MLPR Assignment 2 – Predicting CT Slice Locations

Due: 2pm Monday 22 November, 2021

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: `https://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 are encouraged to work in pairs on this assignment. Like assignment 1, it should be entirely possible to work by yourself. However, you may work with up to *one* other person. You can let us know if you would like to work by yourself, in a pair of your choice, or would like to be assigned to a random pair on this MS Form by 2pm Monday 25 Oct. By asking to work in a pair, you will be committing to submitting an assignment with them, for which you are jointly responsible.

IMPORTANT: Pairs must work jointly on the whole assignment. Like assignment 1, it's a sequence of questions that build on each other, and we hope you'll learn something by doing it all. Splitting up the work, so that parts are done by only one of you, is not acceptable.

Non-sharing policy: You may discuss your high level approach with other pairs or individuals, but you must *never* share code or written materials with other pairs or individuals. Don't post questions on this assignment using the class forum. Email your question to us instead. In response, we might post minor clarifications on the forum, or would email the class list with any major corrections.

Allowable code: You must use 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`, `cho_factor` and `cho_solve` functions imported by the support code and the `scipy.stats.norm.cdf` function.

Make sure to submit your answers as soon as you write them. 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 Rule 1 of the School Late Policy. As in the policy, please do not email Iain or Arno about extensions, but follow the instructions.

Queries: If you think there's a problem with the assignment or support code, email Iain and Arno 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. We 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.

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

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

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.

Download the data here (26 MB).

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. We 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: We provide support code along with this assignment, which you will need for the questions below. Do not copy this code into the answer boxes; your code should `import` it.

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

   As with each question, please report the few lines of code required to perform each requested step. When a question says to report a number, your code should print it, and manually include the resulting number as part of your answer.

   Load the data into Python (your code can assume this step has been done):

   ```
   import numpy as np
   data = np.load('ct_data.npz')
   X_train = data['X_train']; X_val = data['X_val']; X_test = data['X_test']
   y_train = data['y_train']; y_val = data['y_val']; y_test = data['y_test']
   ```

   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?

   [Visit question on website]

   (For the remainder of this assignment you don't need to report error bars on your results. If you were working on this application in the context of a research problem, you should 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.

   Use these modified input arrays for the rest of the assignment. Keep the names of the arrays the same (`X_train`, etc.), so we know what they're called. You should not duplicate the code from this part in future questions. We will assume it has been run, and that the modified data are available.

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.

Warning: As in the real world, mistakes at this stage would invalidate all of your results. We strongly recommend checking your code, for example on small test examples where you can see what it's doing.

Report which columns of the `X_...` arrays you remove at each of the two stages. Report these as 0-based indexes. (For the second stage, you might report indexes in the original array, or after you did the first stage. It doesn't matter, as long as your code is clear and correct.)

[Visit question on website]

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

   Using `numpy.linalg.lstsq`, write a short function "`fit_linreg(X, yy, alpha)`" that fits the linear regression model

   $$f(\mathbf{x}; \mathbf{w}, b) = \mathbf{w}^\top \mathbf{x} + b, \tag{1}$$

   by minimizing the cost function:

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

   with regularization constant α. As discussed in the lecture materials, 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 `lstsq` solver, rather than a 'normal equations' approach. You should only regularize the weights $\mathbf{w}$ and not the bias b.

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

   Use your function to fit weights and a bias to `X_train` and `y_train`. Use $\alpha = 30$.

   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 (RMSE) 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?

   [Visit question on website]

3. **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 20 positions within the range of training positions, and use each of these to define a classification task:

   ```
   K = 20 # 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 each of the 20 classification tasks above with $\alpha = 30$.

Given a feature vector, we can now obtain 20 different probabilities, the predictions of the 20 logistic regression models. Transform both the training and validation input matrices into new matrices with 20 columns, containing the probabilities from the 20 logistic regression models. 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 ($\alpha = 30$) to your transformed 20-dimensional training set. Report the training and validation root mean square errors (RMSE) of this model.

[Visit question on website]

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

   In Question 3 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 Q3.

   Does one initialization strategy work better than the other? Does fitting the neural network jointly work better than the procedure in Q3? Your explanation should include any numbers that your answer is based on.

   [Visit question on website]

5. **Bayesian optimisation [20 marks]:**

   A popular application area of Gaussian processes is Bayesian optimisation, where the uncertainty in the probabilistic model is used to guide the optimisation of a function. Here we will use Bayesian optimisation with Gaussian processes for choosing the regularisation parameter α. (We would normally use Bayesian optimisation when optimizing more than one parameter.)

   Gaussian processes are used to represent our belief about an unknown function. In this case, the function we are interested in is the neural network's validation log root mean square error (log RMSE) as a function of the regularisation paramter α. In Bayesian optimisation, it is common to maximise the unknown function, so we will maximise the negative log RMSE.

   We start with a Gaussian process prior over this function. As we observe the actual log RMSEs for particular α's we update our belief about the function by calculating the Gaussian process posterior.

   Besides the Gaussian process framework that you're already familiar with, Bayesian optimisation involves a so-called acquisition function. Given our Gaussian process posterior model, we use this function to decide which parameter to query next. The acquisition function describes how useful we think it will be to try a given α, while considering the uncertainty that is represented in our posterior belief.

   There are many popular acquisition functions in Bayesian optimisation. One example is the *probability of improvement*. Suppose we have observed $y^{(1)}$ to $y^{(N)}$ (here negative log RMSE at locations $\alpha^{(1)}$ to $\alpha^{(N)}$). Then the function takes the following form:

$$PI(\alpha) = \Phi\left(\frac{\mu(\alpha) - \max(y^{(1)}, \ldots, y^{N})}{\sigma(\alpha)}\right),$$

   where $\mu(\alpha)$ is the Gaussian process posterior mean at location α, $\sigma(\alpha)$ is the posterior standard deviation at location α, and Φ denotes the cumulative density function of the

Gaussian with mean 0 and variance 1.

We pick the next regularization constant $\alpha^{(N+1)}$ by maximizing the acquisition function:

$$\alpha^{(N+1)} = \arg\max_{\alpha} PI(\alpha).$$

We then evaluate our model for this regularization parameter and update our posterior about the unknown function that maps α to negative log RMSE. We repeat the procedure multiple times and then pick the parameter that yielded the best performance y.

Write a function `train_nn_reg` that trains the neural network from Q4 for a given α parameter and returns the validation RMSE.

Consider α on the range `np.arange(0, 50, 0.02)`. Pick three values from this set for α as training locations and use `train_nn_reg` on these locations. Use the remaining locations as values that you will consider with the acquisition function.

Take the performance of the neural network that you trained in Q4 a) as a baseline. Subtract your α-observed log RMSEs from the log of this baseline and take the resulting values as $y^{(1)}$ to $y^{(3)}$. Then calculate the Gaussian process posterior for these observations. To do so, use `gp_post_par` from the support code. Use the default parameters for `sigma_y`, `ell` and `sigma_f`.

Implement the probability of improvement acquisition function. You can use `scipy.stats.norm.cdf` to calculate Φ. With this acquisition function, iteratively pick new α's, re-train and evaluate each new model with `train_nn_reg` and update your posterior with `gp_post_par`. Do five of these iterations.

Report the best α that this procedure found, and its validation and test RMSEs. Have we improved the model?

[Visit question on website]

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 300 words for this part, not including your code (which you do need to include). The bulk of the marks available for this part are for motivating and 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.

   We expect that most students will get less than 10 marks on this question: some of the tasks in this assignment are straightforward and the overall mark on the common marking scheme should reserve marks in the 80s and 90s for truly outstanding work. However, we 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 0.8% of your mark average. Lots of extra effort might nudge your mark average by 0.2% or less.

   [Visit question on website]