CS 189 Introduction to Machine Learning
Summer 2019 Marc Khoury & Brijen Thananjeyan HW4

Due: Monday, July 22 at 11:59 pm

Deliverables:

1. Submit your predictions for the test sets to Kaggle as early as possible. Include your Kaggle scores in your write-up (see below). The Kaggle competition for this assignment can be found at:

   - `https://www.kaggle.com/c/summer-cs189-hw4-wine`

2. Submit a PDF of your homework, **with an appendix listing all your code**, to the Gradescope assignment entitled "Homework 4 Write-Up". In addition, please include, as your solutions to each coding problem, the specific subset of code relevant to that part of the problem. You may typeset your homework in LaTeX or Word (submit PDF format, **not** .doc/.docx format) or submit neatly handwritten and scanned solutions. **Please start each question on a new page.** If there are graphs, include those graphs in the correct sections. **Do not** put them in an appendix. We need each solution to be self-contained on pages of its own.

   - In your write-up, please state with whom you worked on the homework.
   - In your write-up, please copy the following statement and sign your signature next to it. (Mac Preview and FoxIt PDF Reader, among others, have tools to let you sign a PDF file.) We want to make it *extra* clear so that no one inadverdently cheats.

     "I certify that all solutions are entirely in my own words and that I have not looked at another student's solutions. I have given credit to all external sources I consulted."

3. Submit all the code needed to reproduce your results to the Gradescope assignment entitled "Homework 4 Code". Yes, you must submit your code twice: in your PDF write-up following the directions as described above so the readers can easily read it, and once in compilable/interpretable form so the readers can easily run it. Do **NOT** include any data files we provided. Please include a short file named README listing your name, student ID, and instructions on how to reproduce your results. Please take care that your code doesn't take up inordinate amounts of time or memory. If your code cannot be executed, your solution cannot be verified.

1 Logistic Regression with Newton's Method

Consider sample points $X_1, X_2, \ldots, X_n \in \mathbb{R}^d$ and associated values $y_1, y_2, \ldots, y_n \in \{0, 1\}$, an $n \times d$ design matrix $X = [X_1 \quad \ldots \quad X_n]^\top$ and an n-vector $y = [y_1 \quad \ldots \quad y_n]^\top$.

If we add ℓ_2-regularization to logistic regression, the cost function is

$$J(w) = \lambda\,|w|^2 - \sum_{i=1}^{n} \Big(y_i \ln s_i + (1 - y_i) \ln(1 - s_i) \Big)$$

where $s_i = s(X_i \cdot w)$, $s(\gamma) = 1/(1 + e^{-\gamma})$, and $\lambda > 0$ is the regularization parameter. As in lecture, the vector $s = [s_1 \quad \ldots \quad s_n]^\top$ is a useful shorthand.

In this problem, you will use Newton's method to minimize this cost function on the four-point, two-dimensional training set

$$X = \begin{bmatrix} 0 & 3 \\ 1 & 3 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}, \qquad y = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}.$$

You may want to draw these points on paper to see what they look like. The y-vector implies that the first two sample points are in class 1, and the last two are in class 0.

These sample points cannot be separated by a linear decision boundary that passes *through the origin*. As described in lecture, append a 1 to each X_i vector and use a weight vector $w \in \mathbb{R}^3$ whose last component is the bias term (the term we call α in lecture).

1. Derive the gradient of the cost function $J(w)$. Your final answer should be a simple matrix-vector expression. While you may derive the matrix-vector form by first deriving the components of the gradient vector, do NOT write your answer in terms of these individual components.

   Solution: By chain rule, $\nabla_w s_i = s_i(1 - s_i)x_i$, and $\nabla_w(\lambda|w|^2) = 2\lambda w$. Using the fact that $\frac{d}{dx} \log x = 1/x$, we can then write the gradient as

   $$\nabla_w J = 2\lambda w - \sum_{i=1}^{n} \left(y_i s_i(1 - s_i)x_i/s_i + (1 - y_i)s_i(1 - s_i)x_i/(1 - s_i) \right)$$

   Consider a single term of the sum. When $y_i = 1$, this is $(1 - s_i)x_i$. When $y_i = 0$, this is $-s_i x_i$. So we can write the whole thing as $(y_i - s_i)x_i$. In matrix-vector form, this is

   $$\nabla_w J = 2\lambda w - X^\top (Y - s)$$

2. Derive the Hessian of $J(w)$. Again, your answer should be a simple matrix-vector expression.

Solution: The gradient of this first term is $2\lambda I$. The second term resolves to $X^\top \nabla_w s$. We note that the i-th entry s_i of s has gradient $\nabla_w s_i = s_i(1-s_i)x_i$, so the i-th row of $\nabla_w s$ is $s_i(1-s_i)x_i^\top$. Hence, we can write this term as

$$\nabla_w s = \Omega X$$

Where $\Omega = \text{diag}(s)$. Putting this together, we get

$$\nabla_w^2 J = 2\lambda I + X^\top \Omega X$$

3. State the update equation for one iteration of Newton's method for this problem. **Solution:** Generally, the update equation for Newton's method is

$$w_{t+1} = w_t - (\nabla_w^2 J)^{-1}(\nabla_w J)$$

Plugging in our previous results, we are left with

$$w_{t+1} = w_t - (2\lambda I + X^\top \Omega X)^{-1}(2\lambda w_t - X^\top(Y-s))$$

4. We are given a regularization parameter of $\lambda = 0.07$ and a starting point of $w^{(0)} = \begin{bmatrix} -2 & 1 & 0 \end{bmatrix}^\top$. For the following four parts, you need only state the final solution. Thus you may derive the solution by hand or implement Newton's algorithm and report the final result. If you do the latter, you do not need to submit code for this part.

Solution:

```
import numpy as np

# Helper func
def sigmoid(x):
    sigm = 1. / (1. + np.exp(-x))
    return sigm

def calc_next_si(X, wi):
  return sigmoid(X.dot(wi))

def calc_next_wi(X, wi, si, lambda_):
  Omega = np.diag(np.multiply(si, 1 - si))
  XtOX_term = X.T.dot(Omega).dot(X)
  left_side = np.linalg.inv((2 * lambda_ * np.eye(XtOX_term.shape[0]) + XtOX_term))
  right_side = 2 * lambda_ * wi - X.T.dot(y - si)
  w_next = wi - left_side.dot(right_side)
  return w_next

# Define data
X = np.array(
    [[0, 3, 1],
     [1, 3, 1],
     [0, 1, 1],
     [1, 1, 1]]
  )
y = np.array([1, 1, 0, 0])
w_0 = np.array([-2, 1, 0])
lambda_ = 0.07
```

```
# Part A.
s_0 = calc_next_si(X, w_0)
print("Part A. s_0", s_0)

# Part B
w_1 = calc_next_wi(X, w_0, s_0, lambda_)
print("Part B. w_1", w_1)

# Part C
s_1 = calc_next_si(X, w_1)
print("Part C. s_1", s_1)

# Part D
w_2 = calc_next_wi(X, w_1, s_1, lambda_)
print("Part D. w_2", w_2)
```

(a) State the value of $s^{(0)}$ (the value of s before any iterations).

Solution:

$$s^{(0)} = \begin{bmatrix} 0.9526 \\ 0.7311 \\ 0.7311 \\ 0.2689 \end{bmatrix}$$

(b) State the value of $w^{(1)}$ (the value of w after one iteration).

Solution:

$$w^{(1)} = \begin{bmatrix} -0.3868 \\ 1.4043 \\ -2.2842 \end{bmatrix}$$

(c) State the value of $s^{(1)}$.

Solution:

$$s^{(1)} = \begin{bmatrix} 0.8731 \\ 0.8238 \\ 0.2932 \\ 0.2198 \end{bmatrix}$$

(d) State the value of $w^{(2)}$ (the value of w after two iterations).

Solution:

$$w^{(2)} = \begin{bmatrix} -0.5122 \\ 1.4527 \\ -2.1627 \end{bmatrix}$$

2 ℓ_1- and ℓ_2-Regularization

Consider sample points $X_1, X_2, \ldots, X_n \in \mathbb{R}^d$ and associated values $y_1, y_2, \ldots, y_n \in \mathbb{R}$, an $n \times d$ design matrix $X = [X_1 \quad \ldots \quad X_n]^\top$ and an n-vector $y = [y_1 \quad \ldots \quad y_n]^\top$. For the sake of simplicity, assume that the sample data has been centered and whitened so that each feature has mean 0 and variance 1 and the features are uncorrelated; i.e., $X^\top X = nI$. For this question, we will not use a fictitious dimension nor a bias term; our linear regression function will output zero for $x = 0$.

Consider linear least-squares regression with regularization in the ℓ_1-norm, also known as Lasso. The Lasso cost function is

$$J(w) = |Xw - y|^2 + \lambda \, \|w\|_1$$

where $w \in \mathbb{R}^d$ and $\lambda > 0$ is the regularization parameter. Let $w^* = \arg\min_{w \in \mathbb{R}^d} J(w)$ denote the weights that minimize the cost function.

In the following steps, we will show that whitened training data decouples the features, so that w_i^* is determined by the i^{th} feature alone (i.e., column i of the design matrix X), regardless of the other features. This is true for both Lasso and ridge regression.

1. We use the notation X_{*i} to denote column i of the design matrix X, which represents the i^{th} feature. Write $J(w)$ in the following form for appropriate functions g and f.

$$J(w) = g(y) + \sum_{i=1}^{d} f(X_{*i}, w_i, y, \lambda)$$

Solution: We expand the objective, and simplify it using the fact that $X^\top X = nI$ since the data is whitened.

$$\begin{aligned} J(w) &= w^\top X^\top X w - 2y^\top X w + \lambda \|w\|_1 + \|y\|^2 \\ &= n|w|^2 - 2y^\top X w + \lambda \|w\|_1 + |y|^2 \end{aligned}$$

We can write each term in sum form: $n|w|^2 = \sum_{i=1}^d n w_i^2$. $\lambda \|w\|_1 = \sum_{i=1}^d \lambda |w_i|$. And $-2y^\top X w = \sum_{i=1}^d -2y^\top X_{*i} w_i$. So the appropriate functions are $g(y) = |y|^2$, and

$$f(X_{*i}, w_i, y, \lambda) = n w_i^2 - 2y^\top X_{*i} w_i + \lambda |w_i|$$

2. If $w_i^* > 0$, what is the value of w_i^*?

Solution: We want to minimize

$$-2y^\top X_{*i} w_i + n w_i^2 + \lambda w_i.$$

Setting the derivative to zero yields

$$w_i^* = \frac{1}{n}(y^\top X_{*i} - \lambda/2).$$

3. If $w_i^* < 0$, what is the value of w_i^*?

   Solution: We want to minimize

   $$-2y^\top X_{*i}w_i + nw_i^2 - \lambda w_i.$$

   Setting the derivative to zero yields

   $$w_i^* = \frac{1}{n}(y^\top X_{*i} + \lambda/2).$$

4. Considering parts 2 and 3, what is the condition for w_i^* to be zero?

   Solution: w_i^* cannot be positive if $y^\top X_{*i} - \lambda/2 \le 0$, and w_i^* cannot be negative if $y^\top X_{*i} + \lambda/2 \ge 0$. So w_i^* is zero if both are true, i.e., $-\lambda \le 2y^\top X_{*i} \le \lambda$.

5. Now consider ridge regression, which uses the ℓ_2 regularization term $\lambda\,|w|^2$. How does this change the function $f(\cdot)$ from part 1? What is the new condition in which $w_i^* = 0$? How does it differ from the condition you obtained in part 4?

   Solution: The portion $f(\cdot)$ of the cost function involving w_i is

   $$-2y^\top X_{*i}w_i + nw_i^2 + \lambda w_i^2.$$

   Setting the derivative to zero yields

   $$w_i^* = \frac{y^\top X_{*i}}{n+\lambda}.$$

   Hence w_i^* is zero if $y^\top X_{*i} = 0$. In contrast, $w_i^* = 0$ when $|2y^\top X_{*i}| < \lambda$ in Lasso regression. This is why ℓ_1-norm regularization encourages sparsity.

3 Regression and Dual Solutions

1. For a vector w, derive $\nabla\,|w|^4$. Then derive $\nabla_w\,|Xw - y|^4$.

   Solution:

   $$\begin{aligned}
   \nabla\,|w|^4 &= 2|w|^2\,\nabla\,|w|^2 \\
   &= 4|w|^2 w. \\
   \nabla\,|Xw-y|^4 &= 2|Xw-y|^2\,\nabla\,|Xw-y|^2 \\
   &= 4|Xw-y|^2(X^\top Xw - X^\top y).
   \end{aligned}$$

2. Consider sample points $X_1, X_2, \ldots, X_n \in \mathbb{R}^d$ and associated values $y_1, y_2, \ldots, y_n \in \mathbb{R}$, an $n \times d$ design matrix $X = [X_1 \quad \ldots \quad X_n]^\top$ and an n-vector $y = [y_1 \quad \ldots \quad y_n]^\top$, and the regularized regression problem

$$w^* = \arg\min_{w \in \mathbb{R}^d} \; |Xw - y|^4 + \lambda\,|w|^2,$$

where $\lambda > 0$. This problem is similar to ridge regression, but we take the fourth power of the error instead of the squared error. (It is not possible to write the optimal solution w^* as the solution of a system of linear equations, but it can be found by gradient descent or Newton's method.)

Show that the optimum w^* is unique. By setting the gradient of the objective function to zero, show that w^* can be written as a linear combination $w^* = \sum_{i=1}^n a_i X_i$ for some scalars $a_1, \ldots, a_n$. Write the vector a of dual coefficients in terms of X, y, λ, and the optimal solution w^*.

Solution: Let $J(w)$ be the cost function; then

$$\nabla J(w) = 4|Xw - y|^2(X^\top Xw - X^\top y) + 2\lambda w.$$

Setting $\nabla J(w) = 0$ gives

$$w^* = \frac{2}{\lambda}|Xw^* - y|^2(X^\top y - X^\top Xw^*),$$

so we can write $w^* = X^\top a$ where

$$a = \frac{2}{\lambda}|Xw^* - y|^2(y - Xw^*).$$

We now compute the Hessian. For the first term, we use the product rule to obtain

$$\nabla_w 4|Xw - y|^2(X^\top Xw - X^\top y) = 8X^\top(Xw - y)(X^\top Xw - X^\top y)^\top + 4|Xw - y|^2 X^\top X$$

So the Hessian is

$$\nabla^2 J(w) = 8(X^\top Xw - X^\top y)(X^\top Xw - X^\top y)^\top + 4|Xw - y|^2 X^\top X + 2\lambda I.$$

The first two terms are positive semidefinite, and the last term is positive definite, so the sum is positive definite. Therefore, J is strictly convex and has one unique local minimum.

3. Consider the regularized regression problem

$$w^* = \arg\min_{w \in \mathbb{R}^d} \; \frac{1}{n}\sum_{i=1}^{n} L(w^\top X_i, y_i) + \lambda\,|w|^2$$

where $\lambda > 0$ and the cost function L is convex in its first argument. Prove that the optimal solution has the form $w^* = \sum_{i=1}^n a_i X_i$. If the cost function is not convex, does the optimal solution always have the form $w^* = \sum_{i=1}^n a_i X_i$? Justify your answer.

Solution: The expression $w_* = \sum_{i=1}^n \alpha_i X_i$ writes w_* as a linear combination of the columns of $X^\top$. Suppose the optimal weight vector w_* does not lie in the subspace spanned by the columns of $X^\top$ and so can be written as $w'_* = \sum_{i=1}^n \alpha_i X_i + v = w_* + v$, where $v^\top X_i = 0$ for X_i's. Using w'_* as our predictor, our cost function becomes

$$\frac{1}{n}\sum_{i=1}^{n} L((w_*^\top + v^\top)x_i, y_i) + \mu\|w + v\|_2^2$$

$$\frac{1}{n}\sum_{i=1}^{n} L((w_*^\top)x_i, y_i) + \mu(\|w\|_2^2 + \|v\|_2^2)$$

v, being orthogonal to w and the x_i's, should be set to 0 to minimize the objective function. Hence, the optimal solution has the form $w_* = \sum_{i=1}^n \alpha_i x_i$, which is true in the cases of both convex and non-convex cost functions.

4 Wine Classification with Logistic Regression

The wine dataset given to you as part of the homework in `data.mat` consists of 6,497 data points, each having 12 features. The description of these features is provided in `data.mat`. The dataset includes a training set of 6,000 data points and a test set of 497 data points. Your classifier needs to predict whether a wine is white (class label 0) or red (class label 1).

For this homework, you need to do the following.

1. Derive and write down the batch gradient descent update equation for logistic regression with ℓ_2 regularization. (Not Newton's method, but the slower batch gradient descent.)

   Choose a reasonable regularization parameter value and a reasonable learning rate. Run your algorithm and plot the cost function as a function of the number of iterations. (As this is batch descent, one "iteration" should use every sample point once.)

2. Derive and write down the stochastic gradient descent update equation for logistic regression with ℓ_2 regularization. Choose a suitable learning rate. Run your algorithm and plot the cost function as a function of the number of iterations—where now each "iteration" uses *just one* sample point.

   Comment on the differences between the convergence of batch and stochastic gradient descent.

3. Instead of a constant learning rate ϵ, repeat part 2 where the learning rate decreases as $\epsilon \propto 1/t$ for the t^{th} iteration. Plot the cost function vs. the number of iterations. Is this strategy better than having a constant ϵ?

4. Finally, train your classifier on the entire training set. Submit your results to Kaggle. Your classifier, when given the test points, should output a CSV file. (There is a sample one on Kaggle.) You'll upload this CSV file to Kaggle where it'll be scored with both a public test set, and a private test set. You will be able to see only your public score. You can only submit twice per day, so get started early! In your writeup, for this problem, report your Kaggle username, the best score you achieved on Kaggle, and a short writeup describing what you did to achieve that score.

IMPORTANT: Do NOT use any software package for logistic regression that you didn't write yourself!

5 Real World Spam Classification

Motivation: After taking CS 189 or CS 289A, students should be able to wrestle with "real-world" data and problems. These issues might be deeply technical and require a theoretical background, or might demand specific domain knowledge. Here is an example that a past TA encountered.

Daniel (a past CS 189 TA) interned as an anti-spam product manager for an email service provider. His company uses a linear SVM to predict whether an incoming spam message is spam or ham. He notices that the number of spam messages received tends to spike upwards a few minutes before and after midnight. Eager to obtain a return offer, he adds the timestamp of the received message, stored as number of milliseconds since the previous midnight, to each feature vector for the SVM to train on, in hopes that the ML model will identify the abnormal spike in spam volume at night. To his dismay, after testing with the new feature, Daniel discovers that the linear SVM's success rate barely improves.

Why can't the linear SVM utilize the new feature well, and what can Daniel do to improve his results? Daniel is unfortunately limited to only a quadratic kernel. This is an actual interview question Daniel received for a machine learning engineering position!

Write a short explanation. This question is open ended and there can be many correct answers.

Solution: Daniel unfortunately made a mistake by storing the time stamp as the number of milliseconds since the previous midnight. This raw format would have worked perfectly fine if the spam spike was at a different time of day (say 5 PM) since the magnitude of the time stamps would be relatively similar. Thus, identifying the spike would be simple for a linear decision boundary. However, since the spike is at midnight, the magnitude will vary widely since 11:59 PM is 86,340,000 ms while 12:00 AM is 0 ms. Essentially, time is stored in a modular fashion and the linear decision boundary with no regularization and kernel will do poorly. However, we can provide our own transformations to the feature. Take the time stamp in milliseconds since the previous midnight and normalize it to range uniformly between $\theta = [0, 2\pi]$. Now, use θ to generate 2 coordinates x= $\cos(\theta)$, y= $\sin(\theta)$. Notice that we have mapped our time stamp into a unit circle, similar to how an analog clock behaves. Note that in this format, 11:59 PM and 12:00 AM are close if we compare the L2 norm between their two points on the unit circle! Now, we can apply a linear SVM and predict when the spam spike will be.

There are other ways to improve performance:

1. Add 15 minutes to each time stamp, mod 24 hours, then use a linear SVM.
2. Hack together a binary feature indicating whether a time stamp is "close to midnight" or not, then use linear SVM.
3. Remap times to the interval [0, 1) such that 12 PM maps to 0 and 12 AM maps to 1/2. Then,

use a quadratic kernel.