CS189: Introduction to Machine Learning

Homework 4

Due: March 31, 2016 @ 11:59 PM

Homework parties:
Tuesday, March 29th, 2016 (4:00-6:00PM, Wozniak Lounge)
Wednesday, March 30th, 2016 (5:00-6:30PM, 510 Soda)

Submission Instructions

You will submit this assignment to both **bCourses** and **Gradescope**. There will also be a **Kaggle** competition.

In your submission to **Gradescope**, include:

1. A pdf writeup with answers to all the questions and your plots. Include in the pdf a copy of your code for each problem (code for problems 1, 2, and 3). Include also your Kaggle score for the spam test set.

In you submission to **bCourses**, include:

2. A zip archive containing your code for each problem, and a README with instructions on how to run your code. Please include the pdf writeup in this zip archive as well.

Submitting to **Kaggle**

3. Submit a csv file with your best predictions for the examples in the Spam test set (Problem 3) to Kaggle, just like in homeworks 1 and 3.

Note: The Kaggle invite links and more instructional details will be poster on Piazza. Good luck!

Problem 1: Ridge Regression

In this problem we will return to predicting the median home value in a given Census area by extending linear regression. The data is in housing_data.mat and it comes from `ftp://rcom.univie.ac.at/mirrors/lib.stat.cmu.edu/datasets/.index.html`. There are only 8 features for each data point; you can read about the features in **housing_data_source.txt**.

a) You are given a training set $\{(\mathbf{X}_i, y_i)\}_{i=1}^n$. Let X be the design matrix (i.e., the matrix whose i^{th} row is $\mathbf{X}_i^\top$), and let $\mathbf{y}$ be the column vector whose i^{th} entry is y_i. Let $\mathbf{1}$ be an $n \times 1$ column vector of ones.

Define $\bar{\mathbf{x}} = \frac{1}{n}\sum_i \mathbf{X}_i$ and $\bar{y} = \frac{1}{n}\sum_i y_i$. Assume that the input data has been centered, so that $\bar{\mathbf{x}} = \mathbf{0}$. Show that the optimizer of the following loss function $J(\mathbf{w}, \alpha)$

$$J(\mathbf{w}, \alpha) = (X\mathbf{w} + \alpha\mathbf{1} - \mathbf{y})^\top (X\mathbf{w} + \alpha\mathbf{1} - \mathbf{y}) + \lambda \mathbf{w}^\top \mathbf{w}$$

is given by

$$\hat{\alpha} = \bar{y}, \quad \hat{\mathbf{w}} = (X^\top X + \lambda \mathbf{I})^{-1} X^\top \mathbf{y}.$$

Solution: Given $J(\mathbf{w}, \alpha) = (X\mathbf{w} + \alpha\mathbf{1} - \mathbf{y})^\top (X\mathbf{w} + \alpha\mathbf{1} - \mathbf{y}) + \lambda \mathbf{w}^\top \mathbf{w}$, where design matrix X is $n \times m$, $\mathbf{w}$ is $m \times 1$, y is $n \times 1$, $\mathbf{1}$ is a vector of n 1's. The full expansion of $J(\mathbf{w}, \alpha)$ is as follows:

$$\begin{aligned} J(\mathbf{w}, \alpha) =& \mathbf{y}^\top \mathbf{y} - \mathbf{y}^\top X\mathbf{w} - \mathbf{y}^\top \alpha\mathbf{1} \\ & - \mathbf{w}^\top X^\top \mathbf{y} + \mathbf{w}^\top X^\top X\mathbf{w} + \mathbf{w}^\top X^\top \alpha\mathbf{1} \\ & - \mathbf{1}^\top \alpha\mathbf{y} + \mathbf{1}^\top \alpha X\mathbf{w} + \mathbf{1}^\top \alpha^2 \mathbf{1} \\ & + \lambda \mathbf{w}^\top \mathbf{w} \end{aligned} \tag{1}$$

Take derivatives with respect to scalar α, set to zero to find optimizer:

$$\begin{aligned} 0 &= -\mathbf{y}^\top \mathbf{1} + \mathbf{w}^\top X^\top \mathbf{1} - \mathbf{1}^\top \mathbf{y} + \mathbf{1}^\top X\mathbf{w} + 2\alpha \mathbf{1}^\top \mathbf{1} && (2) \\ &= -2\mathbf{y}^\top \mathbf{1} + 2\mathbf{w}^\top X^\top \mathbf{1} + 2\alpha \mathbf{1}^\top \mathbf{1} && (3) \\ &= -\mathbf{y}^\top \mathbf{1} + \mathbf{w}^\top X^\top \mathbf{1} + \alpha \mathbf{1}^\top \mathbf{1} && (4) \end{aligned}$$

notice that $\mathbf{1}^\top \mathbf{1} = n$, $\mathbf{y}^\top \mathbf{1} = \sum_{i=1}^n y_i$ and that $X^\top \mathbf{1} = 0$ since $\bar{x} = 0$. This is since $\bar{x} = \frac{1}{n}\sum x_i$, where i is the i-th row, implies that the sum of all the rows is zero. $X^\top \mathbf{1}$ sums all the rows in X. Thus,

$$\sum_{i=1}^n y_i = 0 + n\alpha \quad \Rightarrow \quad \frac{1}{n}\sum_{i=1}^n y_i = \alpha \tag{5}$$

Using the original $L(\mathbf{w}, \alpha)$ function, take derivatives with respect to $n \times 1$ vector $\mathbf{w}$.

$$\begin{aligned} 0 &= -X^\top \mathbf{y} - X^\top \mathbf{y} + 2X^\top X\mathbf{w} + X^\top \alpha\mathbf{1} + X^\top \alpha\mathbf{1} + 2\lambda I\mathbf{w} && (6) \\ &= -2X^\top \mathbf{y} + 2X^\top X\mathbf{w} + 2\alpha X^\top \mathbf{1} + 2\lambda I\mathbf{w} && (7) \end{aligned}$$

Remember that $X^\top \mathbf{1} = 0$ since $\bar{x} = 0$

$$0 = -X^\top \mathbf{y} + X^\top X \mathbf{w} + \lambda I \mathbf{w} \tag{8}$$

$$X^\top \mathbf{y} = (X^\top X + \lambda I)\mathbf{w} \tag{9}$$

$$\mathbf{w} = (X^\top X + \lambda I)^{-1} X^\top \mathbf{y} \tag{10}$$

b) Using the result from part (a):

i) Implement a ridge regression model with least squares. Include your code in the submission.

ii) Use 10-fold cross-validation to tune the hyperparameter λ. Using the tuned value of λ, test your trained model on the validation set. What is the residual sum of squares (RSS) on the validation set? How does this compare to the result from HW3?

iii) Plot the regression coefficients $\mathbf{w}$. How do these compare to the result from HW3?

Solution: Plotting the training vs. validation error for the simple linear regression, it is clear that the model complexity is low enough so that overfitting is not an issue. Thus, applying ridge regression does not help to improve validation error. Before applying regularization methods, it is important to check that the model is actually running at the risk of overfitting. Otherwise, we are simply weakening the strength of our model without reason. As a result, the RSS should not have improved. At the optimal, the weights should either be closed to the original model or slightly smaller.

Problem 2: Logistic Regression

Let $\{(\mathbf{X}_i, y_i)\}_{i=1}^n$ be a training set, where $\mathbf{X}_i \in \mathbb{R}^d$ and $y_i \in \{0, 1\}$. The empirical risk (based on the logistic loss L, aka the cross-entropy loss, presented in lecture 10) is

$$R(\mathbf{w}) = -\sum_{i=1}^{n} \Big(y_i \ln s(\mathbf{w}^\top \mathbf{X}_i) + (1 - y_i) \ln(1 - s(\mathbf{w}^\top \mathbf{X}_i)) \Big), \quad \text{where} \quad s(\gamma) = \frac{1}{1 + e^{-\gamma}}.$$

(The sign is reversed from the log likelihood because we are minimizing the empirical risk, rather than maximizing the log likelihood.) Recall from lecture 10 that the batch gradient ascent rule to maximize $-R(\mathrm{w})$ is:

$$\mathbf{w} \leftarrow \mathbf{w} - \epsilon \nabla_{\mathbf{w}} R = \mathbf{w} + \epsilon \sum_{i=1}^{n} \Big(y_i - s(\mathbf{X}_i^\top \mathbf{w}) \Big) \mathbf{X}_i$$

and the stochastic gradient ascent rule is

$$\mathbf{w} \leftarrow \mathbf{w} - \epsilon \nabla_{\mathbf{w}} L_i = \mathbf{w} + \epsilon \Big(y_i - s(\mathbf{X}_i^\top \mathbf{w}) \Big) \mathbf{X}_i.$$

In this problem, you will minimize the empirical risk on a small training set. We have four data points in $\mathbb{R}^2$, two of class 1, and two of class 0. Here is the data (you may want to draw this on paper to see what it looks like):

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

Here, X is the design matrix (training data); each row $\mathbf{X}_i^\top$ of X is a transposed data point.

Notice that the data cannot be separated by a boundary that goes through the origin. To account for this, you should *append* 1 to the $\mathbf{X}_i$ vectors and fit a three-dimensional $\mathbf{w}$ vector that includes an offset term.

1. We will now perform batch gradient ascent for a few iterations. Set the learning rate $\epsilon = 1$. We are given that $\mathbf{w}^{(0)} = \begin{bmatrix} -2 & 1 & 0 \end{bmatrix}^\top$. $\mathbf{w}^{(0)}$ is the value of $\mathbf{w}$ at the 0^{th} iteration.

   (a) State the value of $R(\mathbf{w}^{(0)})$.

   Solution: $R(\mathbf{w}^{(0)}) = 1.9884$

   (b) State the value of $\boldsymbol{\mu}^{(0)}$. This should be an $n-$dimensional vector, with $\mu_i^{(0)} = P(Y = 1 | X = \mathbf{x}_i)$.

   Solution:

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

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

Solution:

$$\mathbf{w}^{(1)} = \begin{bmatrix} -2.0000 \\ 0.9491 \\ -0.6836 \end{bmatrix}$$

(d) State the value of $R(\mathbf{w}^{(1)})$.

Solution: $R(\mathbf{w}^{(1)}) = 1.7206$

(e) State the value of $\boldsymbol{\mu}^{(1)}$.

Solution:

$$\boldsymbol{\mu}^{(1)} = \begin{bmatrix} 0.8969 \\ 0.5408 \\ 0.5660 \\ 0.1500 \end{bmatrix}$$

(f) After performing a second iteration, state the value of $\mathbf{w}^{(2)}$.

Solution:

$$\mathbf{w}^{(2)} = \begin{bmatrix} -1.6908 \\ 1.9198 \\ -0.8374 \end{bmatrix}$$

(g) State the value of $R(\mathbf{w}^{(2)})$.

Solution: $R(\mathbf{w}^{(2)}) = 1.8547$

Problem 3: Spam classification using Logistic Regression

The spam dataset given to you as part of the homework in `spam.mat` consists of 5172 email messages, from which 32 features have been extracted for you. Please use the standard features for the first four parts of this problem. In part 5, you are asked to predict the labels of the test set in `test.mat` and submit the predictions to Kaggle. Feel free to use your own featurizes to boost up your score!

One can imagine performing several kinds of preprocessing to this data matrix. Try each of the following separately:

i) Standardize each column to have mean 0 and unit variance.

ii) Transform the features using $X_{ij} \leftarrow \log(X_{ij}+0.1)$, where the X_{ij}'s are the entries of the design matrix.

iii) Binarize the features using $X_{ij} \leftarrow \mathbb{I}(X_{ij} > 0)$. $\mathbb{I}$ denotes an indicator variable.

WARNING: You are NOT supposed to use any kind of software package for logistic regression! Implement all the math yourself.

1. Implement logistic regression to classify the spam data. Use batch gradient descent. You will probably need to tune the step size carefully to avoid numerical issues and to avoid a diverging training risk.

   Plot the training risk (the empirical risk of the training set) vs. the number of iterations. You should have one plot for each preprocessing method.

   Note: One batch gradient descent iteration amounts to scanning through the whole training data and computing the full gradient.

   Solution: Plots should be smooth and strictly decreasing. The risk should converge.

2. Implement logistic regression to classify the spam data again, this time using stochastic gradient descent. Plot the training risk vs. number of iterations. You should have one plot for each preprocessing method. How are the plots different from part (1)?

   Note: One stochastic gradient descent iteration amounts to computing the gradient using one data point.

   Solution: Plots should be rough. Risk should not converge; risk should oscillate within a band of values at the end.

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 training risk vs number of iterations. Is this strategy better than having a constant ϵ? You should have one plot for each preprocessing method.

   Solution: This is better, as it solves the problem of the risk oscillating within a final band of values. The risk should now converge. It requires a bit more sophisticated learning rate policy than just setting $\epsilon_t = \epsilon_0/t$ (e.g. $\epsilon_t = \frac{\alpha}{\beta t+\gamma}$ where α, β, and γ are hyperparameters) for the final risk to be roughly the same as in the previous case.

4. (a) Now let's use kernel logistic regression with a polynomial kernel of degree 2. Our risk is still the same as in problem 2, but we rewrite each $s(\mathbf{X}_i^\top \mathbf{w})$ as

$$s\left(\sum_{i=1}^{n} a_i k(\mathbf{x}_i, \mathbf{x})\right), \quad \text{where} \quad k(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T\mathbf{z} + \rho)^2, \quad \rho \in \mathbb{R}, \quad \text{and} \quad \mathbf{a} \in \mathbb{R}^n.$$

The stochastic gradient descent update rule for kernel logistic regression is

$$a_i \leftarrow a_i + \epsilon(y_i - s((K\mathbf{a})_i))$$

where K is the $n \times n$ *kernel matrix* with components $K_{ij} = k(\mathbf{X}_i, \mathbf{X}_j)$.

Repeat part (2), using the best preprocessing method you found, using kernel logistic *ridge* regression (note how this differs from the normal stochastic gradient descent update rule presented above):

$$\begin{aligned} a_i &\leftarrow a_i - \epsilon\lambda a_i + \epsilon(y_i - s((K\mathbf{a})_i)) \\ a_h &\leftarrow a_h - \epsilon\lambda a_h, \text{ for all } h \neq i \end{aligned}$$

Use whichever learning rate scheme you wish. We suggest trying $\lambda = 10^{-3}$ first, but you may optionally adjust it. Use crossvalidation to find the optimal value of ρ, and report your findings. Generate a plot of training risk vs. number of iterations, and another plot of validation risk vs. number of iterations. Use a split of 2/3 training data, 1/3 validation data.

If numerical issues necessitate it, feel free to clamp the value of $(K\mathbf{a})_i$.

Solution: Since the quadratic kernel is more general than the linear kernel, it should yield a lower training risk. However, training of the quadratic kernel can be fairly difficult, so a result showing some convergence is fine even if the final risk remains high. For the same reason, the optimal value of ρ has much variation, but it should be approximately between 10 and 100.

(b) Repeat the same experiment with the linear kernel $k(\mathbf{x}, \mathbf{z}) = \mathbf{x}^T\mathbf{z} + \rho$. Does the quadratic kernel overfit the data? For each kernel, should you decrease or increase λ to try to improve performance?

Solution: The linear kernel behaves better, so this is easier to train. Again, the optimal ρ should be approximately between 10 and 100. Neither kernel overfits the data, so you could try *decreasing* λ on both to increase performance.

5. Using the best classifier you found, predict the labels of the test set and submit your predictions to Kaggle. You can use the default features in `test.mat`,or if you used an additional featurizer, describe your implementation. Include you Kaggle score also in the writeup.

Problem 4: Revisiting Logistic Regression

Recall that in logistic regression, the logistic function, $s(z) = \frac{1}{1+\exp(-z)}$, is used to map output values between $[0, 1]$. Consider the function $g(z) = \frac{\tanh z+1}{2}$, where $\tanh z = 2s(2z) - 1$, that also maps outputs between $[0, 1]$. In this problem, we will explore using $g(z)$ instead of the logistic function in logistic regression.

(a) Show that $g(z) = \frac{e^z - e^{-z}}{2(e^z + e^{-z})} + \frac{1}{2}$.

Solution:

$$\tanh(z) = \frac{2e^{2z}}{1 + e^{2z}} - 1 = \frac{e^{2z} - 1}{1 + e^{2z}} = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{11}$$

$$\Rightarrow g(z) = \frac{e^z - e^{-z}}{2\left(e^z + e^{-z}\right)} + \frac{1}{2} \tag{12}$$

(b) Find and expression for $g'(z) = \frac{d}{dz}g(z)$ in terms of $\tanh z$.

Solution:

$$\frac{d}{dz}\left(\frac{e^z - e^{-z}}{2\left(e^z + e^{-z}\right)} + \frac{1}{2}\right) = \frac{1}{2}\frac{\left(e^z + e^{-1}\right)\frac{d}{dz}\left(e^z - e^{-z}\right) - \left(e^z - e^{-z}\right)\frac{d}{dz}\left(e^z + e^{-z}\right)}{\left(e^z + e^{-z}\right)^2} \tag{13}$$

$$= \frac{\left(e^z + e^{-z}\right)^2 - \left(e^z - e^{-z}\right)^2}{2\left(e^z + e^{-z}\right)^2} \tag{14}$$

$$= \frac{1}{2} - \frac{\left(e^z - e^{-z}\right)^2}{2\left(e^z + e^{-z}\right)^2} \tag{15}$$

$$= \frac{1}{2} - \frac{\tanh^2(z)}{2} \tag{16}$$

(c) Using the objective function $J(\mathbf{w}) = \sum_{i=1}^{n} y_i \ln\left(g(\mathbf{X}_i \cdot \mathbf{w})\right) + (1 - y_i)\ln\left(1 - g(\mathbf{X}_i \cdot \mathbf{w})\right)$, find the appropriate batch gradient ascent update function. Use λ as your learning rate.

Solution:

$$\frac{d}{d\mathbf{w}}J(\mathbf{w}) = \sum_{i=1}^{n} \frac{y_i}{g(\mathbf{X}_i \cdot \mathbf{w})} \frac{d}{d\mathbf{w}} g(\mathbf{X}_i \cdot \mathbf{w}) - \frac{1-y_i}{1-g(\mathbf{X}_i \cdot \mathbf{w})} \frac{d}{d\mathbf{w}} g(\mathbf{X}_i \cdot \mathbf{w}) \tag{17}$$

$$= \sum_{i=1}^{n} \left(\frac{y_i}{g(\mathbf{X}_i \cdot \mathbf{w})} - \frac{1-y_i}{1-g(\mathbf{X}_i \cdot \mathbf{w})} \right) \frac{d}{d\mathbf{w}} g(\mathbf{X}_i \cdot \mathbf{w}) \tag{18}$$

$$= \sum_{i=1}^{n} \left(\frac{y_i}{g(\mathbf{X}_i \cdot \mathbf{w})} - \frac{1-y_i}{1-g(\mathbf{X}_i \cdot w)} \right) g'(\mathbf{X}_i \cdot \mathbf{w}) \mathbf{X}_i \tag{19}$$

$$= \frac{1}{2} \sum_{i=1}^{n} \left(\frac{2y_i}{\tanh(\mathbf{X}_i \cdot \mathbf{w}) + 1} - \frac{2y_i - 2}{\tanh(\mathbf{X}_i \cdot \mathbf{w})} \right) \left(1 - \tanh^2(z)\right) \mathbf{X}_i \tag{20}$$

$$\Rightarrow \mathbf{w} \leftarrow \mathbf{w} + \frac{\lambda}{2} \sum_{i=1}^{n} \left(\frac{2y_i}{\tanh(\mathbf{X}_i \cdot \mathbf{w}) + 1} - \frac{2y_i - 2}{\tanh(\mathbf{X}_i \cdot \mathbf{w})} \right) \left(1 - \tanh^2(z)\right) \mathbf{X}_i \tag{21}$$

Problem 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 theoretical background, or might demand specific domain knowledge. Here is an example that a past TA encountered.

Daniel recently 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 timestamp 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 timestamp 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 timestamp 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 L_2 norm between their two points on the unit circle! Now, we can apply a linear SVM and predict when the spam spike will be.

Other ways to improve performance:

1. Map times from 12 AM to 12 PM to $[-1, 1]$, then use a quadratic kernel.
2. Add 15 minutes to each timestamp, mod 24 hours, then use a linear SVM.
3. Hack together a binary feature indicating whether a timestamp is "close to midnight" or not, then use linear SVM.