CS7CS4/CSU44061 Machine Learning: Week 1 Assignment

Cian Higgins 20 November 2020

# id:16-280-64

(a) (i) The data was read in with pandas and NumPy using the code given on the assignment sheet.

(ii) The data was normalised (scaled and shifted) making most of the data points lie between -1 and 1. This step ensures that large values don't dominate the cost function and prevents the training from focussing on and fitting to outliers.
Every datapoint, x_j, is replaced with $\frac{x_j - \mu_j}{\sigma_j}$, where:
the shift used is the mean: $\mu_j = \frac{1}{m}\sum_{i=1}^{n} x_j^{(i)}$
and the scaling factor is the standard deviation: $\sigma_j = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(x_j^{(i)} - \mu)^2}$

(iii) Gradient descent is used to minimise the cost function $J(\theta_0, \theta_1)$. Both parameters θ_0 and θ_1 are initialised with 0. The parameters are updated on each pass with a new value that make $J(\theta_0, \theta_1)$ smaller. The algorithm converges to a minimum (local or global depending on the data).

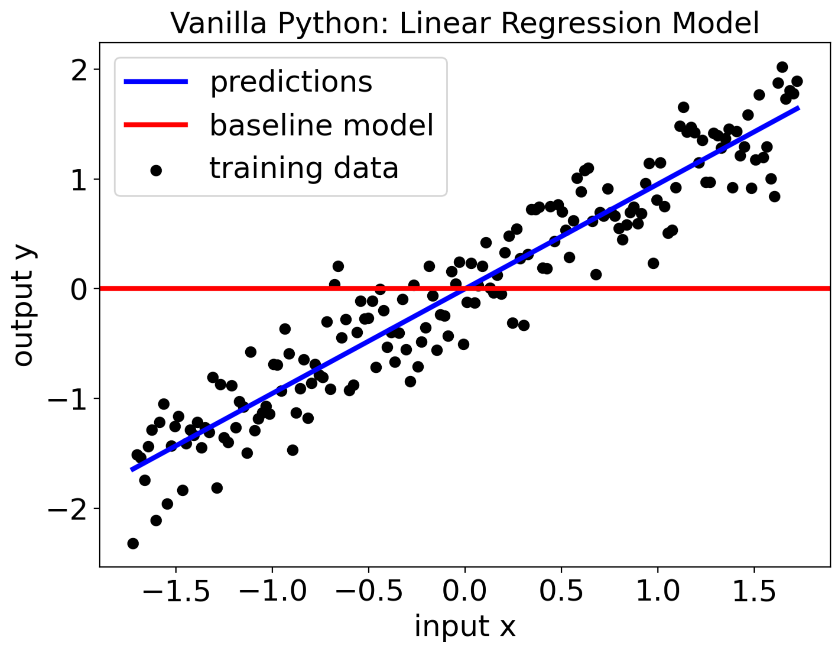
Alpha (α) is called the step size or learning rate. If it is too large the minimum can be missed. If it is too small, it will take a long time to converge. It must be chosen so that the gradient descent converges in good time (see figure 2).

Gradient descent algorithm used:

$\delta_0 := -\frac{2\alpha}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})$	$\theta_0 := \theta_0 + \delta_0$
$\delta_1 := -\frac{2\alpha}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x^{(i)}$	$\theta_1 := \theta_1 + \delta_1$

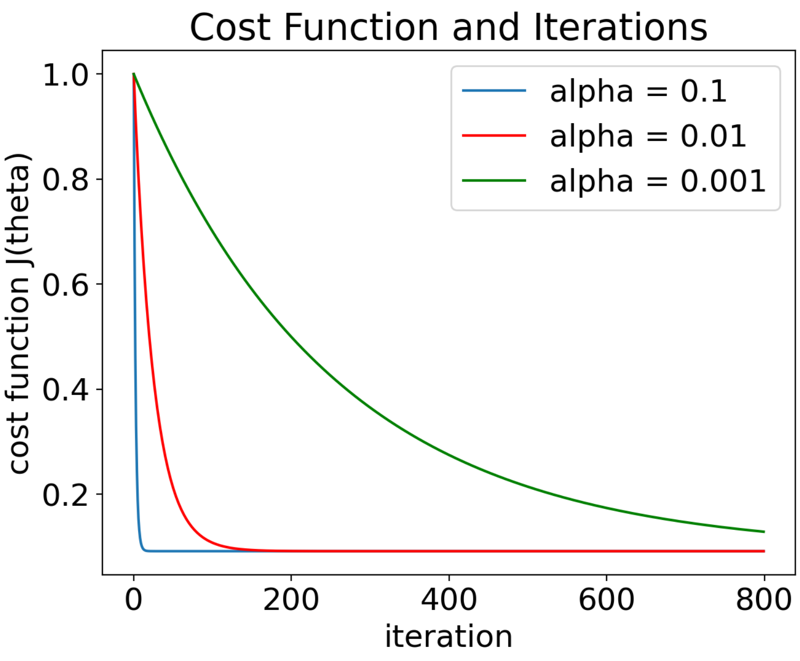
Plot of the linear regression model's predictions based on the normalised data. The baseline model is discussed further at (b)(iii).

Figure 1:



(b) (i) The plot below shows how the model converges with different values for the learning rate, α. Larger values make the model converge after relatively few iterations, and with very small values the model converges very slowly.

Figure 2:



(ii) The parameter values of the linear regression model after it has been trained on the data (using learning rate of 0.01):

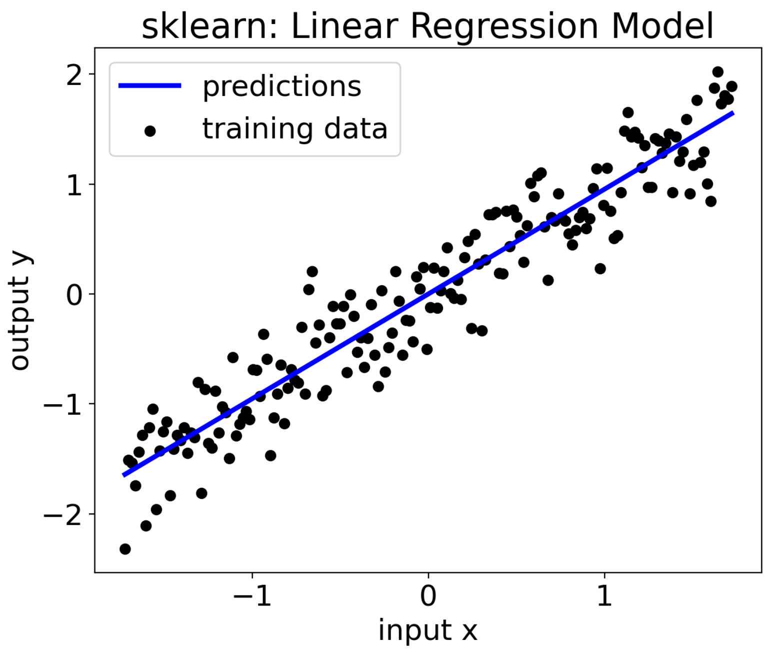
$\theta_0 = 5.547834918734839 \ \times 10^{-16}$

$\theta_1 = 0.953010786844326$

(iii) A constant value for the baseline model was obtained by finding the mean of the normalised y data. This result was $6.11884281 \times 10^{-16} \approx 0$. The number being so close to zero and based on inspection of the data, it was assumed that a reasonable value would be $\theta_1 = 0$.
$\theta_0 = 0$, a horizontal line must have a slope of zero.
Baseline model: $h(x) = \theta_0 + \theta_1 x = 0 + 0x$

(iv) The model was obtained using sklearn's preprocessing.scale() to normalise the data in much the same way as described at part (a)(ii). The LinearRegression() model was then trained with this normalised data. On visual inspection of the plots the models are very similar.

Figure 3:



Parameter values for the sklearn model:
$\theta_0 = -3.22973971 \times 10^{-16}$
$\theta_1 = 0.95301088$

The parameter θ_0 in the sklearn model differs by around 8.7757×10^{-16} when compared to that obtained from the vanilla Python gradient descent model.
The parameter θ_1 is identical for six decimal places in the two trained models.

Appendix

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn import preprocessing

# reading in data with numpy and pandas
# dataset id:16-280-64
df = pd.read_csv("week1.csv", comment='#', header=None)
print(df.head())
X = np.array(df.iloc[:, 0]); X = X.reshape(-1, 1)
y = np.array(df.iloc[:, 1]); y = y.reshape(-1, 1)
# print(X)
# print(y)


# normalising data (vanilla python)
def normalise(data):
  mean = sum(data)/len(data)  # shift
  # scale_fac = max(data) - min(data)  # max-min as scale factor range 0 to 1
  # std deviation as scale factor range -1 to 1
  scale_fac = (1/len(data) * sum([(i - mean)**2 for i in data]))**0.5

  norm_data = np.zeros(shape=(len(data), 1))
  for i in range(len(data)):
      norm_elem = (data[i] - mean) / scale_fac
      norm_data[i] = norm_elem
  return norm_data


norm_X = normalise(X)
norm_y = normalise(y)

# using mean of normalised y values for
baseline_model = sum(norm_y)/len(norm_y)
print("base:", baseline_model)
m = len(norm_X)
max_iter = 800


def gradient_descent(alpha):
  costs = []
  theta0 = 0
  theta1 = 0

  for index in range(max_iter):
      pred = theta0 + (theta1 * norm_X)  # predictions h(x)

      # gradient descent algorithm
      # calculating delta0 and delta1
      delta0 = (-2/m) * np.sum(norm_y - pred)
      delta1 = (-2/m) * np.sum((norm_y - pred) * norm_X)

      # updating theta0 and theta1 with the above calculated values
      theta0 = theta0 - (alpha * delta0)
      theta1 = theta1 - (alpha * delta1)

      cost_func = (1/m) * np.sum((norm_y - pred)**2)  # cost function 
calculation
      costs.append(cost_func)  # list of cost function results
      # print(index, theta0, theta1)
  pred = theta0 + (theta1 * norm_X)
  print("Parameter values:\nTheta0=", theta0, "  Theta1=", theta1)
  print("Cost function val=", costs[-1])
  return costs, pred


cost1, pred1 = gradient_descent(0.1)
cost2, pred2 = gradient_descent(0.01)
cost3, pred3 = gradient_descent(0.001)

# Plotting data points and predictions
plt.rc('font', size=18)
plt.scatter(norm_X, norm_y, color='black')
plt.plot([min(norm_X), max(norm_X)], [min(pred1), max(pred1)], color='blue', 
linewidth=3)
plt.axhline(y=baseline_model, color='r', linewidth=3)
plt.title("Vanilla Python: Linear Regression Model", size=18)
plt.xlabel("input x")
plt.ylabel("output y")
plt.legend(["predictions", "baseline model", "training data"])
plt.show()

# Plotting cost function results and iterations
plt.plot(list(range(max_iter)), cost1)
plt.plot(list(range(max_iter)), cost2, color='red')
plt.plot(list(range(max_iter)), cost3, color='green')
plt.title("Cost Function and Iterations")
plt.xlabel("iteration")
plt.ylabel("cost function J(theta)")
plt.legend(["alpha = 0.1", "alpha = 0.01", "alpha = 0.001"])
plt.show()

# using sklearn to train a linear regression model on the data
X_scaled = preprocessing.scale(X)
y_scaled = preprocessing.scale(y)

model = LinearRegression()
model.fit(X_scaled, y_scaled)
model_y_pred = model.predict(X_scaled)
print(model.intercept_, model.coef_)

plt.scatter(X_scaled, y_scaled,  color='black')
plt.plot(X_scaled, model_y_pred, color='blue', linewidth=3)
plt.title("sklearn: Linear Regression Model")
plt.rc('font', size=18)
plt.xlabel("input x")
plt.ylabel("output y")
plt.legend(["predictions", "training data"])
plt.show()
```