MLPR Assignment 2 – Predicting CT Slice Locations

Due: 4pm Tuesday 20 November 2018

The primary purpose of this assignment is to assess your ability to use a matrix-based computational environment in the context of machine learning methods. The highest marks will go to those who also give excellent *concise* but precise explanations where asked, and do something interesting and well-explained for the last part. Higher marks won't be obtained for answering things that aren't asked for. Keep your answers short and to the point!

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).

You must work individually on this assignment. You may discuss your high level approach, but you must *never* share code or written materials. Don't post questions on this assignment using the class forum. Email your question to me instead. In response, I might post minor clarifications on the forum, or would email the class list with any major corrections.

Allowable code: You must use Matlab/Octave or Python+NumPy+Matplotlib. You may only use your own code written in the base packages, the support code, and no other libraries. You may not use SciPy, except for the `minimize` function imported by the support code.

Provide relevant code and plots within your answers. For this artificial exercise, we need to see your code to assess what you've done. Please include all of your code, split up into the parts relevant for each question. *If you don't include the code or plots you made for a part, you will lose marks.*

You should not write a lengthy report. Please label clearly which question part you are answering, and only provide what is requested, along with any code you used to generate your answers. Do not list all of your code in one block at the end and expect your marker to identify all of the relevant parts. Split it up as requested. If you don't follow these instructions, you will lose marks. Your answers *must* be in a PDF file called `report2.pdf`. Documents in other formats (e.g., `.docx`, `.ipynb`) will not be marked.

Submission instructions:
Please do not put your name in the report, just your student number. Call your document `report2.pdf` and submit it from a DICE machine using *exactly* this command:

```
submit mlpr cw2 report2.pdf
```

You can re-submit as many times as you like *until the submission deadline*. Don't leave it until the last minute. Technical difficulties are not a valid reason for late submission.

This assignment is subject to the School's late policy: `http://tinyurl.com/edinflate`
As in the policy, please do not email Iain about extensions, but follow the instructions.

If you do not submit anything before the deadline, you may submit *exactly once* after the deadline. Late penalties are applied as in the School policy. If you submit both before and after the deadline, we can't guarantee which version will be marked, and you are likely to get a late penalty regardless. Don't submit both before and after the deadline!

Queries: If you think there's a problem with the assignment or support code, email Iain rather than asking on Hypothesis. But first, please invest some effort into debugging things yourself, and providing a good bug report[1].

If your query involves code, create a complete but minimal working example of the issue. That is, a few lines of plain text code (no python notebooks please) that sets everything up and demonstrates the problem. I suggest running the function under question on a tiny toy example, perhaps with input arrays created with `randn`. Then the code will run quickly, and you can look at all the intermediate results at a console or in a debugger.

1. Simon Tatham (1999) gives good guidance: `http://www.chiark.greenend.org.uk/~sgtatham/bugs.html`

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[2]:
`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.

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 how your results demonstrate that these standard error bars do not reliably indicate what the average of locations in future CT slice data will be. Why are standard error bars misleading here?

      (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 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 are constants: they take on the same value for every training example. Identify these features, and remove them from the input matrices in the training, validation, and testing sets.

      Moreover, some of the input features are duplicates: some of the columns in the training set are identical. For each training set column, discard any later columns that are identical. Discard the same columns from the validation and testing sets.

2. 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.

Use these modified input arrays for the rest of the assignment.

Report which columns of the `X_...` arrays you remove at each of the two stages. 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.lstsq` 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. Use a data augmentation approach that maintains the numerical stability of the underlying \ or `lstsq` solver, rather than a 'normal equations' approach. 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 centred 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 centred 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.

   NB throughout I'm asking for the *square root* of the mean square error.

   In all future parts, please go back to using the original features, not the centred 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.concatenate([X, X==0, X<0], axis=1) # 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?

Now please go back to using the original features for the following questions, not the augmented versions that you created for this part.

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.

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 used earlier.

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 beyond the methods we have tried so far. Explain what you tried, why you thought it might work better, how you evaluated your idea, and what you found.

   Do not write more than a page for this part, including figures and results, but not including code. Sticking to the page limit will be a marking criterion. The bulk of the marks available for this part are for trying something sensible, which can be simple, and evaluating it sensibly.

   The constraints on allowable code given at the start of the assignment still apply. This question is about motivating simple ideas that you can implement from scratch.

   Your method does not have to work better to get some marks! However, you need to actually run something and report its results to get some marks. Also, your motivation should be sensible: don't pick an idea that your existing results already indicate are unlikely to work.

   I advise you not to spend a huge amount of time on this final part. If you are taking 120 credits of courses, this part is worth 0.7% of your mark average. Lots of extra effort might nudge your mark average by 0.2% or less.