Least Squares For Classification

***Abstract*—This document contains theory involved in curve fitting.**

1 Objective

The objective is to implement the Least Squares for classification of a data set.

2 Polynomial Curve Fitting

The goal is to find the best line that fits into the pattern of the training data shown in the graph. We shall fit the data using a polynomial function of the form,

$$y(w, x) = \sum_{j=0}^{M} w_j x^j \tag{2.0.1}$$

$$\tag{2.0.2}$$

M is the order of the polynomial The polynomial coefficient are collectively denoted by the vector **w**.The proposed vector **w** of the model referring to Eq (**??**) is given by

$$\hat{\mathbf{w}} = \left(\mathbf{F}^T\mathbf{F}\right)^{-1}\mathbf{F}^T y \tag{2.0.3}$$

3 Least Squares for Classification

Among the simplest statistical models of supervised learning is that of least squares linear regression.When applied to the task of classification, the model seeks to find a line, (or, higher-dimensionally, a hyperplane) which separates input space into two distinct regions, one for each class. If such a division of space exists, we say that the two classes are linearly separable. Given a random variable $X = (X1, X2, \ldots, Xn)^T$, a weight vector $\beta = (\beta 1, \beta 2, \ldots,)^T$, and a bias β_0 we discriminate the class to which X belongs according to the output of the following function

$$\delta(X) = 1, X^T\beta + \beta_0 \geq 0 \tag{3.0.1}$$

$$\delta(X) = -1, X^T\beta + \beta_0 < 0 \tag{3.0.2}$$

where the outputs of +1 and -1 indicate X 's membership in either of class one or class two, respectively. Such a function is clearly deterministic. In this tutorial, we will demonstrate how an optimal decision boundary may be calculated given a set of suitable training data.

4 Generating Dataset

We are given a set of binary classification data in two files, data/x.txt and data/y.txt. The first file contains the data points themselves (i.e. their locations in 2D coordinate space), and the second contains for each data point the class to which it belongs. As it stands, class one has one more data point associated with it than class two, so we pick a point from class one and discard it

```
# Read in data
x_vals = np.loadtxt('/content/gdrive/My Drive/
    data/x.txt')
y_vals = np.loadtxt('/content/gdrive/My Drive/
    data/y.txt')

# Throw out one data point so that there
# are an equal number from each class.
class_one = x_vals[:49, :]
class_two = x_vals[50:, :]

# Create data array for plotting
data = np.hstack((class_one, class_two))
```

5 Visualizing Data set

Here we create a Pandas DataFrame to hold our data and then create a scatter plot of it. We identify the class to which each point belongs by coloring it accordingly. Here we use orange for class one and light blue for class two.

```
# Create Pandas DataFrame for holding binary
    class data.
df = pd.DataFrame(data, columns=['x', 'y', 'x1', '
    x2'])

# Create scatter plot of data points in both classes.
class_ax = df.plot.scatter(x='x', y='y', color='
    Orange', label='+1');
```

```
df.plot.scatter(x='x1', y='x2', color='LightBlue',
    label='-1', ax=class_ax);
```

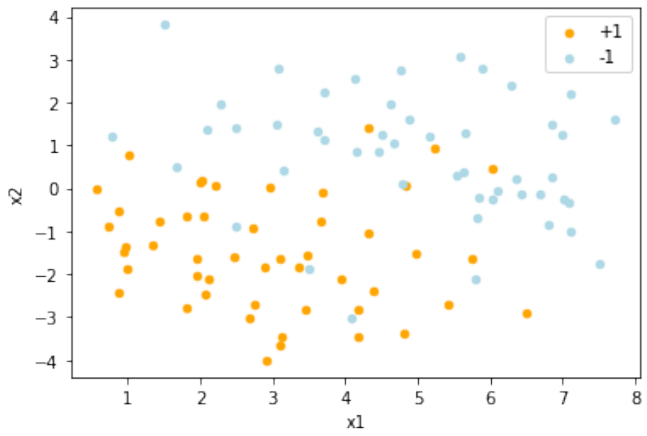

Fig. 0

6 Implementation of Least Squares for classification

Now that we've properly prepared our data for analysis, we can calculate our linear decision boundary using the method of least squares. We will present a general solution for this problem, in n -dimensional space. Given an n -dimensional input, $X^T = (X1, X2, \ldots, Xn)$, our decision boundary Y has the form

$$Y = \beta o + \sum_{i=1}^{n} = X_i \beta_i \tag{6.0.1}$$

The term βo is called the bias of our separating hyperplane. If we incorporate a zeroth column of 1's into X we can likewise include β_0 in the complete coefficient vector β and write our model in the more compact form

$Y = X^T \beta$

In this form, it is clear that $Y' = \beta$ is a gradient vector which completely determines the equation of the line giving our decision boundary.

In order to fit our model to the training data at hand, we use the method of least squares. This method requires that we choose gradient vector β in such a way so as to minimize a quantity called the residual sum of squares, defined as

$$RSS(\beta) = \sum_{i=1}^{N} (y_i - X_i^T \beta_i)^2 \tag{6.0.2}$$

where N is the number of data points in our training set, not to be confused with the dimensionality of these points. Intuitively, this is a sum of the squared losses present in any misclassifications of data points in the training set made by a given decision vector β . Because the residual sum of squares is quadratic in β , it always exhibits some local minima, although these minima are not guaranteed to be unique. To calculate these minima, we proceed as in ordinary differential calculus and find the critical points of the RSS function. Writing the RSS in matrix notation, we see

$$RSS(\beta) = (y - X\beta)^T (y - X\beta) \tag{6.0.3}$$

Differentiating with respect to β , we get the normal equation

$$X^T (y - X\beta) = 0 \tag{6.0.4}$$

the solution to which (provided $X^T X$ is nonsingular) is

$$\beta = (X^T X)^{-1} X^T y \tag{6.0.5}$$

We calculate our β below

```
# Create complete data array comprised
# of all points from both classes.
X = np.vstack((class_one, class_two))
m = len(X)

# Add column of ones to account for bias term
X = np.array([np.ones(m), X[:, 0], X[:, 1]]).T

# Create y array of class labels
y = np.concatenate((y_vals[51:], y_vals[:50])).T

# Calculate the Regularized Least Squares solution
beta = np.linalg.inv(X.T @ X) @ (X.T @ y)
```

7 Plotting the final Classified output

Finally, we plot our decision boundary. As you can see, given the extent to which our dataset is linearly separated, our linear decision boundary provides great utility in the division of input space and the binary classification of unknown data points.

```
# Create Pandas DataFrame for holding binary
    class data.
df = pd.DataFrame(data, columns=['x', 'y', 'x1', '
    x2'])
```

```
# Create scatter plot of data points in both classes.
new_ax = df.plot.scatter(x='x', y='y', color='
    Orange', label='+1');
df.plot.scatter(x='x1', y='x2', color='LightBlue',
    label='-1', ax=new_ax);

# Plot the resulting regression line
line_x = np.linspace(0, 9)
line_y = -beta[0] / beta[2] - (beta[1] / beta[2]) *
    line_x

new_ax.plot(line_x, line_y)
new_ax.set_xlim((0, 9));
new_ax.set_ylim((-5, 5));
plt.show()
```

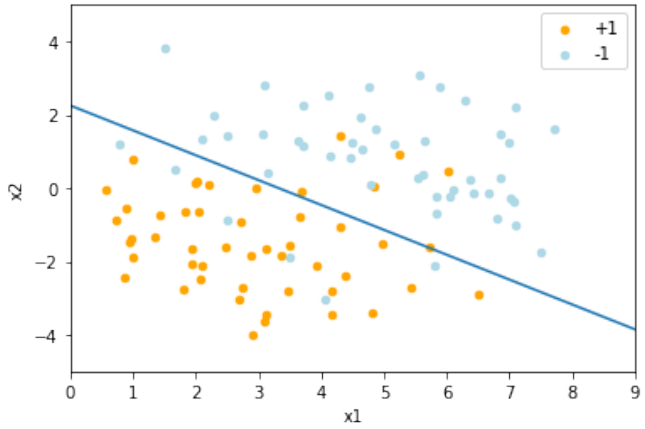

Fig. 0

Python code:

https://github.com/sahilsin/EE_IDP/blob/main/Assignment_5/ls.ipynb