LinearRegression

September 25, 2024

1 House price prediction using Linear Regression

In this assignment, you are asked to run an experimental evaluation of linear regression on the Athens houses dataset, as follows:

- Implement simple linear regression and use it to predict house prices as a function of their size in square feet.
- Implement the normal equations for multiple linear regression and use it to predict house prices as a function of their floorsize, number of bedrooms, and age.

Additionally, you are asked to:

- Use the chain rule to compute derivatives.
- Compute gradients and use them to find the parameters that minimize some simple objective functions.

1.1 Write Your Name Here:

2 Submission instructions

1. Click the Save button at the top of the Jupyter Notebook.
2. Please make sure to have entered your name above.
3. Select Cell -> All Output -> Clear. This will clear all the outputs from all cells (but will keep the content of ll cells).
4. Select Cell -> Run All. This will run all the cells in order, and will take several minutes.
5. Once you've rerun everything, select File -> Download as -> PDF via LaTeX and download a PDF version showing the code and the output of all cells, and save it in the same folder that contains the notebook file.
6. Look at the PDF file and make sure all your solutions are there, displayed correctly.
7. Submit **both** your PDF and the notebook file .ipynb on Canvas.
8. Make sure your your Canvas submission contains the correct files by downloading it after posting it on Canvas.

3 Theory

3.1 Chain rule for differentiation (10 points)

1. Compute the derivative of $h(x) = (2x+3)^2+1$ by writing $h(x) = f(g(x))$ where $g(x) = 2x+3$ and $f(g) = g^2 + 1$.

2. Compute the derivative of $h(x) = (x+1)^2 \ln(x+1)$ by writing $h(x) = f(g_1(x), g_2(x))$ where $g_1(x) = x+1$, $g_2(x) = \ln(x+1)$ and $f(g) = g_1^2 g_2$.

3.1.1 Solutions

1. *your solution goes here*
2. *your solution goes here*

3.2 Gradient-based Optimization (10 + 10 points)

By setting the gradient to 0, find the solutions to the following optimization problems:

- $\hat{x} = \arg\min_{x} J(x)$, where $J(x) = x^2 - 2x + 3$.
- $\hat{x}, \hat{y} = \arg\min_{x,y} J(x,y)$, where $J(x,y) = 2x^2 + 3y^2 - 4x + 12y + 15$.
- (Bonus) Prove that $J(x,y)$ above is a convex function.
- (Bonus) $\hat{x}, \hat{y} = \arg\min_{x,y} J(x,y)$, where $J(x,y) = x^2 + 4y^2 - 4xy + 2x - 4y + 4$.

3.2.1 Solutions

1. *your solution goes here*
2. *your solution goes here*

4 Implementation

[ ]:
```
# Import required packages.
import argparse
import sys

import numpy as np
from matplotlib import pyplot as plt
```

4.1 Read data matrix and labels from text file (5 points)

Assume that examples are stored in a text file, one example per line. Each line contains features separated by spaces, and ends with the label. We recommend that you look at the "../data/simple/train.txt" and "../data/multiple/train.txt" to see the format.

Write a function that reads the text file with examples and stores the feature vectors in the rows of a 2D data matrix `X` and the labels in a 1D vector `t`. To avoid numerical errors when training on this data later, **divide all the label values by 100** before storing them in `t`.

The function should return the tuple `(X, t)`.

For example, if the file *examples.txt* contains the following lines: 11 12 13 14 150 21 22 23 24 200 31 32 33 34 340 then `read_data('examples.txt')` should return the matrix `11 12 13 14 21 22 23 24 31 32 33 34` and the vector `[1.5, 2.0, 3.4]`.

The function should work for both the simple and multiple regression files.

Hint: you can use the `numpy.loadtxt()` *function, or* `pandas.read_csv()`*.*

```
[ ]: def read_data(file_name):
         """
         Input:
             file_name: name of the file containing labeled examples.
         Output:
             The data matrix X and the corresponding label vector t.
         """
         #  YOUR CODE here:
```

4.1.1 Root Mean Square Error (RMSE) (10 points)

Compute the RMSE that a linear regression model **w** obtains on the dataset (X, t), and return 100 * RMSE to account for the fact that the labels were divided by 100 before training.

```
[ ]: def compute_rmse(X, t, w):
         """
         Input:
             X: 2D array with rows containing feature vectors.
             t: 1D array containing the corresponding labels.
             w: 1D array containing the parameters of the lienar regression model.
         Output:
             The RMSE of w on the dataset (X, t).
         """
         # YOUR CODE HERE
```

4.1.2 Cost Function $J(w)$ (5 points)

Compute the cost $J(\mathbf{w})$ that a linear regression model **w** obtains on the dataset (X, t).

```
[ ]: # Compute objective function (cost) on dataset (X, t).
     def compute_cost(X, t, w):
         """
         Input:
             X: 2D array with rows containing feature vectors.
             t: 1D array containing the corresponding labels.
             w: 1D array containing the parameters of the lienar regression model.
         Output:
             The cost that w obtains on the dataset (X, t).
         """
         # YOUR CODE HERE
```

4.2 Simple Linear Regression

4.2.1 Training procedure (15 points)

Solve the system of linear equations for finding the parameters w_0 and w_1, as shown in class. You can code the solution that you obtain using variable elimination, or you can use the `numpy.linalg.solve()` function.

[ ]:
```
def trainSimple(X, t):
    """
    Input:
        X: 2D array with rows containing feature vectors.
        t: 1D array containing the corresponding labels.
    Output:
        The trained weight vector w = [w0, w1], according to the closed form␣
    ↪equations for lienar regression.
    """
    # YOUR CODE HERE
```

4.2.2 Experimental evaluation

[ ]:
```
# Read the training and test data.
Xtrain, ttrain = read_data('../data/simple/train.txt')
Xtest, ttest = read_data('../data/simple/test.txt')

# Train model on training examples.
w = trainSimple(Xtrain, ttrain)

# Print model parameters
print('Params: ', w) # => Params:  [-156.82270216    1.15418452]

# Print cost and RMSE on training data.
print('Training RMSE: %0.2f.' % compute_rmse(Xtrain, ttrain, w)) # => Training␣
↪RMSE: 64083.51.
print('Training cost: %0.2f.' % compute_cost(Xtrain, ttrain, w)) # => Training␣
↪cost: 205334.84.

# Print cost and RMSE on test data.
print('Test RMSE: %0.2f.' % compute_rmse(Xtest, ttest, w)) # => Test RMSE:␣
↪65773.19.
print('Test cost: %0.2f.' % compute_cost(Xtest, ttest, w)) # => Test cost:␣
↪216305.64.

#  Plot the training and test examples with different symbols.
#  Plot the linear approximation on the same graph.

plt.scatter(Xtrain, ttrain * 100, c='blue', marker='o')
plt.scatter(Xtest, ttest * 100, c='red', marker='+')
```

```
x = np.array([0, 5000])
t = w[0] + w[1] * x
plt.plot(x, t*100, '-g')
```

4.2.3 Experimental evaluation using `sklearn` (5 bonus points)

Use the sklearn package to train a linear regression model on the same dataset and print:

- The two parameters w_0 and w_1.
- The test RMSE.

```
[ ]: # YOUR CODE HERE
```

4.2.4 Multiple Linear Regression

4.2.5 Training procedure (15 points)

Compute the parameter vector $\mathbf{w}$ using the *normal equations* shown in class, $\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$.

```
[ ]: def trainMultiple(X, t):
         """
         Input:
             X: 2D array with rows containing feature vectors.
             t: 1D array containing the corresponding labels.
         Output:
             The trained weight vector w = [w0, w1], according to the closed␣
      ↪equations for lienar regression.
         """
         # YOUR CODE HERE
```

4.2.6 Experimental evaluation

```
[ ]: # Read the training and test data.
     Xtrain, ttrain = read_data('../data/multiple/train.txt')
     Xtest, ttest = read_data('../data/multiple/test.txt')

     # Train model on training examples.
     w = trainMultiple(Xtrain, ttrain)

     # Print model parameters.
     print('Params: ', w, '\n') # => Params:  [-667.13841504  0.96602209  253.
      ↪32577975  3.84475147]

     # Print cost and RMSE on training data.
     print('Training RMSE: %0.2f.' % compute_rmse(Xtrain, ttrain, w)) # => Training␣
      ↪RMSE: 61070.62.
```

```
print('Training cost: %0.2f.' % compute_cost(Xtrain, ttrain, w)) # => Training␣
↪cost: 186481.02.

# Print cost and RMSE on test data.
print('Test RMSE: %0.2f.' % compute_rmse(Xtest, ttest, w)) # => Test RMSE:␣
↪58481.32.
print('Test cost: %0.2f.' % compute_cost(Xtest, ttest, w)) # => Test cost:␣
↪171003.24.
```

4.2.7 Experimental evaluation using `sklearn` (5 bonus points)

Use the sklearn package to train a linear regression model on the same dataset and print:

- The parameter vector **w**.
- The test RMSE.

[ ]:
```
# YOUR CODE HERE
```

4.3 Bonus points

Anything extra goes here.