Exercises # 06

Introduction to Image Analysis and Machine Learning
(Spring 2017)

March 28, 2017

Purpose: The purpose of these exercises is to work with linear supervised regression and classification models. The main focus will be to get a better understanding of data fitting and its relation to regression and classification. We will learn the behavior of the data by approximating the coefficients of the known model using training data. We will use the trained model to make predictions in the test data. We will work with fine tuning parameters to get improved results out of the learned model.
Please download exercise resources from learnIT before starting the exercises.

Linear Regression

In this exercise you will work with Linear Regression.

Learning Goal: After completing these exercises you should be able to:

- Estimate statistical quantities from data.
- Estimate linear regression coefficients from training data.
- Make predictions using linear regression for new data.

Description

Linear Regression assumes a linear relationship between input variables (X) and single output variable (y). Specifically, that output (y) is calculated from a linear combination of the input variables (X). When there is a single input variable, the method is referred to as a simple linear regression.
In 2D the regression model is:

$$y = b_1 x_1 + b_0 \tag{1}$$

Where b_0 and b_1 are the coefficients we must estimate from the training data. Once the coefficients are known, we can use this equation to estimate output values for y given new input examples of x. For n-dimensions this relation can be written as:

$$y = \sum_{i=1}^{n} b_i x_i + b_0 \tag{2}$$

This relation can be represented as a set of linear equations as follows:

$$Ax = b \tag{3}$$

Where x is the coefficients to be estimated. These coefficients can be estimated using least squares.

Root Mean Squared Error

The Root Mean Squared Error (RMSE) is used to estimate the error in predicted outputs. RMSE is calculated as the square root of the mean of the squared differences between actual outcomes and predictions. [1]

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(predicted_i - actual_i)^2}{total\ predictions}}$$

[1] For further details about the expressions and code used in this exercise session, you may see SimpleLinearRegression.pdf in reading material. Or visit the link: Simple linear regression tutorial.

Exercise: 2.1.1

This exercise uses a contrived dataset which consists of five data points. The first value in each instance is input variable (x) and second value is output variable (y). We use this contrived data set for training the simple regression model i.e. estimating the values of the coefficients based on least squares. Once our model is trained, we omit the *output variable* in the contrived dataset and predict it using trained model (b_0 and b_1). The following steps elaborate simple linear regression implementation on contrived dataset.
Following steps are performed to implement linear regression.

1. In your exercise material execute `simple_linear_regression.py`. The output of the script, that you will see in output console, is an array consisting of predicted outputs for the contrived dataset. It also gives RMSE in predicted outputs. Figure 1 shows the actual and predicted outputs.

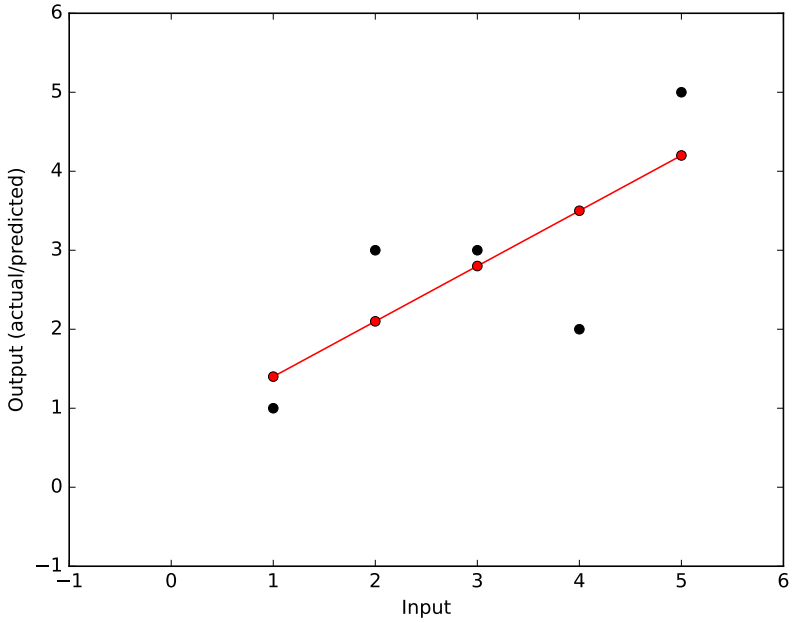

Figure 1: Actual and Predicted Data

2. **Calculate Mean and Variance:** The first step is to estimate the mean and the variance of both the input and output variables from the training data. The functions `mean()` and `variance()` implement these functionalities.

3. **Calculate Covariance:** The covariance of two groups of numbers describes how those numbers change together.The function named `covariance()` that implements this statistic. It builds upon the previous step and takes the lists of x and y values as well as the mean of these values as arguments.

4. **Estimate Coefficients:** Now we have all the required statistics for the estimation of coefficients in simple linear regression. The expressions needed for coefficient estimation

are already described in the introductory part of the exercise. In our script, the function `coefficients()` implements these details.

5. **Make Predictions:** Since we know that simple linear regression model is a line defined by coefficients estimated from training data. Once the coefficients are estimated, we can use them to make predictions. In our script, `simple_linear_regression()` implements the prediction logic on test data. In this particular example our training data is also used as test data.

6. **Evaluate Predictions:** We use RMSE metric to evaluate the predictions. The function `evaluate algorithm()` takes care of this step along with `rmse_metric()` to estimate the error in predicted outputs. The expression related to RMSE is already given in the beginning of this exercise.

You will apply linear regression to the Swedish insurance dataset. The dataset file `insurance.csv` is already available in exercise resources. For the following tasks you will use `simple_linear_regression_insurance.py` which is already present in the assignment resources. Some convenience functions are added to the simple linear regression from the previous steps. Specifically a function to load the `CSV` file called `load_csv()`, a function to convert a loaded dataset to numbers called `str_column_to_float()`, a function to evaluate an algorithm using a train and test set called `train_test_split()` and a function to plot the actual and approximated model results is called `plot_outputs()`. In current configuration of the script, a training dataset of 60% of the given dataset is used to prepare the model (training) and predictions are made on the remaining 40%.

Tasks

1. Implement function `evaluate_algorithm()` that trains the linear regression model and then make prediction on test dataset. As a hint you may perform the following steps: (Difficulty level: Medium, Estimated time: 40min)

   - Split given dataset into training and test datasets. You may change the proportions of the train and test datasets by setting the value of `split` variable. Use `train_test_split()` function to perform this task.
   - Approximate model coefficients. Use `simple_linear_regression()` function to do this task.
   - Make predictions based on the approximated model. Once you have approximated values for the coefficients, use linear model to make predictions in `simple_linear_regression()`.
   - Calculate RMSE at the fitted model. Use `rmse_metric()` function to accomplish this task.
   - Return appropriate data to plot actual and predicted outputs.

2. Plot the actual and predicted output. You can do that by providing appropriate parameters to the function `plot_output()`. (Difficulty level: Low, Estimated time: 10min)

3. Evaluate algorithm with different train and test proportions. You can do it by changing the value of the *split* parameter. By default it is set to 0.6 which means that given dataset is split into 60% training and 40% test sets. Monitor RMSE value while you run your algorithm with different train and test proportions. Which proportions give you best results and why? (Difficulty level: low, Estimated time: 20min)

Swedish Auto Insurance Dataset

The Swedish Auto Insurance Dataset involves predicting the total payment for all claims in thousands of Swedish Kronor, given the total number of claims. It is a regression problem. It is comprised of 63 observations with 1 input variable and 1 output variable. The variable names are as follows:

1. Number of claims.
2. Total payment for all claims in thousands of Swedish Kronor.

The baseline performance of predicting the mean value is an RMSE of approximately 72.251 thousand Kronor.

Logistic Regression

Logistic regression is a linear algorithm for binary classification problems. It is easy to understand and produces sufficiently good results for a variety of problems.

Learning Goal: After completing these exercises you should be able to:

- Make binary-classification using logistic regression model.
- Estimate coefficients using stochastic gradient descent.
- Apply logistic regression to a real binary classification problem.

Description

The main expression that logistic regression uses is as follows:

$$y = \frac{1}{1 + \mathrm{e}^{-(b_0 + b_1 \times x)}} \tag{4}$$

Where e is the base of the natural logarithms (Eulers number), y is the predicted output, b_0 is the bias or intercept term and b_1 is the coefficient for the single input value x_1. The y prediction is a real value between 0 and 1 that needs to be rounded to an integer value and mapped to a predicted class value. Each column in the input data has an associated b coefficient (a constant real value) that must be learned from the training data. The actual representation of the model that you would store in memory or in a file is the coefficients in the equation (the beta value or b's). The coefficients of the logistic regression algorithm must be estimated from the training data. Logistic Regression uses gradient descent to update the coefficients. In each gradient descent iteration, the coefficients related to every column (input variable) in the training dataset are updated.
In the following discussion and examples we will use the Diabetes Dataset. This dataset involves the prediction of the onset of diabetes within 5 years. The baseline performance on the dataset is approximately 65%. In order to understand the process, classification of contrived dataset is divided into the following steps:

1. **Making Predictions:** The first step is to develop a function that makes predictions. This will be needed both in the evaluation of candidate coefficient values in stochastic gradient descent and after the model is finalized and we wish to start making predictions on test data or new data. Open file `logistic_regression_contrived.py` and read function named `predict()` which predicts an output value for a row given a set of coefficients. A row or observation consists of a set of input variables and a class label e.g. [2.7810836,2.550537003,0] is a row where first two values are input variables and they belong to the class 0. Also see a contrived small dataset in the same file. Figure 2 is the plot of the same dataset using different colors to show the different classes for each point.

2. **Estimating Coefficients** We can estimate the coefficient values for our training data using stochastic gradient descent. Stochastic gradient descent requires two parameters:

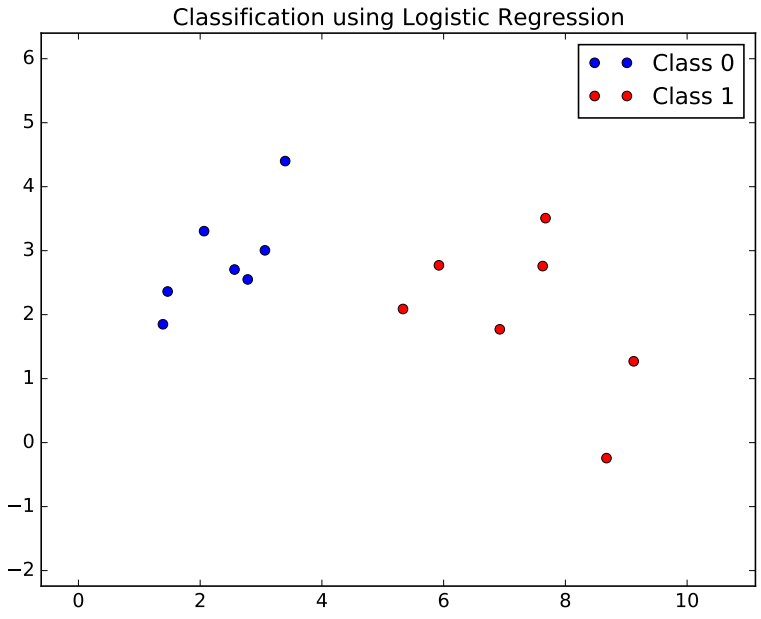

Figure 2: Classification using Logistic Regression

- **Learning Rate:** Used to limit the amount each coefficient is corrected each time it is updated.
- **Epochs/Iterations:** The number of times to run through the training data while updating the coefficients.

These, along with the training data will be the arguments to the function. There are 3 loops we need to perform in the function:

```
Loop over each epoch.
    Loop over each row in the training data for an epoch.
        Loop over each coefficient and update it for a row in an epoch.
```

Each coefficient is updated for each row in the training data, for each `epoch`. Coefficients are updated based on the error the model made. The error is calculated as the difference between the expected output value and the prediction made with the candidate coefficients. There is one coefficient to weight each input attribute. In a simple case where we have only one input variable, the following relation is used to estimate the coefficients iteratively. During each iteration, the coefficients (b) updated using the equation:

$$b(t+1) = b(t) + \alpha \times error \times x(t) \tag{5}$$

Where b is the coefficient being optimized, α is the learning rate that you must configure (e.g. 0.01), *error* is the prediction error for the model on the training data (also called weight), and x is the input value. Read 3rd nested loop in `coefficients_sgd()` for

our own implementation. The coefficients are updated based on the *error* the model made. The error is calculated as the difference between the expected output value and the prediction made with the candidate coefficients. There is one coefficient to weight each input attribute.

In this example, we use a larger learning rate of 0.3 and train the model for 100 epochs, or 100 exposures of the coefficients to the entire training dataset. Running the example prints a message each epoch with the sum squared error for that epoch and the final set of coefficients. You may have noticed that RMSE continues to drop in subsequent *epochs*. We could probably train for a lot longer (more epochs) or increase the amount we update the coefficients each epoch (higher learning rate). Experiment and see what you come up with.

3. **Application to the real dataset:** In this section, we will train a logistic regression model using stochastic gradient descent on the diabetes dataset. The dataset file named `diabetes-dataset.csv` is already provided in the exercise resources. Similar to the previous exercise function load_csv() loads the dataset.

   Each column is normalized to values in the range of 0 to 1. This is a usual practice in machine learning domain that in case of some algorithms or nature of data, the normalized data gives better results. The normalization is achieved through function named `normalize_dataset()`.

   In previous exercise we used `split_test_train()` in order to split given data into training and test datasets. There are several way to make this train test split. In this exercise, we will use another method for train test data split that is *k-fold cross-validation*. Read the function `cross_validation_split()` to know how it actually works. The function `evaluate_algorithm()` works the same way as it was explained in the previous exercise. The functions `predict()`, `coefficients_sgd()` and `logistic_regression()` are used to train the model.

Tasks

1. In order visualize the classification, plot predicted test data. For help, look at the example program. (Difficulty level: low, Estimated time: 30min)

2. Change the learning rate and plot the test data to get an improved RMSE score on the dataset. (Difficulty level: Low, Estimated time: 20min)

3. Change the number of epochs/iterations and plot the test data to get an improved RMSE score on the dataset. (Difficulty level: Low, Estimated time: 20min)

4. Evaluate algorithm with different train and test proportions. Monitor RMSE value while you run your algorithm with different train and test proportions. Which proportions give you best results and why? (Difficulty level: low, Estimated time: 20min)

Diabetes Dataset

The Diabetes Dataset involves predicting the onset of diabetes within 5 years in Pima Indians given medical details. It is a binary (2-class) classification problem. The number of observations for each class is not balanced. There are 768 observations with 8 input variables and 1 output variable. Missing values are believed to be encoded with 0 values. The variable names are as follows:

1. Number of times pregnant.
2. Plasma glucose concentration a 2 hours in an oral glucose tolerance test.
3. Diastolic blood pressure (mm Hg).
4. Triceps skinfold thickness (mm).
5. 2-Hour serum insulin (mu U/ml).
6. Body mass index (weight in kg/(height in m)2).
7. Diabetes pedigree function.
8. Age (years).
9. Class variable (0 or 1).

The baseline performance of predicting the most prevalent class is a classification accuracy of approximately 65%. Top results achieve a classification accuracy of approximately 77%.