Machine Learning Homework 2

Ian Sinclair
ENCE 3631-1

March 29, 2022
Dr. Zhihui Zhu

Problem 1:

In this problem you will explore implementations of three different solvers for logistic regression.

a) We will begin by implementing logistic regression using standard gradient descent for performing the maximum log likelihood step. For this problem, all you need to do is to report the value of α you used, the number of iterations required for convergence and the final negative log-likelihood (which is the last element of cost) for your choices of α.

> Note: Consider gradient descent uses the gradient of a particular convex function to gradually move towards a minimum point for some fixed step size at each iteration. Here, through a trial and error method, we take step size α, to be small enough to ensure the algorithm converges, and large enough to minimize redundancy in the number of iterations. Additionally, a tolerance, tol, around the converges of the algorithm is defined to an acceptable amount of error from the minimum value.

α	iterations	negative log likelihood
0.0025	286	21.1625

b) Using the notes from lecture 6 (page21), adaptlr-gd.py to implement Newton's method for solving this problem. The Hessian in this case is given by,

$$\frac{\Delta^2 \ell(\theta)}{\partial \theta} = -\sum_{i=1}^{n} \tilde{x}_i \tilde{x}_i^T g(\theta^T \tilde{x}_i)(1 - g(\theta^T \tilde{x}_i)).$$

where g is define as in the notes.

> Note: newtons method is similar to gradient descent, except it uses a matrix of mixed second partial derivatives, called the Hessian, to pick the best step size for each iteration.
> Therefore, in building a function for Newtons method we can re-use most of gradient descent and add a helper function to define the hessian for each iteration of θ (shown in the code below).
> Now we take the Hessian as in the above definition, and define a for-each loop which takes all of $\tilde{x}_i = x1 \in \tilde{x}$, for training data set x.

Python Code: Newtons Method function

```
def Hessian(theta,x):
    hessian = 0
    for x1 in x:
        hessian = hessian + x1*x1.T*logistic_func(theta.T,x1)*(1-logistic_func(theta.T,x1))
    return 1/hessian

def Newtons_Method(theta, x, y, tol, maxiter):
```

```
nll_vec = []
nll_vec.append(neg_log_like(theta, x, y))
nll_delta = 2.0*tol
alpha = 0
iter = 0
while (nll_delta > tol) and (iter < maxiter):
    alpha = Hessian(theta,x);
    theta = theta - (alpha * log_grad(theta, x, y))
    nll_vec.append(neg_log_like(theta, x, y))
    nll_delta = nll_vec[-2]-nll_vec[-1]
    iter += 1
print(iter);
return theta, np.array(nll_vec)
```

This results in an output.

iterations	negative log likelihood
20	21.0789

Which has approximately the same negative log likelihood, but converges (by number of iterations) roughly 10 times faster.

c) Finally try comparing the results from parts (a) and (b) when applied to a much larger dataset. Try changing n-samples to 100000. Record the final loss, running time, and the number of iterations for both algorithms when applied to this dataset.

method	alpha	iterations	negative log likelihood	time (seconds)
Gradient Descent	0.0000025	643	23830.6	31.4
Newtons Method		27	23830.5	78.113

Importantly, from the results of increasing the dataset, it can be seen that gradient descent preformed significantly more iterates the Newtons method, however converged in nearly half the time. And therefore, even though Gradient descent must preform more iterations is can still be faster when the dimensions of the training data is large enough to make computing the Hessian inefficient.

Problem 2:

In this problem we are going to prove that the perceptron learning algorithm (PLA) will eventually convergeto a linear separator for a separable data set $\{x_i, y_i\}, i = 1m..., n$, where we assume $y_i \in \{-1, 1\}$. We will analyze the algorithm assuming for simplicity that the starting point for the algorithm is given by $\theta_0 = 0$. In the version that we will analyze here, we will suppose that for iterations $i \geq 1$, the algorithm proceeds by setting,

$$\theta^j = \theta^{j-1} + y_{i_j} \tilde{x_{i_j}}$$

where (x_{i_j}, y_{i_j}) is any input/output pair in the training data that is mislabeled by the classifier defined by θ^{j-1}, and $\tilde{x_{i_j}} = \begin{bmatrix} 1 & x_{i_j}^T \end{bmatrix}^T$. The general approach of the proof will be to argue that for any θ^* which separates the data, we can show that in a sense θ^j and θ^* get more "aligned" as j grows, and that this ultimately yields an upper bound on how many iterations the algorithm can take.

a) Suppose that θ^* is normalized (i.e., $||w^*||_2 = 1$ so that,

$$\rho = \min_i \left|(\theta^*)^T \tilde{x_i}\right|$$

calculates the distance from the hyperplane defined by θ^* to the closest x_i in the training data. Argue that,

$$\min_i y_i (\theta^*)^T \tilde{x}_i = \rho > 0.$$

Proof.

Assume first there exists a hyperplane θ^* that correctly separates every training data point $\{x_i, y_i\}$. Then extending the definition of θ^* to, $\theta^* = \{x \in \mathbb{R} | {w^*}^T x + b^* = 0\}$, for some $w^*, b^* \in \mathbb{R}^d$ it is possible to define the output y_i for any point in the training data x_i, such that,

$$y_i = sign((w^*)^T x_i + b^*) = sign((\theta^*)^T \tilde{x}_i)$$

Now take ρ to be the closest point to the hyperplane defined by,

$$\rho = \min_i \left|(\theta^*)^T \tilde{x_i}\right| > 0$$

Finally, consider the cases for the inner product $((\theta^*)^T \tilde{x}_i)$, for any particular x_i in the training data.

i) $((\theta^*)^T \tilde{x}_i) < 0$,

Consider if $((\theta^*)^T \tilde{x}_i) < 0$, then by our choice of y_i it follows that,

$$y_i = sign(((\theta^*)^T \tilde{x}_i)) = -1.$$

Then, consider the expression,

$$y_i((\theta^*)^T \tilde{x}_i) = -1((\theta^*)^T \tilde{x}_i) = |((\theta^*)^T \tilde{x}_i)| > 0.$$

ii) $((\theta^*)^T \tilde{x}_i) > 0$,

Again by our choice of definition for y_i, it follows that,

$$y_i = sign(((\theta^*)^T \tilde{x}_i)) = 1$$

and so,

$$y_i((\theta^*)^T \tilde{x}_i) = 1((\theta^*)^T \tilde{x}_i) = |((\theta^*)^T \tilde{x}_i)| > 0.$$

iii) Lastly, consider the trivial case $((\theta^*)^T \tilde{x}_i) = 0$, then it follows that $x_i \in \theta^*$, and because θ^* correctly divides the training data, this is impossible.

Furthermore, this implies that regardless of the sign of $((\theta^*)^T\tilde{x}_i)$, it follows that,

$$y_i((\theta^*)^T\tilde{x}_i) = |((\theta^*)^T\tilde{x}_i)|$$

And therefore it follows that,

$$0 < \min_i(y_i(\theta^*)^T\tilde{x}_i) = \min_i |((\theta^*)^T\tilde{x}_i)| = \rho.$$

Therefore, $\min_i(y_i(\theta^*)^T\tilde{x}_i) = \rho$ as requested.

□

b) Show that $(\theta^*)^T\theta^j \geq (\theta^*)^T\theta^{j-1} + \rho$, and calculate that $(\theta^*)^T\theta^j \geq j\rho$.

Proof.

Assume that we count iterations j as the number of actual updates to θ, then for each iteration it follows that,

$$\theta^j = \theta^{j-1} + y_{i_j}\tilde{x}_{i_j}$$

for (x_{i_j}, y_{i_j}) are misclassified training data points by the hyperplane θ^{j-1}.
Now, proceeding by induction, allow for the hypothesis;

$$\textit{Hypothesis:} \qquad (\theta^*)^T\theta^j \geq (\theta^*)^T\theta^{j-1} + \rho$$

Then proceeding by induction it follows that,

i) *Basis:* $j = 1$,

Consider the case for $j = 1$, then the hypothesis is supported if,

$$(\theta^*)^T\theta^1 \geq (\theta^*)^T\theta^0 + \rho$$

Now $\theta^0 = \theta_0 = 0$ by definition,
additionally, by our choice of iteration parameters, there exists (x_{i_1}, y_{i_1}) in the training data set; such that, $\theta^1 = \theta^0 + y_{i_1}\tilde{x}_{i_1}$. Therefore,

$$(\theta^*)^T\theta^j = (\theta^*)^T(\theta^0 + y_{i_1}\tilde{x}_{i_1}) \geq \rho, \longrightarrow y_{i_1}(\theta^*)^T\tilde{x}_{i_1} \geq \rho.$$

Next consider the definition for ρ from the previous problem, $\rho = \min_i(y_i(\theta^*)^T\tilde{x}_i)$, it follows that,

$$y_{i_1}(\theta^*)^T x_{i_1} \geq \min_i(y_i(\theta^*)^T\tilde{x}_i)$$

Now note, that (x_{i_j}, y_{i_j}) are misclassified by θ^{j-1}; however, by our choice of (θ^*), which correctly classifies every data set, it follows that (x_{i_j}, y_{i_j}) is correctly classified by (θ^*).
But then, by the definition of ρ it must be true that,

$$y_{i_1}(\theta^*)^T x_{i_1} \geq \min_i(y_i(\theta^*)^T\tilde{x}_i)$$

Thus, the hypothesis is verified for the basis step.

ii) Inductive step: $j + 1$.

Consider the arbitrary case for $j + 1$ iterations, then to test our hypothesis, assume,

$$(\theta^*)^T\theta^{j+1} \geq (\theta^*)^T\theta^j + \rho$$

Then it follows that there exists $(x_{i_{j+1}}, y_{i_{j+1}})$ that are misclassified by θ^j such that, $\theta^{j+1} = \theta^j + y_{i_{j+1}}x_{i_{j+1}}$. Then,

$$(\theta^*)^T\theta^{j+1} = (\theta^*)^T(\theta^j + y_{i_{j+1}}x_{i_{j+1}}) = (\theta^*)^T\theta^j + y_{i_{j+1}}(\theta^*)^T x_{i_{j+1}} \geq (\theta^*)^T\theta^j + \rho.$$
$$\longrightarrow y_{i_j}(\theta^*)^T x_{i_j} \geq \rho.$$

And by a symmetric argument to the basis case, $(x_{i_{j+1}}, y_{i_{j+1}})$ is correctly classified by θ^*, and so for $\rho = \min y_i(\theta^*)^T x_i$ then by definition,

$$y_{i_{j+1}}(\theta^*)^T x_{i_{j+1}} \geq \min y_i(\theta^*)^T x_i = \rho.$$

Therefore, the hypothesis is verified for the inductive step.

Finally, this proves $(\theta^*)^T\theta^j \geq (\theta^*)^T\theta^{j-1} + \rho$ by induction.

Now it is possible to show that, $(\theta^*)^T\theta^j \geq j\rho$, by again using induction.
And so assume $(\theta^*)^T\theta^j \geq j\rho$, then it follows that,

i) Basis: $j = 1$,

For the basis step consider,

$$(\theta^*)^T\theta^1 = (\theta^*)^T(\theta^0 + y_{i_1}x_{i_1}) = y_{i_1}(\theta^*)^T x_{i_1}$$

for some $(y_{i_1}x_{i_1})$ misclassified by θ^0.
Now by our choice of θ^*, it must be true that $(y_{i_1}x_{i_1})$ is correctly classified by θ^*.
Therefore, by definition of ρ, it follows that,

$$j\rho = \rho = \min_i y_i(\theta^*)^T x_i$$

And so trivially,

$$y_{i_1}(\theta^*)^T x_{i_1} \geq \min_i y_i(\theta^*)^T x_i$$

So,

$$(\theta^*)^T\theta^1 \geq \rho.$$

Therefore, the hypothesis is satisfied for the basis step.

ii) Inductive step: $j+1$ Consider the case iteration $j+1$, then there exists $(x_{i_{j+1}}, y_{i_{j+1}})$ in the training data that is misclassified by θ^j, and note from the hypothesis,

$$(\theta^*)^T\theta^j \geq j\rho.$$

Then it follows that,

$$(\theta^*)^T\theta^{j+1} = (\theta^*)^T(\theta^j + y_{i_{j+1}}x_{i_{j+1}}) = \underbrace{(\theta^*)^T\theta^j}_{\geq j\rho} + \underbrace{y_{i_{j+1}}(\theta^*)^T x_{i_{j+1}}}_{\geq \rho} \geq (j+1)\rho.$$

Therefore, $(\theta^*)^T\theta^j \geq j\rho$ by induction.

□

c) Show that $||\theta^j||_2^2 \leq ||\theta^{j-1}||_2^2 + ||\tilde{x}_{i_j}||_2^2$. [hint: use the fact that $\tilde{x}_{i_j}$ was misclassified by θ^{j-1}, which implies $y_i \neq sign((\theta^{j-1})^T\tilde{x}_{i_j})$. Also use the formula that for any $a, b \in \mathbb{R}^d$, we have $||a+b||_2^2 = ||a||_2^2 + ||b_2^2|| + 2a^Tb$.]

Proof.

Consider if there exists (x_{i_j}, y_{i_j}) in the training data that is misclassified by θ^{j-1}, then it follows that, $\theta^j = (\theta^{j-1} + y_{i_j}x_{i_j})$, and so,

$$||\theta^j||_2^2 = ||\theta^{j-1} + y_{i_j}x_{i_j}||_2^2 = ||\theta^{j-1}||_2^2 + ||y_{i_j}x_{i_j}||_2^2 + y_{i_j}(\theta^{j-1})^T x_{i_j}$$

Now note because $y_{i_j} \in \{-1, 1\}$ it follows that $||y_{i_j}x_{i_j}||_2^2 = ||x_{i_j}||_2^2$.
Additionally, because $\tilde{x}_{i_j}$ was misclassified by θ^{j-1}, it follows that $y_{i_j} \neq sign((\theta^{j-1})^T\tilde{x}_{i_j})$.
Therefore,

$$y_{i_j}(\theta^{j-1})^T\tilde{x}_{i_j} \leq 0.$$

Finally, this implies,

$$||\theta^{j-1}||_2^2 + \underbrace{||y_{i_j}x_{i_j}||_2^2}_{\leq ||\tilde{x}_{i_j}||_2^2} + \underbrace{y_{i_j}(\theta^{j-1})^T x_{i_j}}_{\leq 0} \leq ||\theta^{j-1}||_2^2 + ||\tilde{x}_{i_j}||_2^2.$$

And so it must be true that $||\theta^j||_2^2 \leq ||\theta^{j-1}||_2^2 + ||\tilde{x}_{i_j}||_2^2$.

□

d) Show by induction that $||\theta^j||_2^2 \leq j(1+R^2)$, where $R = \max_i ||x_i||_2$.

Proof.

Consider the following argument by induction under the hypothesis that, $||\theta^j||_2^2 \leq j(1+R^2)$.

i) basis $j = 1$,

Consider for $j = 1$, it follows that there exists (x_{i_1}, y_{i_1}) in the training data that is misclassified by θ^0, then

$$||\theta^1||_2^2 = ||\theta^0 + y_{i_1}x_{i_1}||_2^2 = ||y_{i_1}x_{i_1}||_2^2.$$

Now because $y_{i_j} \in \{-1, 1\}$ it follows that $||y_{i_j}x_{i_j}||_2^2 = ||x_{i_j}||_2^2$.
And so by definition of $R = \max_i ||x_i||_2$, it follows that,

$$||x_{i_j}||_2^2 \leq R^2 \leq 1 + R^2 \longrightarrow ||\theta^1||_2^2 \leq (1+R^2)$$

And so the hypothesis is verified for the basis step.

ii) Inductive step $j+1$.

Consider the case $j+1$, and note again our hypothesis, $||\theta^j||_2^2 \leq j(1+R^2)$, then it follows that there exists $(x_{i_{j+1}}, y_{i_{j+1}})$, that is misclassified by θ^j, and so,

$$||\theta^{j+1}||_2^2 = ||\theta^j + y_{i_{j+1}}x_{i_{j+1}}||_2^2 = ||\theta^j||_2^2 + ||y_{i_{j+1}}x_{i_{j+1}}||_2^2 + y_{i_{j+1}}(\theta^j)^T x_{i_{j+1}}.$$

Now because $(x_{i_{j+1}}, y_{i_{j+1}})$ are misclassified by θ^j, it follows that $y_{i_{j+1}}(\theta^j)x_{i_{j+1}} \leq 0$.
And, trivially, $||y_{i_{j+1}}x_{i_{j+1}}||_2^2 \leq ||x_{i_{j+1}}||_2^2 \leq R^2 \leq 1 + R^2$.
Lastly, by the hypothesis, $||\theta^j||_2^2 \leq j(1+R^2)$.
it follows that,

$$\underbrace{||\theta^j||_2^2}_{\leq j(1+R^2)} + \underbrace{||y_{i_{j+1}}x_{i_{j+1}}||_2^2}_{\leq (1+R^2)} + \underbrace{y_{i_{j+1}}(\theta^j)^T x_{i_{j+1}}}_{\leq 0} \leq j(1+R^2) + (1+R^2) + 0 = (j+1)(1+R^2).$$

Implies,

$$||\theta^{j+1}||_2^2 \leq (j+1)(1+R^2).$$

And the hypothesis is verified for the inductive step.

Therefore, $||\theta^j||_2^2 \leq j(1+R^2)$ by induction.

□

e) Show that (b) and (d) imply that,

$$j \leq \frac{(1+R^2)||\theta^*||_2^2}{\rho^2}$$

[Hint: Use the Cauchy-Schwartz inequality: for any $a, b \in \mathbb{R}^d$, we have $|a^T b| \leq ||a||_2||b||_2$.]

Proof.

Consider by (b), and the fact that $j\rho \geq 0$, it must be true that,

$$(\theta^*)^T(\theta^j) \geq j\rho \longrightarrow |(\theta^*)^T(\theta^j)| \geq j\rho$$

Now by Cauchy-Schuwartz inequality,

$$|(\theta^*)^T(\theta^j)| \geq j\rho \longrightarrow ||\theta^*||_2||\theta^j||_2 \geq j\rho \longrightarrow ||\theta^*||_2^2||\theta^j||_2^2 \geq j^2\rho^2.$$
$$\longrightarrow j^2 \leq \frac{||\theta^*||_2^2||\theta^j||_2^2}{\rho^2}$$

Now by (d), it follows that, $||\theta||_2^2 \leq j(1+R^2)$, so

$$j^2 \leq \frac{||\theta^*||_2^2||\theta^j||_2^2}{\rho^2} \leq \frac{||\theta^*||_2^2(j)(1+R^2)}{\rho^2}$$

Implies,

$$j \leq \frac{||\theta^*||_2^2(1+R^2)}{\rho^2}$$

Therefore, j must have an upper bound assuming θ^* exists.

And by extension, PLA must converge in a finite number of iterations. □

Appendix:

Full Python Code:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn import datasets
from math import exp
import time


def Hessian(theta,x):
    hessian = 0
    for x1 in x:
        hessian = hessian + x1*x1.T*logistic_func(theta.T,x1)*(1-logistic_func(theta.T,x1))
    return 1/hessian

def Newtons_Method(theta, x, y, tol, maxiter):
    nll_vec = []
    nll_vec.append(neg_log_like(theta, x, y))
    nll_delta = 2.0*tol
    alpha = 0
    iter = 0
    while (nll_delta > tol) and (iter < maxiter):
        alpha = Hessian(theta,x);
        theta = theta - (alpha * log_grad(theta, x, y))
        nll_vec.append(neg_log_like(theta, x, y))
        nll_delta = nll_vec[-2]-nll_vec[-1]
        iter += 1
    print(iter);
    return theta, np.array(nll_vec)


# the logistic function
def logistic_func(theta, x):
    t = x.dot(theta)
    g = np.zeros(t.shape)
    # split into positive and negative to improve stability
    g[t>=0.0] = 1.0 / (1.0 + np.exp(-t[t>=0.0]))
    g[t<0.0] = np.exp(t[t<0.0]) / (np.exp(t[t<0.0])+1.0)
    return g

# function to compute log-likelihood
def neg_log_like(theta, x, y):
    g = logistic_func(theta,x)
    return -sum(np.log(g[y>0.5])) - sum(np.log(1-g[y<0.5]))

# function to compute the gradient of the negative log-likelihood
def log_grad(theta, x, y):
    g = logistic_func(theta,x)
    return -x.T.dot(y-g)

# implementation of gradient descent for logistic regression
def grad_desc(theta, x, y, alpha, tol, maxiter):
    nll_vec = []
    nll_vec.append(neg_log_like(theta, x, y))
    nll_delta = 2.0*tol
    iter = 0
    while (nll_delta > 0) and (iter < maxiter):
        theta = theta - alpha * log_grad(theta, x, y)
        nll_vec.append(neg_log_like(theta, x, y))
        nll_delta = nll_vec[-2]-nll_vec[-1]
        iter += 1
    print(iter);
    return theta, np.array(nll_vec)

# function to compute output of LR classifier
def lr_predict(theta,x):
    # form Xtilde for prediction
    shape = x.shape
    Xtilde = np.zeros((shape[0],shape[1]+1))
    Xtilde[:,0] = np.ones(shape[0])
    Xtilde[:,1:] = x
    return logistic_func(theta,Xtilde)

## Generate dataset
np.random.seed(2020) # Set random seed so results are repeatable
x,y = datasets.make_blobs(n_samples=100000,n_features=2,centers=2,cluster_std=6.0)

## build classifier
# form Xtilde
shape = x.shape
```

```
xtilde = np.zeros((shape[0],shape[1]+1))
xtilde[:,0] = np.ones(shape[0])
xtilde[:,1:] = x

# Initialize theta to zero
theta = np.zeros(shape[1]+1)

# Run gradient descent
alpha = 0.0025
tol = 1e-3
maxiter = 10000
#theta, cost = grad_desc(theta, xtilde, y, alpha, tol, maxiter)

start = time.time()
theta, cost = Newtons_Method(theta, xtilde, y, tol, maxiter)
end = time.time()
print(end-start)
## Plot the decision boundary.
# Begin by creating the mesh [x_min, x_max]x[y_min, y_max].
#h = .02  # step size in the mesh
#x_delta = (x[:, 0].max() - x[:, 0].min())*0.05 # add 5% white space to border
#y_delta = (x[:, 1].max() - x[:, 1].min())*0.05
#x_min, x_max = x[:, 0].min() - x_delta, x[:, 0].max() + x_delta
#y_min, y_max = x[:, 1].min() - y_delta, x[:, 1].max() + y_delta
#xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
#Z = lr_predict(theta,np.c_[xx.ravel(), yy.ravel()])

# Create color maps
#cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA'])
#cmap_bold = ListedColormap(['#FF0000', '#00FF00'])

# Put the result into a color plot
#Z = Z.reshape(xx.shape)
#plt.figure()
#plt.pcolormesh(xx, yy, Z, cmap=cmap_light)

## Plot the training points
#plt.scatter(x[:, 0], x[:, 1], c=y, cmap=cmap_bold)

## Show the plot
#plt.xlim(xx.min(), xx.max())
#plt.ylim(yy.min(), yy.max())
#plt.title("Logistic regression classifier")
#plt.show()
```