) \end{gathered}$$

Hence, $\nabla f(\theta^*)$ is a global minimum.

3 Gradient Descent

3.1 Problem 5

06_hw_optimization_logistic_regression_v2

December 3, 2017

1 Programming assignment 6: Optimization: Logistic regression

```
In [224]: 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 [225]: 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_
```

1.3 Task 1: Implement the sigmoid function

```
In [226]: 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 [227]: 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
              sigma = sigmoid(np.matmul(X,w))
```

```
    nll = - np.sum(y*np.log(sigma+1e-15)+(1-y)*(np.log(1-sigma+1e-15)))
    # 1e-15 to avoid log(0)
    return nll
```

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

```
In [228]: 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.
```

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 [229]: def get_gradient(X, y, w, ind, lmbda):
    """
    Calculates the gradient (full or mini-batch) of the negative log like

    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 = ind
    ind : array, shape [mini_batch_size]
        The indices of the data points to be included in the (stochastic)
```

```
        This includes the full batch gradient as well, if mini_batch_indi
    lmbda: float
        Regularization strentgh. lmbda = 0 means having no regularizatio

    Returns
    -------
    dw : array, shape [D]
        Gradient w.r.t. w.
    """
    # TODO
    """
    Thanks to Marcel Caraciolo
    http://aimotion.blogspot.de/2011/11/machine-learning-with-python-logi
    """
    n = len(ind)
    D = len(w)
    dw = np.zeros(D)
    for i in range(D):
        dw[i] = 1/n*np.matmul((sigmoid(np.matmul(X[ind],w))-y[ind]),X[inc
    return dw
```

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

```
In [230]: def logistic_regression(X, y, num_steps, learning_rate, mini_batch_size,
    """
    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 regularizatio
    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 t
    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
        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 [231]:
```
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 = np.matmul(X,w)
    y_pred[y_pred>0] = 1
    y_pred[y_pred<0] = 0
    return y_pred
```

1.6.1 Full batch gradient descent

```
In [232]: # Change this to True if you want to see loss values over iterations.
          verbose = False

In [233]: 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 [234]: 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
```

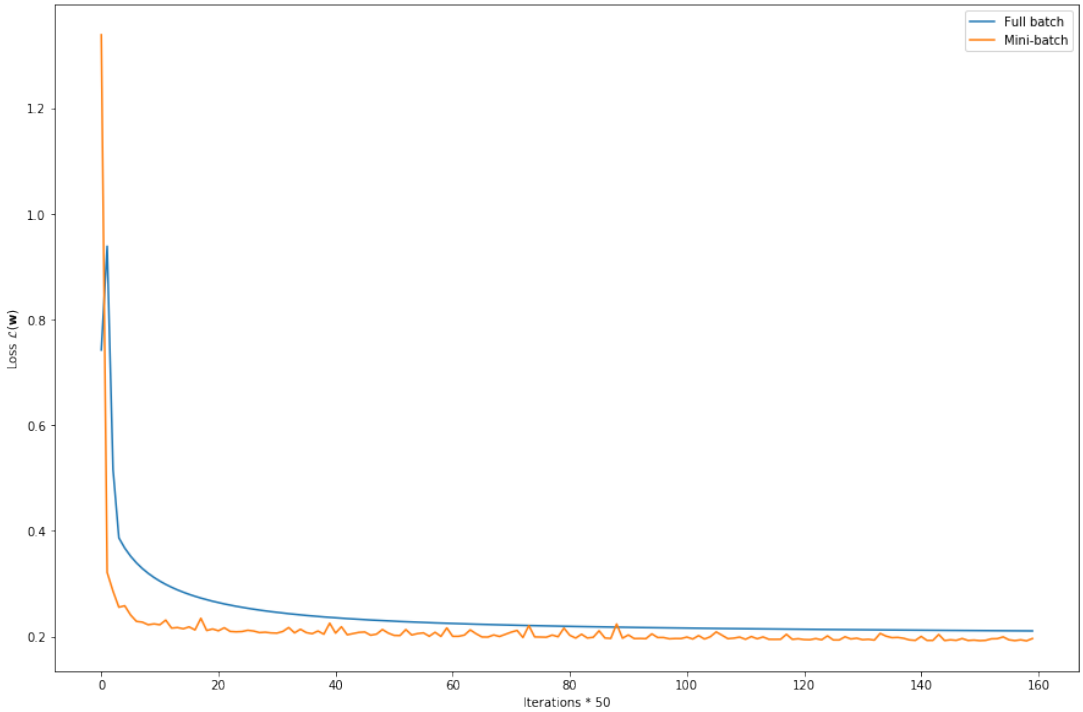
```
In [235]: y_pred_full = predict(X_test, w_full)
          y_pred_minibatch = predict(X_test, w_minibatch)

          print('Full batch: accuracy: {:.4f}, f1_score: {:.4f}'
                .format(accuracy_score(y_test, y_pred_full), f1_score(y_test, y_pre
          print('Mini-batch: accuracy: {:.4f}, f1_score: {:.4f}'
                .format(accuracy_score(y_test, y_pred_minibatch), f1_score(y_test,
```

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

```
In [236]: 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 [ ]: