MATH3204: Assignment 04

Due: 26-Oct-2020 @11:59pm

1. Logistic regression is an important binary classification technique in machine learning that builds off of the concepts of linear regression. Recall that in linear regression, there is a set of predictor variables (or features) $\boldsymbol{a}_i \in \mathbb{R}^d$, $i = 1, \ldots, n$, with corresponding outcome variables $b_i \in \mathbb{R}$, $i = 1, \ldots, n$ and our aim is to fit a linear model to the data that minimizes a quadratic loss, namely

$$\min_{\boldsymbol{x} \in \mathbb{R}^d} \frac{1}{2} \sum_{i=1}^{n} \Big(\langle \boldsymbol{a}_i, \boldsymbol{x} \rangle - b_i \Big)^2.$$

It can be easily seen that the solution to the above least-squares is the same as

$$\min_{\boldsymbol{x} \in \mathbb{R}^d} f(\boldsymbol{x}) \triangleq \sum_{i=1}^{n} \phi(\langle \boldsymbol{a}_i, \boldsymbol{x} \rangle) - b_i \langle \boldsymbol{a}_i, \boldsymbol{x} \rangle,$$

where $\phi(t) = t^2/2$. Logistic regression can be thought of as a modification of linear regression in two ways: first, the outcome variables are binary representing the two classes, i.e., $b_i \in \{0, 1\}$, $i = 1, \ldots, n$, and second, the least-squares loss is replaced with a logistic loss, i.e., $\phi(t) = \ln(1 + e^t)$, where "ln" is natural logarithm. Logistic regression is thus the problems of finding the parameter $\boldsymbol{x}$ that minimizes this new loss (note that unlike linear regression, there is no closed form solution for logistic regression and one has to resort to optimization algorithms). In machine learning, this phase is often referred to as "training". After the training phase, we now have a way of classifying a new input, that is not part of your training data, i.e., predicting the label of an out-of-sample data. This phase in machine learning is often called "testing" or "generalization". More specifically, using our trained logistic model and given any new data $\boldsymbol{a}$, we can obtain the probability of $\boldsymbol{a}$ belonging to either class 0 or 1 as

$$\mathbf{Pr}(y = 0 \mid \boldsymbol{a}) = \frac{1}{1 + e^{\langle \boldsymbol{a}, \boldsymbol{x} \rangle}}, \quad \text{and} \quad \mathbf{Pr}(y = 1 \mid \boldsymbol{a}) = \frac{1}{1 + e^{-\langle \boldsymbol{a}, \boldsymbol{x} \rangle}}.$$

(a) Derive the gradient and the Hessian matrix of $f(\boldsymbol{x})$. **[10 Marks]**

(b) Prove that $f(\boldsymbol{x})$ is convex. **[10 Marks]**

(c) Write a function that takes in three arguments: the parameter $\boldsymbol{x} \in \mathbb{R}^d$, the matrix of features $\boldsymbol{A} \in \mathbb{R}^{n \times d}$ and the vector of labels $\boldsymbol{b} \in \mathbb{R}^n$ defined as

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{a}_1^\intercal \\ \boldsymbol{a}_2^\intercal \\ \vdots \\ \boldsymbol{a}_n^\intercal, \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n, \end{bmatrix}$$

and outputs $f(\boldsymbol{x}), \nabla f(\boldsymbol{x})$ and the Hessian matrix $\nabla^2 f(\boldsymbol{x})$. Call your function `logisticFun(x,A,b)`. **Note:** In implementing your function, it is very useful to employ log-sum-exp trick for log-domain calculations. This will help avoid underflow and overflow problems, which commonly occur in functions involving logarithm of sum of exponential functions. **[10 Marks]**

(d) Test your derivations using the provided `Matlab` `Python` code. The function `derivativeTest` takes in a function handle "`fun`" and a point "`x0`", and uses first and second-order Taylor exapnsions around "`x0`" to verify if the gradient or Hessian of "`fun`" are correct; see here and here for some explanation on function handles in `Matlab` and `Python`, respectively. The argument `x0` is then fed into the function `fun`, which in turn is required to return three arguments: "`f0`, "`g0`", and "`H0`", which are respectively the function value, the gradient and Hessian of the function at `x0`. Recall from lectures that the error in approximations of $f(\boldsymbol{x}_0 + \boldsymbol{p})$ using first and second order Taylor expansions around $\boldsymbol{x}_0$ should behave, respectively, as $\mathcal{O}(\|\boldsymbol{p}\|^2)$ and $\mathcal{O}(\|\boldsymbol{p}\|^3)$, i.e., when the $\boldsymbol{p}$ is halved, the error should respectively go down by a factor 1/4 and 1/8. Of course, if the gradient or Hessian is not correctly computed, the error will behave differently and goes down at slower rates. The code uses this principle to check your gradient and the Hessian matrix. The error lines corresponding to your function are plotted in "red". These lines must be, for the most part, parallel to the lines of "Theoretical Order", which are ploted in color "blue". It is OK if some parts of the red lines at the beginning and/or at the end deviate from the blue lines (these are due to the fact that Taylor approximations are bad when $\boldsymbol{p}$ is large, and only get more accurate as $\boldsymbol{p}$ gets smaller until the round-off error takes over). However, the red lines must be parallel to the blue lines for a good portion in the middle. Use the following code to test your function

```
n=1000; d = 100;
A = randn(n,d);
I = eye(2,1);
ind = randsample(2,n,true); b = I(ind, :);
derivativeTest(@(x) logisticFun(x,A,b),ones(d,1));
```

```

```

For Matlab, use the above code snippet.

```
n,d = 1000, 50
A = rand.randn(n,d)
I = np.eye(2, 1)
ind = rand.randint(2, size = n)
b = I[ind, :]
fun = lambda x: logisticFun(x,A,b)
derivativeTest(fun,np.ones((d,1)))

```

For Python, use the above code snippet.

[5 Marks]

Explicitly forming the Hessian matrix of our logistic regression model $f(\boldsymbol{x})$ requires $\mathcal{O}(nd^2)$ in terms of computations and $\mathcal{O}(d^2)$ in storage (in addition to the unavoidable storage required for the data itself). These costs can be computationally prohibitive in large-scale problems when $d, n \gg 1$. However, many optimization algorithms only require access to Hessian in the form of Hessian-vector products and explicitly forming the Hessian matrix is not necessary.

(e) Modify your function to, instead of computing the explicit Hessian matrix, return a function handle, which given a vector $\boldsymbol{v}$, computes Hessian-vector product, $\nabla^2 f(\boldsymbol{x})\boldsymbol{v}$, without every explicitly forming Hessian. The cost of this matrix-vector product should be $\mathcal{O}(nd)$ with no additional storage requirements (**Hint:** To multiply the matrix $\boldsymbol{z}\boldsymbol{z}^\intercal$ by a vector $\boldsymbol{v}$ for some $\boldsymbol{z}, \boldsymbol{v} \in \mathbb{R}^d$, we can either form the matrix $\boldsymbol{z}\boldsymbol{z}^\intercal$ first and then apply $\boldsymbol{v}$ to it, which will take $\mathcal{O}(d^2)$ both in computation and storage or re-order the computation to do $\boldsymbol{z}\langle\boldsymbol{z}, \boldsymbol{v}\rangle$, which will only take $\mathcal{O}(d)$ in computation and storage.) **[10 Marks]**

(f) Test your Hessian-vector product function handle using the function `derivativeTest`. The code automatically checks and determines whether the matrix is explicitly formed or it is a function handle. **[5 Marks]**

2. In this question, and in the context of the logistic regression problems, you will experiment with the following optimization methods:

– Gradient Descent with Armijo line-search,

– Newton-CG Method with Armijo line-search, and

– L-BFGS with strong Wolfe conditions.

Let us recall the ℓ_2-regularized logistic regression function

$$\min_{\boldsymbol{x}\in\mathbb{R}^d} f(\boldsymbol{x}) \triangleq \sum_{i=1}^{n} \left(\ln\left(1+e^{\langle \boldsymbol{a}_i,\boldsymbol{x}\rangle}\right) - b_i \langle \boldsymbol{a}_i,\boldsymbol{x}\rangle\right) + \frac{\lambda}{2}\left\|\boldsymbol{x}\right\|^2,$$

where λ is a given regularization parameter.

(a) Show that $f(\boldsymbol{x})$ is strongly convex. **[10 Marks]**

(b) Show that the gradient of f is $L_{\boldsymbol{g}}$-Lipschitz continuous and

$$L_{\boldsymbol{g}} \leq \frac{1}{4}\left\|\boldsymbol{A}\right\|^2 + \lambda,$$

where $\|\boldsymbol{A}\|$ is the spectral norm of the matrix $\boldsymbol{A}$. **[10 Marks]**

From Question 1, you should have a function called `logisticFun(x,A,b)` that takes in three arguments: the parameter $\boldsymbol{x}\in\mathbb{R}^d$, the matrix of features $\boldsymbol{A}\in\mathbb{R}^{n\times d}$ and the vector of 0/1 labels $\boldsymbol{b}\in\mathbb{R}^n$, and outputs the function value, its gradient, and a function handle, which given a vector $\boldsymbol{v}$, computes Hessian-vector product. **Note:** If you did not manage to get your code working properly for Question 1, we have included a function called "`softMaxFun`" (for both `Matlab` and `Python`), which implements a multi-class generalization of logistic regression, i.e., when you only have 0/1 labels, it is exactly logistic regression, but it can also handle more than binary labels. You can use it to implement your optimization algorithms for this question. The documentation for `softMaxFun` is self-explanatory. Just note that in `softMaxFun`'s definition, `(x,A,b)` are, respectively, denoted by `(w,X,Y)`.

(c) Modify your function to account for the added ℓ_2-regularization term for a given λ. Check that your implementation is correct using the `derivativeTest` function from before. **[5 Marks]**

(d) Implement the above three optimization methods. Set the history size of L-BFGS to 20. Set the parameters of Armijo and curvature conditions, respectively, to 10^{-4} and 0.9. Set the termination condition to be either $\|\nabla f(\boldsymbol{x}_k)\| \leq 10^{-4}$ or $k > 1000$. For solving the linear system within Newton-CG, terminate the CG iterations if either maximum iteration count of 100 is reached or the relative residual falls below 10^{-2} (you can use native CG function in `Matlab` or `Python`). For gradient descent, set the initial trial step-size of its line-search to $\alpha = 10/L_{\boldsymbol{g}}$ using the upper-bound estimate on $L_{\boldsymbol{g}}$ from an earlier part of this question. For Newton-CG and L-BFGS, the best choice for the initial trial step size is always $\alpha = 1$, which is often accepted in line-search. The attached `MATLAB/Python` code called `lineSearchWolfeStrong` implements the strong Wolfe conditions, which you can use in your implementation of L-BFGS. The function `lineSearchWolfeStrong` takes in a few arguments:

– `objFun`: A function handle to the objective function,

– `xk`, `pk`: current iterate and the current search direction, respectively,

– `alpha0`: initial trial step-size (which for L-BFGS is always 1),

– `beta1`, `beta2`: parameters of Armijo and curvature conditions,

– `linesearchMaxItrs`: maximum allowable number of line-search (set this to be, say, 1000).

The function `lineSearchWolfeStrong` also returns two outputs:

– `alpha`: the chosen step-size, and

– `itrLS`: the total of number of iterations of Wolfe line-search, which you can ignore.

[15 Marks (5 marks for each method)]

For this question, we always initialize all optimization methods at $\boldsymbol{x}_0 = \mathbf{0}$, and set the regularization parameter $\lambda = 1$. You will test your code on a real data set, namely the spam classification data set from https://archive.ics.uci.edu/ml/datasets/spambase. You are provided with a code that loads this data set in `MATLAB` or `Python`. Download the data folder `spam` and the file `loadData`. Calling `loadData` loads the spam data set, randomly splits it into a training set and a test set, and outputs

– `A_train`: the training matrix of features/attributes,

– `b_train`: the training vector of labels,

– `A_test`: the test matrix of features/attributes, and

– `b_test`: the test vector of labels.

Note that, depending on where you store the data and your code, you might have to change the argument of `load('../data/spam/spambase.data')` inside the function `loadData` to point to the location of the data on your machine. In a nutshell, each row of the training feature matrix `A_train` consists of "attributes" for a given document. The corresponding row of `b_train` is the label of that document as being spam (1) or not (0). Same correspondence holds in the test data set.

By fitting a logistic regression to this training data, we obtain a parametric decision rule, parameterized by $\boldsymbol{x}$, that can be used to classify a new unlabeled document.

(e) Using our logistic regression model, write a function that given a document in the form of a vector of features, $\boldsymbol{a}$, and the parameter $\boldsymbol{x}$, assigns a label to $\boldsymbol{a}$. This can be done as follows: we first calculate the probability of $\boldsymbol{a}$ belonging to either class 0 or 1 as

$$\mathbf{Pr}\left(y = 0 \mid \boldsymbol{a}\right) = \frac{1}{1 + e^{\langle \boldsymbol{a}, \boldsymbol{x} \rangle}}, \quad \text{and} \quad \mathbf{Pr}\left(y = 1 \mid \boldsymbol{a}\right) = \frac{1}{1 + e^{-\langle \boldsymbol{a}, \boldsymbol{x} \rangle}}.$$

The label of $\boldsymbol{a}$ is determined as the one that has a higher probability (ties are unlikely, but you can break them arbitrarily if you encounter them). **[5 Marks]**

(f) Apply the above optimization methods to this logistic regression model using the training data `A_train` and `b_train`. Compare the performance of these three optimization methods by plotting the objective value, gradient norm, and the test accuracy (proportion of correct predictions using the test set `A_test`, `b_test`), as a function of iterations as well as "wall-clock" time. In total you should have 6 plots, namely objective value vs. iterations, gradient norm vs. iterations, test accuracy vs. iterations, objective value vs. time, gradient norm vs. time, test accuracy vs. time. Each of these plots will depict the performance of all three methods at once. **[5 Marks]**

Bonus Question 1

3. Logistic regression is a convex classification model. But there are also non-convex alternatives. As a simple example, one can consider ℓ_2-regularized non-linear least squares based on sigmoid function as

$$\min_{\boldsymbol{x}\in\mathbb{R}^d} f(\boldsymbol{x}) \triangleq \sum_{i=1}^{n} \left(b_i - \frac{1}{1+e^{-\langle \boldsymbol{a}_i, \boldsymbol{x}\rangle}} \right)^2 + \frac{\lambda}{2} \|\boldsymbol{x}\|^2 ,$$

where λ is a given regularization parameter. It is easy to see that if λ is not too large, the above problem is non-convex (of course, if λ is too large, then the problem can become convex). In this question, you will compare the performance of gradient descent, L-BFGS, and Gauss-Newton on the above model. Note that since the problem is no-longer strongly convex, vanilla Newton's method can no longer be applied (though, there are a plethora of Newton-type methods that can be applied).

(a) Write a function that takes in three arguments: the parameter $\boldsymbol{x} \in \mathbb{R}^d$, the matrix of features $\boldsymbol{A} \in \mathbb{R}^{n\times d}$ and the vector of 0/1 labels $\boldsymbol{b} \in \mathbb{R}^n$, and outputs $f(\boldsymbol{x}), \nabla f(\boldsymbol{x})$ and a function handle, which given a vector $\boldsymbol{v}$, computes matrix-vector product, $\boldsymbol{G}(\boldsymbol{x})\boldsymbol{v}$, where $\boldsymbol{G}$ is the Gauss-Newton matrix. Call your function `nlsFun(x,A,b)`. **[5 Marks]**

(b) Test your gradient using the `derivativeTest` function from before. Note that, unfortunately, there is no easy way to test the correctness of the Gauss-Newton matrix and `derivativeTest` does not apply for this test. **[5 Marks]**

(c) Implement the Gauss-Newton method with Armijo line-search and CG inner iterations. Use the same parameters as that for Newton-CG above. **[5 Marks]**

One can show that the gradient of f is $L_{\boldsymbol{g}}$-Lipschitz continuous and

$$L_{\boldsymbol{g}} \leq \frac{1}{6} \|\boldsymbol{A}\|^2 + \lambda.$$

So, for gradient descent, set the initial trial step-size of its line-search to $\alpha = 10/L_{\boldsymbol{g}}$ using the above upper-bound estimate on $L_{\boldsymbol{g}}$. For Gauss-Newton, like Newton-CG, again the best choice for the initial trial step size is $\alpha = 1$, which is often accepted in line-search. All other parameters, e.g., termination criteria, line-search parameters, etc, are as in the previous question.

(d) Run your code on the spam data set by fitting the above non-linear least squares on the training data `A_train` and `b_train`. Compare the performance of gradient descent, L-BFGS and Gauss-Newton by plotting the same set of plots as the previous question. **[5 Marks]**

100 marks in total + 20 extra marks for the bonus questions

Note:

• This assignment counts for 15% of the total mark for the course.

• You don't have to do the bonus questions, but they may help bring up your mark if you miss some points on other questions.

• Your mark for this assignment will be calculated as

$$\min\{\text{Total marks obtained for all the questions}, 100\} \times 0.15.$$

So with or without bonus, you can only get a maximum of 15%.

• Although not mandatory, if you could type up your work, e.g., `LaTex`, it would be greatly appreciated.

• Show all your work and attach your code and all the plots (if there is a programming question).

• Combine your solutions, all the additional files such as your code and numerical results, along with your coversheet, **all in one single PDF file**.

• Please submit your single PDF file on Blackboard.