CSE 517a: Homework #3

Due on Monday, March 31st, 2015

2:30pm

Contents

Loss functions

Loss Function Optimization

1. Derive the gradient update (with stepsize c) for your weight vector w for each of the following loss functions: (here: $\|w\|_2^2 = w^\top w$ and $|w| = \sum_{\alpha=1}^d |w_\alpha|$, also λ and C are non-negative constants.)

   (a) Ridge Regression: $\mathcal{L}(w) = \sum_{i=1}^n (w^\top x_i - y_i)^2 + \lambda \|w\|_2^2$

   (b) Lasso Regression: $\mathcal{L}(w) = \sum_{i=1}^n (w^\top x_i - y_i)^2 + \lambda |w|$

   (c) Logistic Regression ($y_i \in \{+1, -1\}$): $\mathcal{L}(w) = \sum_{i=1}^n \log(1 + \exp(-y_i w^\top x_i))$

   (d) Linear Support Vector Machine ($y_i \in \{+1, -1\}$): $\mathcal{L}(w) = C \sum_{i=1}^n \max(1 - y_i w^\top x_i, 0) + \|w\|_2^2$

2. You suddenly have this vision, that it might be beneficial to pre-compute all inner-products in your data set. I.e. you store a $n \times n$ matrix $K_{ij} = x_i^\top x_j$.

   (a) Imagine you start the gradient descent procedure from above with initial weights $w^0 = 0$ (this is just to help intuition). Take the gradient update from 1d and show that after t gradient steps you can express the weight vector as $w^t = \sum_{i=1}^n \alpha_i^t x_i$ for some values α_i^t. (HINT: each gradient step update the α's, if the update is correct you can always write $w^t = \sum_{i=1}^n \alpha_i^t x_i$ starting from update $t = 0$ to any update $t > 0$)

   (b) Take this new definition $w = \sum_{i=1}^n \alpha_i x_i$ and substitute it into the loss function of 1d. You can now write the loss function as $\mathcal{L}(\alpha)$ where $\alpha = [\alpha_1 \dots \alpha_n]^\top$ (no need to force the vector representation in your formula). Further, write the loss $\mathcal{L}(\alpha)$ only using the precomputed matrix entries K_{ij}.

   (c) Can you derive a gradient descent update rule for 1d with respect to α_i?

Weighted Ridge Regression

Assume that in addition to your data $\{(x_1, y_1), \dots, (x_n, y_n)\}$ you also have weights $p_i \geq 0$ for each example. Let your loss function be

$$\mathcal{L}(w) = \sum_{i=1}^n p_i (w^\top x_i - y_i)^2 + \lambda w^\top w. \tag{1}$$

1. Rephrase eq. 1 in terms of the matrices $X = [x_1, \dots, x_n]^\top$, $Y = [y_1, \dots, y_n]^\top$ and the diagonal matrix $P = diag([p_1, \dots, p_n])$ (where the diag operator performs like the Matlab function with the same name.)

2. Derive a closed form solution for w. (You can use: $\frac{\partial(w^\top A)}{\partial w} = A$, $\frac{\partial(w^\top B w)}{\partial w} = Bw + B^\top w$ and $w^\top w = w^\top I w$ where I is the identity matrix.)

Newton's Method

Let us re-visit Logistic Regression, however with $y_i \in \{0, 1\}$.

1. Let $\sigma(a) = 1/(1 + e^{-a})$. Show that with these new labels the loss function can be written as

$$\mathcal{L}(w) = -\sum_{i=1}^{n} (y_i \log(\sigma(w^\top x_i)) + (1 - y_i) \log(1 - \sigma(w^\top x_i))) \tag{2}$$

2. Show that the gradient of $\mathcal{L}$ can be written as

$$\frac{\partial L}{\partial w} = -\sum_{i=1}^{n} (y_i - \sigma(w^\top x_i))x_i. \tag{3}$$

   (HINT: You can make use of the earlier result that $\frac{\partial \sigma(z)}{\partial z} = \sigma(z)(1 - \sigma(z))$.)

3. Let the $n \times n$ diagonal matrix $W_{ii} = \sigma(w^\top x_i)(1 - \sigma(w^\top x_i))$ and let $X = [x_1, \ldots, x_n]^\top$. Show that the Hessian matrix is $H = X^\top W X$. For which examples is W_{ii} large, for which is it small?

4. Write down the update rule for a newton step. Show that if you use the substitution $\vec{z}$ where $z_i = \vec{x}_i^\top w + \frac{1}{W_{ii}}(y_i - \sigma(w^\top x_i))$, you arrive at

$$w^{new} \leftarrow (X^\top W X)^{-1} X^\top W z. \tag{4}$$

5. Look at the result of Question 2 in the Weighted Ridge Regression section ($\lambda = 0$). Hold your breath and enjoy the moment of amazement. Why is this algorithm also called *Iteratively Reweighted Least Squares*?

Leave One Out Cross-Validation

In class we introduced LOOCV. Imagine we perform OLS, (the same loss as ridge regression with $\lambda = 0$). Remember that with the matrix X, Y we can write $w = (X^\top X)^{-1}X^\top Y$. Let us define $H = X(X^\top X)^{-1}X^\top$. Let $\hat{Y}$ be the predictions of Y (ie. $\hat{y}_i = w^\top x_i$).

1. Validate that $\hat{Y} = HY$.

2. Let us define $\hat{y}_i^{-i}$ as the prediction of y_i if we train on all data points *except* x_i. So the LOOCV loss is

$$LOOCV = \sum_{i=1}^{n}(y_i - \hat{y}_i^{-i})^2 \tag{5}$$

   What is the algorithmic complexity of computing LOOCV? Assume that a matrix inversion has complexity $O(d^3)$ (no need to focus on terms that are faster than $O(d^3)$).

3. Let us define a new set of labels:

$$z_i = \begin{cases} y_i, & i \neq k \\ \hat{y}_k^{-k}, & i = k \end{cases} \tag{6}$$

   In other words z_i is the standard label for all x_i except for one particular x_k, in which case z_k is the leave-one-out estimate y_k^{-k}. Show that $p_i = \hat{y}_i^{-k}$ minimizes

$$\sum_{i=1}^{n}(p_i - z_i)^2 \tag{7}$$

4. Express $\hat{y}_k^{-k}$ in terms of H and $Z = [z_1, \ldots, z_n]^\top$. (Your answer must involve both H and Z.)

5. Show that $\hat{y}_k - \hat{y}_k^{-k} = H_{kk}y_k - H_{kk}\hat{y}^{-k}$.

6. Show that

$$LOOCV = \sum_{k=1}^{n}\left(\frac{y_k - \hat{y}_k}{1 - H_{kk}}\right)^2. \tag{8}$$

   What is the algorithmic complexity of this expression?

7. (**Extra credit**) Open Matlab and load the data "cars.mat". Implement the function looreg.m which takes the input matrices xTr, yTr and outputs the LOOCV root mean-squared error. (The features of xTr are [Acceleration,Cylinders,Displacement,Horsepower,Model_Year,Weight]. The target yTr is MPG.) In this context, the root mean squared error (RMSE) is defined as

$$RMSE = \sqrt{\frac{1}{n}LOOCV}. \tag{9}$$

Programming

In this assignment you will build a spam filter. First, perform an "svn update" in your svntop directory. Then Download the spam data set from

- `http://www.cse.wustl.edu/~kilian/cse517a2015/hw3/spam_train.zip`
- `http://www.cse.wustl.edu/~kilian/cse517a2015/hw3/spam_train.mat`

and unzip the zip file in your hw3 directory. The *spam_train.zip* file contains the raw data. *data_train.mat* contains the pre-processed data. Type in *load data_train* Run *valsplit.m*, which generates a training data xTr, yTr and a validation set xTv, yTv for you. It is now time to implement your classifiers. We will always use gradient descent, but with various loss functions. **As always, make sure to add a test for every function you implement.**

1. Implement the function *ridge.m* which computes the loss and gradient for a particular data set xTr, yTr and a weight vector w. Make sure you don't forget to incorporate your regularization constant lambda. You can check your gradient with the following expression (it checks the difference between the interpolated- and the actual function value):

   ```
   err=checkgrad('ridge',zeros(size(xTr,1),1),1e-05,xTr,yTr,10);
   ```

   The return value should be very small (around 10^{-8}). (You can also use the checkgrad function for the later functions.)

2. Implement the function *grdescent.m* which performs gradient descent. Make sure to include the *tolerance* variable (you can use the function *norm(x)*). The first parameter of grdescent is a function which takes a weight vector and returns loss and gradient. In Matlab you can make inline functions e.g. with the following code (first line)

   ```
   f=@(w) ridge(w,xTr,yTr,0.1);
   w=grdescent(f,zeros(size(xTr,1),1),1e-06,1000);
   ```

   You can choose what kind of step-size you implement (e.g. constant, decreasing, line search,...). [HINT: Personally, I increase the stepsize by a factor of 1.01 each iteration where the loss goes down, and decrease it by a factor 0.5 if the loss went up. – if you are smart you also undo the last update in that case to make sure the loss decreases every iteration.]

3. Write the (almost trivial) function *linclassify* which returns the predictions for a vector w and a data set xTv.

4. Now call

   ```
   trainspamfilter(xTr,yTr);
   spamfilter(xTv,yTv);
   ```

   The first command trains a spam filter with ridge regression and saves the resulting weight vector in $w0.mat$. The second command will run your spam filter with the weights in $w0.mat$ over the validation data set. You can also run $vishw3$ to see where you still make mistakes.

5. Now implement the function *hinge.m*, which is the equivalent to *ridge* but with the hinge loss.

continued on next page. . .

6. Now implement the function *logistic.m*, which is the equivalent to *ridge* but with the log-loss (logistic regression). [By default the logistic loss does not take a regularization constant, but feel free to incorporate regularization if you want to.]

7. You can now run hw3rocs to see if your algorithms all work. You might have to fiddle with the STEPSIZE parameter at the very top (maybe set it to something very small initially (e.g. 1e-08) and work yourself up). If you want to, change the $trainspamfilter.m$ to a different loss function with different parameters.

8. (Optional) you can implement the function spamupdate to make small gradient steps during test time (basically you still correct the classifier after you made a mistake).

9. (Optional) If you take a look at the script "loaddata.m", you can see that there is a whole part that is never executed (after the *if 0*). You can change this to *if 1* if you want to modify the pre-processing of the data. For example, by default the data uses $2^{10} = 1024$ dimensional features. You could change this by increasing 10 to 11. You could also change the tokenize.py function (e.g. to include bigrams). A common trick is e.g. to remove stopwords. A full list is here `http://jmlr.org/papers/volume5/lewis04a/a11-smart-stop-list/english.stop` You can also include bi-grams or feature re-weighting with TFIDF:
   `http://en.wikipedia.org/wiki/Tfidf`