CS342 Machine Learning: Lab #2

Labs on Week 2 of Term 2

Week 16

Instructor:
Dr Theo Damoulas (T.Damoulas@warwick.ac.uk)

Tutors:
Helen McKay (H.McKay@warwick.ac.uk),
Joe Meagher (J.Meagher@warwick.ac.uk)
Karla Monterrubio-Gomez (K.Monterrubio-Gomez@warwick.ac.uk),
Jevgenij Gamper (J.Gamper@warwick.ac.uk)

In the second Lab we will work on linear regression and the ordinary least squares (OLS) solution.

Linear regression with OLS

The material here builds on the first three lectures. As with the lectures, the lab follows Chapter 1 from the Rogers & Girolami book. In general, you can find the data and Matlab codes for these exercises in the R&G book: `https://www.dropbox.com/sh/7p6tu1t29idgliq/AAC3ipz88K?m` but at this stage you should start to implement them on your own and in Python. We will be providing the Python (unoptimised) "solutions" a week after each Lab.

Feature scaling

→ Import the men's 100m Olympic dataset into a Pandas data structure. Calculate the mean and the standard deviation of each column. Subtract the mean value of each column from every element of the column. Then divide every element of the column with the standard deviation of each column.

This is a typical pre-processing step called "standardisation" that rescales our data in order to avoid problems of relative scale (e.g. when one attribute takes values in [10,1000] whereas another in [0.01, 0.2]) . There are other such **feature scaling** steps that one can use, such as to scale the N D-dimensional inputs to unit length: $\mathbf{x}_n = \frac{\mathbf{x}_n}{||\mathbf{x}_n||_p}$ where $||\mathbf{x}_n||_p$ is the L_p norm of input $\mathbf{x}_n$. This is defined as $L_p = \left(\sum_{d=1}^{D} |x_{nd}|^p\right)^{\frac{1}{p}}$ and for p=1 it gives us the Manhattan norm: $L_1 = \sum_{d=1}^{D} |x_{nd}|$. Norms can be thought of, for our purposes, as different distances of the vectors $\mathbf{x}_n$ from the centre of the Cartesian axis.

→ What does it mean if the std of an attribute/feature is zero? What should we do if that happens?

→ Create a function called *scaleData* that takes as input our input data $\mathbf{X}$ and an option/flag second argument that defines which scaling the function implements. Implement both scaling techniques from above.

Linear regression example with Ordinary Least Squares (OLS)

→ Write a function called *linreg* that takes as inputs the training data $\mathbf{X}, \mathbf{t}$ and returns $\widehat{\mathbf{w}}$ the OLS estimate of the parameters. The OLS solution, see lecture notes or the R&G book chapter, is:

$$\widehat{w_0} = \bar{t} - w_1 \bar{x} = \frac{1}{N}\sum_{n=1}^{N} t_n - w_1 \sum_{n=1}^{N} x_n \tag{1}$$

$$\widehat{w_1} = \frac{\overline{xt} - \bar{x}\bar{t}}{\overline{x^2} - (\bar{x})^2} = \frac{\frac{1}{N}\left(\sum_{n=1}^{N} x_n t_n\right) - \bar{x}\bar{t}}{\left(\frac{1}{N}\sum_{n=1}^{N} x_n^2\right) - (\bar{x})^2} \tag{2}$$

An equivalent and easier implementation of the above solution is to use the following vector/matrix representation (pages 15 - 22 in R&G book) that makes use of the transpose $(\cdot)^{\mathrm{T}}$ and inverse $(\cdot)^{-1}$ operators:

$$\widehat{\mathbf{w}} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{t} \tag{3}$$

note that $\mathbf{X}$ here includes a column of 1s to account for the intercept term w_0. i.e. a row of this matrix $\mathbf{X}$ is $\mathbf{x}_n = [1\ x_{n1}\ \ldots\ x_{nD}]$.

To implement this, Pandas data frames can easily be converted into matrices using *dataFrame.as_matrix(columns)*. Once converted, matrix operations in the *numpy* library can be used such as *matrix.transpose()*.

→ Now apply this function on your *standardized* data $\mathbf{X}$, and targets $\mathbf{t}$ from the Olympic dataset. Plot your dataset and the OLS hypothesis (the fitted line).

→ Use the built-in linear regression (OLS) package available in *scipy.stats* and compare the solutions. To call the built-in function import the library using: *from scipy import stats* and execute using: *stats.linregress(x,y)*.

→ Write a function called *predict* that takes as input your inferred OLS parameters and predicts the output t^* for any new input $\mathbf{x}^*$. You will have to apply the same standardisation to $\mathbf{x}^*$ as you did for $\mathbf{X}$. Predict the winning time for 2025. What do you observe? What would you do if you were asked to give a realistic prediction?

Non-linear response from a linear model

→ Extend your *linreg* function to take as input a parameter that dictates the order of the polynomial feature expansion (see lecture notes and chapter). Repeat 1.2 but this time with a 3rd order polynomial expansion of the data. Your matrix $\mathbf{X}$ should have N rows and D=4 columns with the nth row as $[1\ x_n\ x_n^2\ x_n^3]$ and your $\mathbf{w}$ vector will be a 4-dimensional vector $[w_o\ w_1\ w_2\ w_3]$.

K-fold Cross-validation

Implement K-fold Cross Validation and use it, together with a performance metric such as R^2 or RMSE, to select the best model between the 1st order and 3rd order polynomials you have fitted to the data so far in previous sections. You may find the *cross_validation* package from the *sklearn* library helpful for creating folds (*from sklearn import cross_validation as cv*), particularly the *KFold* function.

Learning curve

Split one of the Olympic datasets into two sets, a test set and the remaining data. Keep the test set fixed and train multiple models from the remaining data while reducing the size of the training set each time. Predict with each resulting model on the test set, record R^2 and/or RMSE and construct a figure showing the "learning curve": How performance of your learning algorithm on a fixed test set improves as you increase the amount of training data (up to a point).

Woman's vs Man's Olympic times

Perform linear regression with no feature expansion on the men's and women's 100 m. Using these models find the Olympic games when it is predicted for women to run a faster winning time than men. What are the predicted winning times? Do they seem realistic?