Linear Regression Tutorial
COMS3007

Benjamin Rosman, Devon Jarvis

April 12, 2021

Due date: Tuesday, 20 March at 2.00pm.

Instructions: Use your notes and any resources you find online to answer the following questions. You need to submit a PDF of your answers to Question 4 on Moodle. This can be done in groups of *up to four*. Make sure all your names and student numbers appear on the document!

1. The regression problem involves making predictions of a continuous variable. This is done by learning a model of the relationship between the input variables (features) and the output variable (target). This learning is done from labelled training data.

   Linear regression then refers to a particular choice of model, which is *linear* in its parameters θ:

$$y = f(\mathbf{x}, \theta) = \sum_{j=0}^{d} \theta_j x_j \tag{1}$$

   where $\mathbf{x} = \{x_1, x_2, ..., x_d\}$ is a d-dimensional set of features which we augment with $x_0 = 1$, and for each feature x_j there is a corresponding weight (parameter) θ_j that needs to be learned.

   Consider the training data in Table 1. This case has a single feature x, and 5 training data points.

   Table 1: Training dataset

x	1	2	3	4	5
y	1	3	2	3	5

   Here we may choose to fit a simple model (a straight line) of the form:

$$\hat{y} = \sum_{j=0}^{1} \theta_j x_j = \theta_0 + \theta_1 x \tag{2}$$

   Obtaining a best fit to the data may give that $\theta_0 = 0.4$ and $\theta_1 = 0.8$, such that $\hat{y} = 0.4 + 0.8x$. This is shown in Figure 1, where the blue points are the training data, and the red line is the model. The predicted values are shown as red points.

   We can determine the quality of the fit by the sum of squares error function, which measures the difference, for each point x in the training data, between the true value y and the predicted value $\hat{y} = f(\mathbf{x}, \theta)$. The squared error of a single point is $(y - \hat{y})^2$, and for N data points, the sum of squares error is:

$$E(\theta) = \frac{1}{2} \sum_{n=1}^{N} (\hat{y} - y)^2 \tag{3}$$

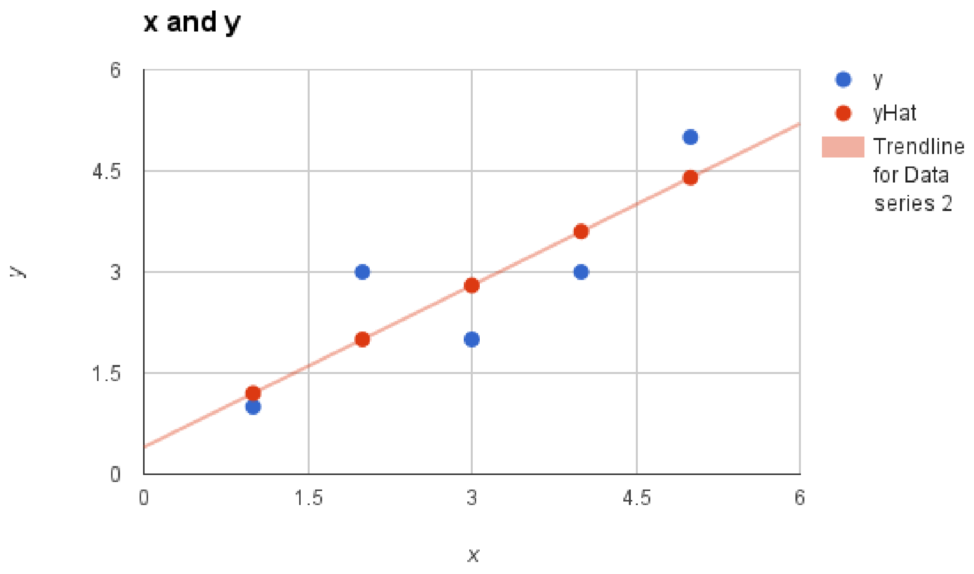


Figure 1: A simple example

(a) Using the regresssion model given above, compute the predicted value $\hat{y}(x)$ for $x = 1$. What is the error at $x = 1$?

(b) Compute the predicted value $\hat{y}(x)$ for $x = 2$. What is the error at $x = 2$?

(c) Repeat this process for the remaining training points. What is the sum of squares error for this model?

(d) Now consider the model $y = x$. What is the error for this model?

(e) What is the error for the model $y = 1 + 2x$?

(f) With a quick sketch, convince yourself which of these three models is the best fit to the data, and compare this to the error values.

(g) Given the best model, make a prediction for $\hat{y}(6)$.

2. One way of learning the parameters θ is to use a closed form solution. To do this, we first write down the design matrix. This is the matrix X of all the features and all the training data, such that the i^{th} row represents data point i, and the j^{th} column is feature j. By convention, the first column is all 1s.

   This allows us to express our training data in equation (2) as $\mathbf{y} = X\theta$.

   Now, we should just be able to solve for $\theta = X^{-1}\mathbf{y}$, but we cannot do this, as X is not in general a square matrix, and because $\hat{y}$ is not exactly equal to y. We instead solve for θ using:

$$\theta = (X^T X)^{-1} X^T y \tag{4}$$

   Note: the derivation of this comes from setting the derivative of the error function to 0. This implies finding the best fit, and not the exact solution.

   (a) Write down the design matrix X for the problem in the previous question, using the model in equation (2).

   (b) Compute the parameters of best fit for θ_0 and θ_1 for this problem using equation (4).

3. Although this is *linear* regression, the features themselves do not need to be linear functions, we just need to have the model be linear in the parameters θ.

   Consider the training data in Table 2.

   Table 2: Training dataset

x	1	2	3	4	5
y	1	5	9	15	25

   (a) Using the model in equation (2), compute the parameters of best fit for θ_0 and θ_1 for this problem using equation (4). Remember to create the design matrix X.

   (b) Compute the training error of this fit.

   (c) Now we wish to add a new feature which may help fit the data better: x^2. We will now try fit the model $y = \theta_0 + \theta_1 x + \theta_2 x^2$. Write out the design matrix X corresponding to this model.

   (d) Compute the parameters of best fit θ given the design matrix in (c) above, and equation (4).

   (e) What is the training error of this fit?

   (f) Given a validation point $(x, y) = (10, 104.5)$, compare the error of this point given the model with features $\{1, x\}$ from (a), and the model with features $\{1, x, x^2\}$ from (d).

   (g) Now repeat the process with the feature set $\{1, x, x^2, x^3, x^4, x^5\}$. Compute the training error. Then compute the validation error, given the point in (f). What do you notice?

   (h) Given the results in (a), (d), and (g) above, which model would you consider to be overfitting and which to be underfitting. Why? Plot the curves given by the three models, with the training data.

4. As discussed in lectures, being able to sample/generate our own data is a useful skill which allows us to experiment with our algorithms on controlled data. Thus, for this week's lab we will be generating our own data to work with.

   (a) Lets begin by obtaining the data.

      i. Sample 150 x-values from a Normal distribution using a mean of 0 and standard deviation of 10. Hint: np.random.normal

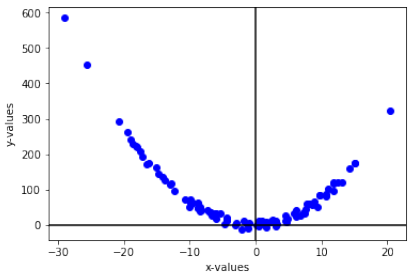
      ii. From the x-values construct a design matrix using the features $\{1,x,x^2\}$.

      iii. Use a uniform distribution to sample **true** values for θ_0, θ_1 and θ_2.
      NOTE: when dealing with real-world data we will not have these values. These are the parameter values we want our algorithm to learn. We have to define them here because we are creating a function to create the data.

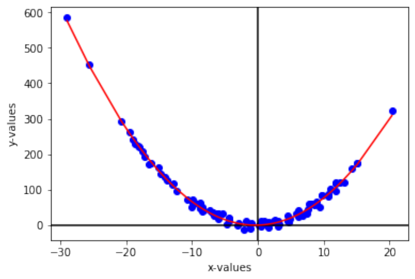
      iv. Use your design matrix and the true parameters you obtained to create the y-values for the regression data. Finally add random noise to the y-values using a Normal distribution with mean 0 and standard deviation of 8.

      v. Plot the x-values and their corresponding y-values on a 2D-axis. Your data should look similar to the data shown in Figure 2a. Hint: pyplot.scatter

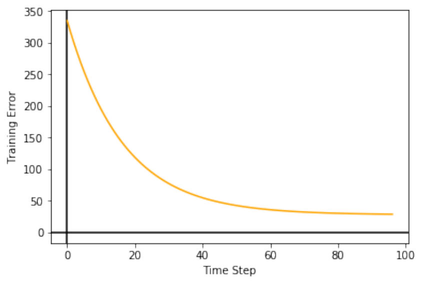
      vi. Split the data into training, validation and test datasets.



(a) Example Data



(b) Example Regression



(c) Example Error over Time

Figure 2: Example Plots

(b) Now that we have data we can train our models.

i. Use the Moore-Penrose pseudo-inverse to calculate the closed form solution for the model's parameter values.

ii. How close are the learned parameter values to the true parameter values we used to generate the data?

iii. Compute the training error and validation error for the learned regression model.

iv. Create a scatter plot of the individual data points along with the learned regression function, your plot should look like Figure 2b. Hint: pyplot.plot, this plotting function will give weird results if the x-values of the data are not sorted. x_train[x_train[:,1].argsort()] will give you the design matrix for your training data sorted by the second column (where the x values should be).

v. Repeat the above process using Gradient Descent to train your model. In addition, plot the training error of your regression model over time (observe or capture the training error every 20 parameter updates/time steps). Your plot should look like Figure 2c.

(c) We will now experiment with overfitting and regularization.

i. Begin by appending a third feature to your design matrix for x^3.

ii. Train a model using Gradient Descent with the new design matrix. Repeat the process used above in Question 4b. Note, we are now using a third-order polynomial to fit data which was generated using a second-order polynomial. Our function is, thus, more complicated than is necessary to fit the data and as a result will overfit.

iii. Repeat the training process one final time, this time use regularization when training the third-order polynomial model.

iv. Compare your results of the three gradient descent based models, which model achives the best final training error? Which model trains the fastest? Which model achieves the best validation error? Can you see any visible difference in the function approximation (fit of the data) by the models or in the learned parameter values? Did any of these models achieve a lower training or validation error than the closed form solution?

v. Compute the final test error for all four of your models. Which model obtains the lowest test error? Did the regularisation improve the test error performance of the third-order polynomial model? Which of the three gradient descent models achieved the lowest test error? Would you say it is better to use higher-order polynomials and regularize or use a model which only uses the necessary features to fit the data.