hw3_problem3_starter_code

April 9, 2022

1 Logistic Regression vs Linear Regression via Gradient Descent

[1]:
```
# Let us first import some modules for this problem
import numpy as np
import matplotlib.pyplot as plt
import math

%matplotlib inline

np.random.seed(1)
```

[2]:
```
def sigmoid(z):
    """
  sigmoid function that maps inputs into the interval [0,1]
  Your implementation must be able to handle the case when z is a vector (see␣
 ↪unit test)
  Inputs:
  - z: a scalar (real number) or a vector
  Outputs:
  - trans_z: the same shape as z, with sigmoid applied to each element of z
  """
  # *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

    if (isinstance(z, (list, np.ndarray))):
        size = z.size
        ans = np.zeros(size)

        for i in range(size):
            ans[i] = 1/(1 + (math.exp(-z[i])))

        return ans

    trans_z = 1/(1 + (math.exp(-z)))


  # *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
    return trans_z
```

```
def logistic_regression(X, w):
    """
  logistic regression model that outputs probabilities of positive examples
  Inputs:
  - X: an array of shape (num_sample, num_features)
  - w: an array of shape (num_features,)
  Outputs:
  - logits: a vector of shape (num_samples,)
  """
  # *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
    size = X.size
    size2 = X[0].size

    logits = sigmoid(np.matmul(X,w))

  # *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
    return logits

# unit test
# sample inputs:
z = np.array([215, -108, 0, 0.32])
X = np.array([[4.17022005e-01, 7.20324493e-01, 1.14374817e-04],
              [3.02332573e-01, 1.46755891e-01, 9.23385948e-02],
              [1.86260211e-01, 3.45560727e-01, 3.96767474e-01],
              [5.38816734e-01, 4.19194514e-01, 6.85219500e-01]])
w = np.array([0.20445225, 0.87811744, 0.02738759])

# sample outputs:
out1 = sigmoid(z)
out2 = logistic_regression(X, w)
print(out1)
print(out2)
# out1 : [1.00000000e+00 1.24794646e-47 5.00000000e-01 5.79324252e-01]
# out2 : [0.67212099 0.5481529  0.5871972  0.62176131]
```

```
[1.00000000e+00 1.24794646e-47 5.00000000e-01 5.79324252e-01]
[0.67212099 0.5481529  0.5871972  0.62176131]
```

[4]:
```
def logistic_loss(X, w, y):
    """
  a function that compute the logistic loss value for the given dataset (X, y)␣
 ↪and parameter w;
  It also returns the gradient of loss function w.r.t w
  Here (X, y) can be a set of examples, not just one example.
  Remember -- we always take the average (divide by size of dataset) while␣
 ↪calculating loss, gradient etc.
```

```
  Inputs:
  - X: an array of shape (num_sample, num_features)
  - w: an array of shape (num_features,)
  - y: an array of shape (num_sample,), it is the ground truth label of data X
  Output:
  - loss: a scalar which is the value of loss function for the given data and␣
 ↪parameters
  - grad: an array of shape (num_featues,), the gradient of loss
  """
  # *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
    z = logistic_regression(X, w)
    m = y.size
    loss = 0
    grad = (np.matmul(np.transpose(X),z-y))/m

    for i in range(m):
        if (y[i]):
            loss -= math.log(z[i])
        else:
            loss -= math.log(1-z[i])

    z = np.matmul(X,w)
    loss = loss/m
  # *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
    return loss, grad
#unit test
# inputs:
X = np.array([[0.67046751, 0.41730480, 0.55868983],
              [0.14038694, 0.19810149, 0.80074457],
              [0.96826158, 0.31342418, 0.69232262],
              [0.87638915, 0.89460666, 0.08504421]])
w = np.array([0.03905478, 0.16983042, 0.8781425 ])
Y = np.array([1, 1, 0, 1])

# sample outputs:
loss, grad = logistic_loss(X, w, Y)
print(loss)
print(grad)
# loss: 0.626238298577102
# grad: [-0.00483685, -0.09821878, -0.0080873 ]
```

```
0.626238298577102
[-0.00483685 -0.09821878 -0.0080873 ]
```

[94]:
```
def square_loss(X, w, y):
    """
```

```
  a function that compute the linear regression loss value for the given␣
↪dataset (X, y) and parameter w;
  It also returns the gradient of loss function w.r.t w
  Here (X, y) can be a set of examples, not just one example.
  Remember -- we always take the average (divide by size of dataset) while␣
↪calculating loss, gradient etc.
  Inputs:
  - X: an array of shape (num_sample, num_features)
  - w: an array of shape (num_features,)
  - y: an array of shape (num_sample,), it is the ground truth label of data X
  Output:
  - loss: a scalar which is the value of loss function for the given data and␣
↪parameters
  - grad: an array of shape (num_featues,), the gradient of loss
  """
  # *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
    m = y.size
    loss = (np.sum(np.power((np.matmul(X, w) - y),2)))/m

    grad = (2*(np.matmul(np.matmul(np.transpose(X),X),w)) - 2*(np.matmul(np.
↪transpose(X),y)))/m

  # *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
    return loss, grad
#unit test
# inputs:
X = np.array([[0.67046751, 0.41730480, 0.55868983],
              [0.14038694, 0.19810149, 0.80074457],
              [0.96826158, 0.31342418, 0.69232262],
              [0.87638915, 0.89460666, 0.08504421]])
w = np.array([0.03905478, 0.16983042, 0.8781425 ])
Y = np.array([1, 1, 0, 1])

# sample outputs:
loss, grad = square_loss(X, w, Y)
print(loss)
print(grad)
# loss: 0.317848572145887241
# grad: [-0.14180542, -0.33264742, -0.007825 ]
```

```
0.31784857214588724
[-0.14180542 -0.33264742 -0.007825  ]
```

1.1 1.2 Recognizing hand-written digits with logistic regression

We have gone through all the theoretical concepts of the logistic regression model. It's time to put hands on a real problem in which we aim to recognize images of hand-written digits. The dataset

we will use is the Optical Recognition of Handwritten Digits dataset, and the description of the dataset can be found here.

1.2.1 Data preprocessing (not for grading)

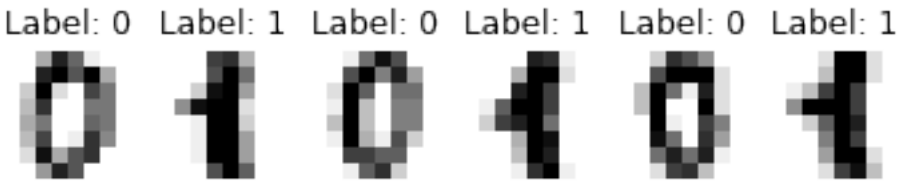
The original dataset contains 10 classes (digits 0 to 9). Since for now we are concerned about logistic regression for binary classification, we will only use a subset of the dataset that contains 360 examples from 2 classes (digits 0 and 1). Each example is a 8×8 matrix (image) where each element is an integer in the range $[0, 16]$. Let's load the dataset by using the off-the-shell method from `sklearn` and print out some images to get a good understanding of the dataset.

[67]:
```
# set up the code for this experiment
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits

%matplotlib inline
np.random.seed(1)

# load the digits dataset
digits = load_digits(n_class=2)
# digits is a dictionary-like object that hold all the features and labels,
# along with some metadata about the dataset.
# The features are stored in the '.data' member, a (#sample, #feature) array.
# The labels are stored in the '.target' member.

print(f'There are {len(digits.target)} examples in total.')
print(f'All examples are images of hand-written digit {list(set(digits.
 ↪target))[0]} or hand-written digits {list(set(digits.target))[1]}')
print(f'Each example is an array of shape {digits.data[0].shape}')
print(f'An example of data point:\n{digits.data[0]}')

# You may wondering why the shape of data is (64,) instead of (8, 8). Actually,
# You can access to matrix shape of data through the '.images' member.
print(f'The shape of image is {digits.images[0].shape}')
print(f'An example of 2D array data:\n {digits.images[0]}')
```

```
There are 360 examples in total.
All examples are images of hand-written digit 0 or hand-written digits 1
Each example is an array of shape (64,)
An example of data point:
[ 0.  0.  5. 13.  9.  1.  0.  0.  0.  0. 13. 15. 10. 15.  5.  0.  0.  3.
 15.  2.  0. 11.  8.  0.  0.  4. 12.  0.  0.  8.  8.  0.  0.  5.  8.  0.
  0.  9.  8.  0.  0.  4. 11.  0.  1. 12.  7.  0.  0.  2. 14.  5. 10. 12.
  0.  0.  0.  0.  6. 13. 10.  0.  0.  0.]
The shape of image is (8, 8)
An example of 2D array data:
 [[ 0.  0.  5. 13.  9.  1.  0.  0.]
```

```
 [ 0.  0. 13. 15. 10. 15.  5.  0.]
 [ 0.  3. 15.  2.  0. 11.  8.  0.]
 [ 0.  4. 12.  0.  0.  8.  8.  0.]
 [ 0.  5.  8.  0.  0.  9.  8.  0.]
 [ 0.  4. 11.  0.  1. 12.  7.  0.]
 [ 0.  2. 14.  5. 10. 12.  0.  0.]
 [ 0.  0.  6. 13. 10.  0.  0.  0.]]
```

[33]:
```
# The data we are interested in is made up of 8x8 images of digits.
# Let's have a look at the first 6 images that are drawn from the dataset.
# For these images,
#   we know the digit they represented is given in the 'target' of the dataset.
_, axes = plt.subplots(1, 6)
img_label = list(zip(digits.images, digits.target))
for ax, (img, target) in zip(axes, img_label[:6]):
  ax.set_axis_off()
  ax.imshow(img, cmap=plt.cm.gray_r, interpolation='nearest')
  ax.set_title('Label: %i' % target)
plt.show()
```



1.2.2 Padding features (not for grading)

As we did in Homework 2, to simplify the notation, we pad the input x by inserting 1 to the **beginning** so that we can absorb the bias term into the parameter w.

The following code morphs the variable `digits.data` by concatenating 1 and features.

[34]:
```
ones = np.ones(digits.data.shape[0]).reshape(-1, 1)
digits.data = np.concatenate((ones, digits.data), axis=1)
print(digits.data.shape)
```

```
(360, 65)
```

1.2.3 Create training and test sets (not for grading)

As we have practiced in our previous homework, we will use the `train_test_split()` method to partition the dataset into training and test sets. In this experiment, we use 80% data for training and the remaining 20% data for testing. To ensure your results are replicable, we will set the `random_state` argument of `train_test_split()` to **1**.

[35]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(digits.data, digits.target,␣
↪train_size=0.8, random_state=1)
print(f'The training set contains {X_train.shape[0]} examples.')
print(f'The testing set contains {X_test.shape[0]} examples.')
```

```
The training set contains 288 examples.
The testing set contains 72 examples.
```

1.2.3 Feature Normalization (not for grading)

Since, we have implemented the function `featureNormalization()` to normalize the features that have different scale, now, we will learn to use the built-in function `StandardScaler()` in `scikit-learn`. As we did in `featureNormalization()`, `StandardScaler()` returns standardized features by removing the mean and scaling to unit variance. Please read through the API documentation for detailed instructions.

[37]:
```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
sc.fit(X_train)
X_train = sc.transform(X_train)
X_test = sc.transform(X_test)
```

1.1.1 Training the model with gradient descent

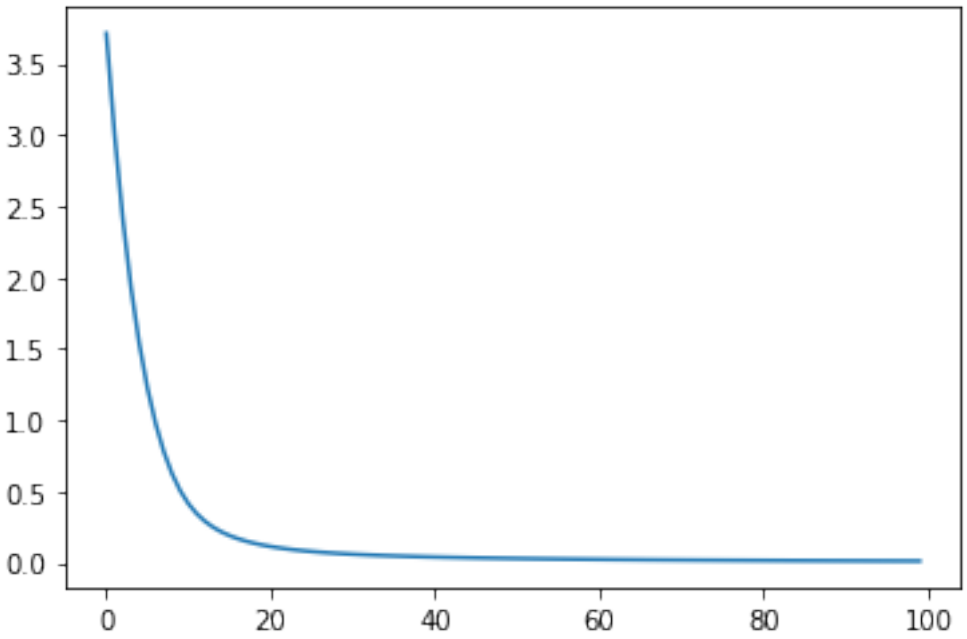
Now after all the pre-processing, we can train a logisitic regression model with the training data. It is quite straightforward to make predictions on test data by using the learned model. To simplify the task, when the probability of being positive is greater than 0.5, we classify the sample to 1. Otherwise, we classify it to 0.

In this part, we will train the model with gradient descent. After that, predict the label for test examples and compute the test accuracy. You may want to follow the procedures below to obtain the results: + Randomly initialize the parameter w by `np.random.rand`. + Use gradient descent to update w (number of iteration `num_iters` and learning rate `lr` are provided). + Plot the curve of the $\ell(w)$ value as a function of how many update steps have been taken (you need a variable to store the history of $\ell(w)$ values). + Compute and report the test accuracy on test set.

[132]:
```
num_iters = 200
lr = 0.1

# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

w1 = np.random.rand(65)
w2 = np.random.rand(65)
n = X_train.size
x = []
```

```
y1 = []
y2 = []

# training
for i in range(num_iters):
    loss1, grad1 = square_loss(X_train, w1, y_train)
    loss2, grad2 = logistic_loss(X_train, w2, y_train)

    w1 = w1 - (lr*grad1)
    w2 = w2 - (lr*grad2)

    x.append(i)
    y1.append(loss1)
    y2.append(loss2)

# plotting
plt.plot(x, y2)

# testing


# *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
```

[132]: [<matplotlib.lines.Line2D at 0x2707b9796c8>]



[ ]:
```
Best accuracy
lr: T = 800, lr = 0.6
logr: T = 1000, lr = 0.8
```

2 Q1 , Q2 in the BB zip