LinearRegression

October 8, 2024

1 House price prediction using Linear Regression trained with Gradient Descent

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

- Implement feature scaling through *standardization*.
- Implement the *gradient* of the sum-of-square-errors objective function.
- Implement training through *gradient descent*.
- Train a simple linear regression model and use it to predict house prices as a function of their size in square feet.
- Train a multiple linear regression model and use it to predict house prices as a function of their floor size, number of bedrooms, and age.

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.

2.1 Theory question on feature scaling (30p)

[11]:
```
from IPython.display import Image
Image('scaling.png', width = 700)
```

[11]:

Consider the training and test examples shown in the first table below:

Original

	Train			Test
	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$
ϕ_1	0	1	2	-1
ϕ_2	1	2	3	2
ϕ_3	2	3	4	5

Scaled $[0, 1]$

	Train			Test
	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$
ϕ_1				
ϕ_2				
ϕ_3				

Standardized $\mathcal{N}(0, 1)$

	Train			Test
	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$
ϕ_1				
ϕ_2				
ϕ_3				

Complete the two tables as explained below. Show your work.

1. Scale the features in the dataset to be between $[0, 1]$. Show the resulting scaled dataset in the second table.
2. Standardize the features in the dataset. Show the resulting standardized dataset in the third table. For the standardized values, you do not have to compute the final numbers, you can leave them in fractional form.

2.1.1 Your calculations and solutions for the two tables go here:

- Scaling solution:
- Standardization solution

2.2 Functions for reading data, creating the data matrix, computing cost and RMSE

The functions below have already been implemented in previous assignments, feel free to reuse code. To avoid numerical errors when training on this data later, make sure that you **divide all the label values by 100** before storing them in `y`.

[ ]:
```
# Import necessary Python modules.
import numpy as np
from matplotlib import pyplot as plt

# Read data X and labels y from text file.
def read_data(file_name):
    data = np.loadtxt(file_name)

    # YOUR CODE HERE
    X = []
    y = []

    return X, y


# Compute objective function (cost J) on dataset (X, t).
```

```
def compute_cost(X, t, w):
    # YOUR CODE HERE

    return 0


# Compute RMSE on dataset (X, t).
def compute_rmse(X, t, w):
    # YOUR CODE HERE


    return 0


# Create the design matrix, i.e. data matrix. Simply add the bias feature to␣
 ↪each example in X.
def data_matrix(X):
    N = X.shape[0]
    return np.column_stack((np.ones(N), X))
```

2.3 Feature scaling through Standardization (30p)

To improve the convergence of gradient descent, it is important that the features are scaled so that they have similar ranges. Implement the scaling to standard normal distribution, using the skeleton code below.

- The formulas for standard scaling are shown on the slides. Do not scale the bias feature!

[2]:
```
# Compute the sample mean and standard deviations for each feature (column)
#   across the training examples (rows) from the data matrix X.
def mean_std(X):
    mean = np.zeros(X.shape[1])
    std = np.ones(X.shape[1])

    ## Your code here.


    return mean, std


# Standardize the features of the examples in X by subtracting their mean and
#   dividing by their standard deviation, as provided in the arguments mean and␣
 ↪std.
def standardize(X, mean, std):
    S = np.zeros(X.shape)

    ## Your code here.
```

```
    return S
```

2.4 Gradient computation and training with Gradient Descent (30p)

Write code for computing the gradient of the sum-of-square-error objective J(w):

- The objective and the gradient are shown on slide 27.
- The vectorized gradient was derived in class, with the corresponding The NumPy code shown on slide 38.
- The pseudocode for the gradient descent algorithm is shown on slide6.

[3]:
```
# Compute the gradient of the objective function (cost) J(w) on dataset (X, y).
def compute_gradient(X, y, w):
    # YOUR CODE HERE
    grad = np.zeros(w.shape)

    return grad


# Implement the gradient descent algorithm to train w = [w0, w1, ..., wK] that␣
 ↪minimizes J(w).
#    Your code will need to call compute_gradient in the GD loop.
def train(X, y, eta, epochs):
    # YOUR CODE HERE
    w = np.zeros(w.shape)









    return w
```

2.5 Simple Linear Regression

Train a simple linear regression model to predict house prices as a function of their floor size, by running gradient descent for 500 epochs with a learning rate of 0.1. Use the dataset from the folder `../data/simple/`. Plot the objective $J(\mathbf{w})$ vs. the number of epochs in increments of 10 (i.e. after 0 epochs, 10 epochs, 20 epochs, …). After training print the parameters and RMSEs and compare with the solution from normal equations from previous assignment. Plot the training examples using the default blue circles and test examples using lime green triangles. On the same graph also plot the linear approximation.

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

```
# Standardize the features using the mean and std computed over *training*.
mean, std = scaling.mean_std(Xtrain)
Xtrain = scaling.standardize(Xtrain, mean, std)
Xtest = scaling.standardize(Xtest, mean, std)

# Make sure you add the bias feature to each training and test example.
Xtrain = data_matrix(Xtrain)
Xtest = data_matrix(Xtest)

# Train using gradient descent.
eta, epochs = 0.1, 500
w, J = train(Xtrain, ttrain, eta, epochs)

plt.title('Cost vs. Epochs')
plt.plot(np.arange(0, epochs + 1, 10), J[::10], label = 'Learning curve', color␣
 ↪= 'blue')
plt.xlabel('E')
plt.ylabel('J(w)')
plt.legend()
plt.savefig('simple-learning-curve.png')
plt.show()

print('w =', w)
print('RMSE on train = %0.3f.' % compute_rmse(Xtrain, ttrain, w))
print('RMSE on test = %0.3f.' % compute_rmse(Xtest, ttest, w))

#  Plot the training and test examples with different symbols.
#  Plot the linear approximation on the same graph.
plt.title('Simple Regression')
plt.scatter(Xtrain[:,1], ttrain / 10, marker = 'o',color = 'b', label =␣
 ↪'Training data')
plt.scatter(Xtest[:,1], ttest / 10, marker = '^', color = 'lime', label = 'Test␣
 ↪data')

xline = np.array([-2, 4])
pred = w[1] * xline + w[0]
plt.plot(xline, pred / 10, label = 'Regression line', color = 'red')
plt.xlabel('Floor size (square feet)')
plt.ylabel('House price (x $1000)')
plt.legend()
plt.savefig('train-test-line.png')
plt.show()
```

2.6 Multiple Linear Regression

Train a multiple linear regression model to predict house prices as a function of their floor size, number of bedrooms, and age. Run gradient descent for 200 epochs with a learning rate of 0.1. Use the the dataset from the folder `../data/multiple/`. Plot the objective $J(\mathbf{w})$ vs. the number of epochs in increments of 10 (i.e. after 0 epochs, 10 epochs, 20 epochs, …). After training print the parameters and RMSEs and compare with solution obtained using the normal equations from previous assignment.

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

#  Standardize the features using the mean and std computed over *training*.
mean, std = scaling.mean_std(Xtrain)
Xtrain = scaling.standardize(Xtrain, mean, std)
Xtest = scaling.standardize(Xtest, mean, std)

#  Make sure you add the bias feature to each training and test example.
Xtrain = data_matrix(Xtrain)
Xtest = data_matrix(Xtest)

eta, epochs = 0.1, 200
w, J = train(Xtrain, ttrain, eta, epochs)

plt.title('Cost vs. Epochs')
plt.plot(np.arange(0, epochs + 1, 10), J[::10], label = 'Learning curve', color␣
 ↪= 'blue')
plt.xlabel('E')
plt.ylabel('J(w)')
plt.legend()
plt.savefig('multiple-learning-curve.png')
plt.show()

print('w =', w)
print('RMSE on train = %0.3f.' % compute_rmse(Xtrain, ttrain, w))
print('RMSE on test = %0.3f.' % compute_rmse(Xtest, ttest, w))
```

2.7 Bonus points

- (20p) Implement SGD and run it for the simple and multiple regression problems above, using the same hyperparameters (learning rate, number of epochs). Experiment with minibatch sizes of 1 and 5. Compare the SGD solution to the batch GD solution. Does SGD Converge faster or is it the same?
- (15p) Use matplotlib or seaborn to plot a histograms of features before and after standardization.
- Anything extra that is non-trivial.

[ ]: