MLPR Assignment 2 – Predicting CT Slice Locations

Due: 4pm Tuesday 22 November 2016

The main purpose of this assignment is to assess your ability to use a matrix-based computational environment in the context of machine learning methods. While you *could* investigate all sorts of things beyond what is suggested, that is not required for a good mark.

Good Scholarly Practice: Please remember the University requirements for all assessed work for credit:
`http://web.inf.ed.ac.uk/infweb/admin/policies/academic-misconduct`
Furthermore, you are required to take reasonable measures to protect your assessed work from unauthorised access. For example, if you put any such work on a public repository then you must set access permissions appropriately (permitting access only to yourself).

While you may discuss the assignment with your peers at a high level, you should *never* share code or written materials. You can ask clarifying questions about what is expected on the class forum. However, please don't post anything that gives part of an answer. If in doubt, email your question instead.

Submission instructions: Submit your answers on paper to the ITO office in Forrest Hill by the deadline. This assignment is subject to the School's strict late policy:
`http://tinyurl.com/edinflate`

Provide relevant code and plots within your answers.
Machine learning papers and dissertations sometimes have pseudo-code boxes, but almost never include code in the main text. However, for this exercise, we need to see your code to assess what you've done. Please include all of your code. *If you don't include code for a part, you will lose marks.*

A lengthy report is not expected. A few sentences is plenty for most question parts.

Background: In this assignment you will perform some preliminary analysis of a dataset from the UCI machine learning repository. Some features have been extracted from slices of CT medical scans. There is a regression task: attempt to guess the location of a slice in the body from features of the scan. A little more information about the data is available online[1]:
`https://archive.ics.uci.edu/ml/datasets/Relative+location+of+CT+slices+on+axial+axis`
However, you should use the processed version of the dataset given below.

The data: Please use the data file located at:

```
/afs/inf.ed.ac.uk/group/teaching/mlprdata/ctslice/ct_data.mat
```

If you're working on DICE, use a symbolic link (`ln -s`) to "copy" this file into your home directory.

The patient IDs were removed from this version of the data, leaving 384 input features which were put in each of the "`X_...`" arrays. The corresponding CT scan slice location has been put in the "`y_...`" arrays. I shifted and scaled the "`y_...`" location values for the version of the data that you are using. The shift and scaling was chosen to make the training locations have zero mean and unit variance.

The first 73 patients were put in the `_train` arrays, the next 12 in the `_val` arrays, and the final 12 in the `_test` arrays. Please use this training, validation, test split as given. *Do not shuffle the data further in this assignment.*

The code: There is some code on the website along with this assignment, which you will need for the questions below. There are two versions, one for Matlab/Octave and one for Python.

---

1. Some of the description on the UCI webpage doesn't seem to match the dataset. For example there are 97 unique patient identifiers in the dataset, although apparently there were only 74 different patients.

Questions:

1. **Getting started [15 marks]:**

   As with each question, please report the few lines of code required to perform each requested step.

   Load the data into Matlab/Octave (using `load`) or Python (using `scipy.io.loadmat`). Python users may want to use "`squeeze_me=True`" so that `y_train.shape` is `(N,)` rather than `(N,1)`.

   a) Verify that (up to numerical rounding errors) the mean of the training positions in `y_train` is zero. The mean of the 5,785 positions in the `y_val` array is not zero. Report its mean with a "standard error", temporarily assuming that each entry is independent. For comparison, also report the mean with a standard error of the first 5,785 entries in the `y_train`. Explain why these standard error bars do not reliably indicate what the average of locations in future CT slice data will be.

      (For the remainder of this assignment you don't need to report error bars on your results. If I were working on this application I would definitely work on estimating error bars. But given the time available for this assignment, we're just going to acknowledge that the issue is slightly tricky here, and move on.)

   b) Some of the input features take on the same value for every item in the training set. Identify these features, and remove them from the input matrices in the training, validation, and testing sets. *Use these modified input arrays for the rest of the assignment.*

      Report which columns of the `X_...` arrays you are removing. Report 1-based indexes if you are using Matlab/Octave, or 0-based indexes if you are using Python.

2. **Linear regression baseline [15 marks]:**

   Using the least squares operator `\` in Matlab/Octave, or `numpy.linalg.leastsq` in Python, write a short function "`fit_linreg(X, yy, alpha)`" that fits the linear regression model

   $$f(\mathbf{x}; \mathbf{w}, b) = \mathbf{w}^\top \mathbf{x} + b,$$

   by minimizing the cost function:

   $$E(\mathbf{w}, b) = \alpha \mathbf{w}^\top \mathbf{w} + \sum_n (f(\mathbf{x}^{(n)}; \mathbf{w}, b) - y^{(n)})^2,$$

   with regularization constant α. As discussed in lectures, fitting a bias parameter b and incorporating the regularization constant can both be achieved by augmenting the original data arrays. You should only regularize the weights $\mathbf{w}$ and not the bias b.

   (In lectures I used λ for the regularization constant, matching Murphy and others. However, `lambda` is a reserved word in Python, so I swapped to `alpha` for my code.)

   Use your function to fit weights and a bias to `X_train` and `y_train`. Use $\alpha = 10$.

   We can fit the same model with a gradient-based optimizer. The support code has a function `fit_linreg_gradopt`, which you should look at and try.

   Report the root-mean-square errors on the training and validation sets for the parameters fitted using both your `fit_linreg` and the provided `fit_linreg_gradopt`. Do you get exactly the same results? Why or why not?

3. **Decreasing and increasing the input dimensionality [20 marks]:**

   a) It's possible for even a linear regression 'baseline' to overfit when there are a lot of features. Reducing the dimensionality of the data could help in these cases. As a quick check that we don't have far too many features, we will try reducing the dimensionality of the data using Principal Components Analysis (PCA).

   The `pca_zm_proj` function provides a projection matrix assuming your data is zero-mean. First compute the mean training feature vector:

   ```
   X_mu = mean(X_train, 1); % Matlab/Octave
   X_mu = np.mean(X_train, 0) # Python
   ```

   Obtain zero-mean versions of the training and validation sets by subtracting `X_mu` from every row of both `X_train` and `X_val`. You can then use `pca_zm_proj` to obtain a PCA projection matrix V from the zero mean version of the training set. You can then apply the projection to the zero-mean versions of the training and validation sets to reduce their dimensionality.

   Use your `fit_linreg` function from the previous part to fit linear regression models to $K=10$ and $K=100$ dimensional versions of your data. As before, use regularization with $\alpha = 10$. Why is it likely that the training error will go up? Report the training and validation root mean square errors.

   In all future parts, please go back to using the original features, not the zero-mean versions that you created for this part.

   b) Plot a histogram of the values of the 46th feature (index 45 in python). Have a look at the histograms for some of the other features as well. What do you notice? What fraction of the values in the whole `X_train` matrix are equal to `-0.25` and `0`?

   It is sometimes useful to add extra binary features, encoding special values. To try this idea, use the `aug_fn` function, created with one of the following lines of code:

   ```
   aug_fn = @(X) [X X==0 X<0]; % Matlab
   aug_fn = lambda X: np.hstack([X, X==0, X<0]) # Python
   ```

   You should apply this function to the training inputs, and refit the linear regression model. Report the root mean square error for the system applied to the augmented training set and an augmented validation set. Why would we expect the training error to go down?

4. **Invented classification tasks [20 marks]:**

   It is often easier to work with binary data than real-valued data: we don't have to think so hard about how the values might be distributed, and how we might process them. We will invent some binary classification tasks, and fit these.

   a) We will pick 10 positions within the range of training positions, and use each of these to define a classification task:

   ```
   % Matlab
   K = 10;
   mx = max(y_train); mn = min(y_train); hh = (mx-mn)/(K+1);
   thresholds = mn+hh:hh:mx-hh;
   for kk = 1:K
       labels = y_train > thresholds(kk);
       % ... fit logistic regression to these labels
   end
   ```

```
# Python
K = 10 # number of thresholded classification problems to fit
mx = np.max(y_train); mn = np.min(y_train); hh = (mx-mn)/(K+1)
thresholds = np.linspace(mn+hh, mx-hh, num=K, endpoint=True)
for kk in range(K):
    labels = y_train > thresholds[kk]
    # ... fit logistic regression to these labels
```

The logistic regression cost function and gradients are provided with the assignment in the function `logreg_cost`. It is analogous to the `linreg_cost` function for least-squares regression, which is used by the `fit_linreg_gradopt` function that you've already used.

Fit logistic regression to the each of the 10 classification tasks above with $\alpha = 10$.

Given a feature vector, we can now obtain 10 different probabilities, the predictions of the 10 logistic regression models. Transform both the training and validation input matrices into new matrices with 10 columns, containing the predictions of your 10 logistic regression fits. You don't need to loop over the rows of `X_train` or `X_val`, you can use array-based operations to make the logistic regression predictions for every datapoint at once.

Fit a regularized linear regression model ($\alpha = 10$) to your transformed 10-dimensional training set. Report the training and validation root mean square errors of this model.

You previously considered PCA as a way to reduce the dimensionality of the data. Briefly comment on the differences between the dimensionality reduction method in this part and PCA, which lead to such different performance.

5. **Small neural network [10 marks]:**

   In Question 4 you fitted a small neural network. The logistic regression classifiers are sigmoidal hidden units, and a linear output unit predicts the outputs. However, you didn't fit the parameters jointly to the obvious least squares cost function. A least squares cost function and gradients for this neural network are implemented in the `nn_cost` function provided.

   Try fitting the neural network model to the training set, with a) a sensible random initialization of the parameters; b) the parameters initialized using the fits made in Q4.

   Does one initialization strategy work better than the other? Does fitting the neural network jointly work better than the procedure in Q4?

6. **What next? [20 marks]** Try to improve regression performance. Explain what you tried, why you thought it might work better, how you evaluated your idea, and what you found. Your method does *not* have to actually work better to get marks!

   Do not write more than a page for this part, including figures. The bulk of the marks available for this part are for trying something sensible, which can be simple, and evaluating it sensibly. You are advised to *not* spend a long time on this part.