Week 8 exercises

This is the fifth page of *assessed* questions, as described in the background notes. These questions form 70% of your mark for Week 8. The introductory questions in the notes and the Week 8 discussion group task form the remaining 30% of your mark for Week 8.

Unlike the questions in the notes, you'll not immediately see any example answers on this page. However, you can edit and resubmit your answers as many times as you like until the deadline (Friday 13 November 4pm UK time, UTC). This is a *hard deadline*: This course does not permit extensions and any work submitted after the deadline will receive a mark of zero. See the late work policy.

Queries: Please don't discuss/query the assessed questions on hypothesis until after the deadline. If you think there is a mistake in a question this week, please email Arno and Iain.

Please only answer what's asked. Markers will reward succinct to-the-point answers. You can put any other observations in the "Add any extra notes" button (but this is for your record, or to point out things that seemed strange, not to get extra credit). Some questions ask for discussion, and so are open-ended, and probably have no perfect answer. For these, stay within the stated word limits, and limit the amount of time you spend on them (they are a small part of your final mark).

Feedback: We'll return feedback on your submission via email by Friday 20 November.

Good Scholarly Practice: Please remember the University requirements for all assessed work for credit. 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 may not publish your solutions after the deadline either.

1 Combining linear and logistic regression

In this question 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]. However, you should use our pre-processed version of the dataset:

Download the data (26 MB).

Download the support code.

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.*

We are going to try out using a gradient-based optimizer to fit some models. Our dataset is not very large, so "batch" optimizers like L-BFGS provided by SciPy work well. If we used a variant of stochastic gradient descent, it would require tuning, and you have limited time.

The support code contains a wrapper of SciPy's routine called `minimize_list`, that can optimize a Python list of parameters given a cost function and its gradients. This setup will

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.

seem overly-complicated for the simple models we consider here. But next week will let us fit a more complicated model with minimal extra effort.

a) **Linear regression with a gradient method** [20 marks]

`linreg_cost` in the support code implements the square cost function for simple linear regression with regularization applied to the weights, but not the bias. The regularization constant is called `alpha`, because `lambda` (which we used in the week 1 notes) is a reserved word in Python.

`fit_linreg_gradopt` in the support code demonstrates how to fit weights and a bias to this cost function, using the provided SciPy wrapper.

In the answer box below:

i) Report the root-mean-square errors on the training and validation sets for the parameters fitted by `fit_linreg_gradopt`, using `alpha=10`.

ii) Then explain anything you did to check whether your results seem right.

iii) Then include your code between lines containing three backtics "'. Your code should run as provided, assuming the data file is in the current directory, and should print out the two results. Only use NumPy and the support code (which uses SciPy). Please don't duplicate the support code, just include an import line.

[The website version of this note has a question here.]

b) **Invented classification tasks.** [20 marks]

Binary classification is in some ways simpler than real-valued regression. We don't have to think so hard about how the target values might be distributed, and how we might process them. Here we invent some binary classification tasks based on our data, and fit these. We then use the results as features in an attempt to improve our regression model.

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

```
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 in the support code's function `logreg_cost`. It is analogous to the `linreg_cost` function for least-squares regression, which was 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 an input 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 (as probabilities) from 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. After that,

include your code. Again, it should run as provided (with the data in the current directory), must only import NumPy and the support code, and should print out the two results. Duplicate any code from the previous part that's needed for your demonstration to run.

[The website version of this note has a question here.]

2 Difference observations

Sometimes we can only measure the difference in a function at two different locations. We might display two panels on a webpage with features $\mathbf{x}^{(a)}$ and $\mathbf{x}^{(b)}$ and measure how many times the first one was selected instead of the second. In this example, we're not directly measuring how good the first panel was, but how much better it is than the second one.

A simplistic model could be that a function measures how good each panel is, with scalar values $f(\mathbf{x}^{(a)})$ and $f(\mathbf{x}^{(b)})$, then an observation y_n measures the difference in the function evaluated at $\mathbf{x}^{(n,a)}$ and $\mathbf{x}^{(n,b)}$:

$$y_n \sim \mathcal{N}(f(\mathbf{x}^{(n,a)}) - f(\mathbf{x}^{(n,b)}), \sigma_y^2). \tag{1}$$

For linear regression with basis functions:

$$f(\mathbf{x}^{(a)}) - f(\mathbf{x}^{(b)}) = \mathbf{w}^\top (\boldsymbol{\phi}(\mathbf{x}^{(a)}) - \boldsymbol{\phi}(\mathbf{x}^{(b)})). \tag{2}$$

We decide to do Bayesian linear regression with prior over weights $p(\mathbf{w}) = \mathcal{N}(\mathbf{w}; \mathbf{0}, V_0)$.

a) If the set of basis functions includes a constant, $\phi_K(\mathbf{x}) = 1$, what is the posterior distribution over w_K in this context, and give a brief interpretation. Write no more than 2 sentences. [10 marks]

[The website version of this note has a question here.]

We now decide to modify our model of the difference observations to use a Gaussian process prior (with kernel k) over the underlying function $f(\mathbf{x})$. We need to modify the approach in the notes, to use the correct covariances for this setting.

Hint: You can find the covariances by interpreting the kernel as an inner product between feature vectors, or by reasoning about covariances or expectations directly.

b) Write down the covariance between two observations y_n and y_m. [10 marks]

[The website version of this note has a question here.]

c) Write down the covariance between an observations y_n and the function evaluated at a test point, $f_{*,j} = f(\mathbf{x}^{(*,j)})$. Briefly describe how you arrived at your result. [10 marks]

[The website version of this note has a question here.]