LR-Numpy

November 14, 2024

1 Logistic Regression in NumPy

In this assignment, you will implement the logistic regression model using NumPy and evaluate it on 3 artificial datasets: *shapes*, *flower*, and *spiral*.

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 notebook 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 Properties of the logistic sigmod and the softmax (15p)

Prove that: 1. The probability of the negative class when the logit score z is known can be computed as $1 - \sigma(z) = \sigma(-z)$

2. The derivative of the sigmoid is $\sigma'(z) = \sigma(z)(1 - \sigma(z))$

3. Subtracting a constant c from all logit score does not change the probabilities computed by softmax, i.e., $softmax(z_1 - c, z_2 - c, ..., z_K - c) = softmax(z_1, z_2, ..., z_K)$

3.1.1 YOUR SOLUTION goes here.

3.2 On the fitting power of Logistic Regression (20p)

Consider a training set that contains the 4 training examples shown in the table below. Each training example $\mathbf{x}$ has 2 features x_1 and x_2 and a label $y \in \{0, 1\}$.

$\mathbf{x}$	x_1	x_2	y
$\mathbf{x}^{(1)}$	0	0	0
$\mathbf{x}^{(2)}$	0	1	1
$\mathbf{x}^{(3)}$	1	0	1
$\mathbf{x}^{(4)}$	1	1	0

Prove that no binary logistic regression model can perfectly classify this dataset.

Hint: Prove that there cannot be a vector of parameters $\mathbf{w} = [w_1, w_2]$ *and bias* b *such that* $P(y = 1|\mathbf{x}; \mathbf{w}, b) \geq 0.5$ *for all examples* $\mathbf{x}$ *that are positive, and* $P(y = 1|\mathbf{x}; \mathbf{w}, b) < 0.5$ *for all examples* $\mathbf{x}$ *that are negative.*

3.2.1 YOUR SOLUTION goes here.

3.3 Binary vs. Multiclass Logistic Regression (bonus 10p)

Show that binary Logistic Regression is a special case of multiclass Logistic (Softmax) Regression. That is to say, if $\mathbf{w}_1$ and $\mathbf{w}_2$ are the parameter vectors of a Softmax Regression model for the case of two classes, then there exists a parameter vector $\mathbf{w}$ for binary Logistic Regression that computes the same probabilities for the two classes as the Softmax Regression model, for any input feature vector $\mathbf{x}$. It goes without saying that the proof should work for any $\mathbf{w}_1$ and $\mathbf{w}_2$.

Hint: Find $\mathbf{w}$ *as a function of* $\mathbf{w}_1$ *and* $\mathbf{w}_2$*.*

3.3.1 YOUR SOLUTION goes here.

3.4 Gradient computation (bonus 20p)

Show that the negative log-likelihood loss (shown on slide 40) has the gradient below below (also shown on slide 41):

$$\frac{\delta L(\mathbf{w})}{\delta \mathbf{w}} = \sum_{n=1}^{N} (\hat{y}_n - y_n)\mathbf{x}_n$$

3.4.1 YOUR SOLUTION goes here.

4 Implementation

[6]:
```python
import numpy as np
from scipy.sparse import coo_matrix
from numpy.random import randn, randint
from numpy.linalg import norm
import matplotlib.pyplot as plt

import utils
```

```
np.random.seed(1)
```

4.1 The Logistic Regression model (60 points)

In this part, you will write code to:

1. Compute the cost and the gradient for the logistic regression model (40 points)
2. Compute the model predictions (20 points).

While not essential for this assignment, it is important to vectorize your code so that it runs quickly on larger datasets.

Implement the *softmax()* function, which computes and return the cost and its gradient with respect to the parameters **W** and **b**.

- **Cost & Gradient**: The cost and gradient should be computed according to the formulas shown on the slides 73 to 76, where we represent explicitly the bias and its gradient. Thus, if there are D features and K classes, the softmax model will be comprised of two types of parameters: **W** will be a K x D matrix of the feature weights, whereas **b** will be a K x 1 vector of bias terms.
- **Ground truth**: The *groundTruth* is a matrix M such that M[c, n] = 1 if sample n has label c, and 0 otherwise. This can be done quickly, without a loop, using the SciPy function *sparse.coo_matrix()*. Specifically, coo_matrix((data, (i, j))) constructs a matrix A such that A[i[k], j[k]] = data[k], where the shape is inferred from the index arrays. Sample code for computing the ground truth matrix has been provided in the lecture slides.
- **Overflow**: Make sure that you prevent overflow when computing the softmax probabilities, as shown on the slides 71 & 72.

[1]:
```
def softmaxCost(W, b, numClasses, inputSize, decay, data, labels):
    """Computes and returns the (cost, gradient)
    Args:
        W, b - the weight matrix and bias vector parameters.
        numClasses - the number of classes.
        inputSize - the size D of each input vector.
        decay - weight decay parameter.
        data - the D x N input matrix, where each column data[:,n] corresponds␣
 ↪to a single sample.
        labels - an N x 1 matrix containing the labels corresponding for the␣
 ↪input data.

    Returns:
        The cost, W's gradient, and b's gradient.
    """

    N = data.shape[1]
    groundTruth = coo_matrix((np.ones(N, dtype = np.uint8), (labels, np.
↪arange(N)))).toarray()
    cost = 0;
    dW = np.zeros((numClasses, inputSize))
    db = np.zeros((numClasses, 1))

    ## ---------- YOUR CODE HERE --------------------------------------
    #  Instructions: Compute the cost and gradient for softmax regression.
    #                You need to compute dW, dW, and cost.
    #                The groundTruth matrix might come in handy.










    return cost, dW, db
```

Implement the *softmaxPredict()* function, which computes the most likely label for each example in the input data matrix.

[18]:
```
def softmaxPredict(W, b, data):
    """Computes and returns the softmax predictions over the examples in data.
    Args:
        W, b - model parameters trained using softmaxTrain, a numClasses x D␣
↪matrix and a numClasses x 1 column vector.
        data - the D x N input matrix, where each column data[:,n] corresponds␣
↪to a single sample.

    Returns:
        pred - a vector of N predicted labels.
    """

    pred = np.zeros(data.shape[1])

    #  Your code should produce the prediction matrix pred,
    #  where pred(i) is argmax_c P(c | x(i)).

    ## ---------- YOUR CODE HERE --------------------------------------------
    #  Instructions: Compute pred using W and b, assuming that the labels
    #                start from 0.
    # ---------------------------------------------------------------------

    return pred
```

4.2 2. Gradient Checking of the LR Model (20 points)

In this part you will write code to compute the gradient numerically, and then compare the numerical gradient with the analytical gradient. They should be very close.

Implement the *computeNumericalGradient()* function

```
[3]: def computeNumericalGradient(J, theta):
         """Compute numgrad = computeNumericalGradient(J, theta)
         Args:
             theta: A matrix of parameters (numClasses x D)
             J:     The cost function that outputs a real-number cost and the␣
       ↪gradient with respect to its parameters.
                    Calling y = J(theta)[0] will return the cost function evaluated␣
       ↪at theta.

         Returns:
             numgrad: The matrix of partial derivatives with respect to the␣
       ↪parameters theta (numClasses x D)
         """

         # Initialize numgrad with zeros
         numgrad = np.zeros(theta.shape)

         ## ---------- YOUR CODE HERE --------------------------------------
         # Instructions:
         # Implement numerical gradient checking, and return the result in numgrad.
         # You should write code so that numgrad[i][j] is (the numerical␣
       ↪approximation to) the
         # partial derivative of J with respect to theta[i][j], evaluated at theta.
         # I.e., numgrad[i][j] should be the (approximately) partial derivative of J␣
       ↪with
         # respect to theta[i][j].
         #
         # Hint: You will probably want to compute the elements of numgrad one at a␣
       ↪time.
```

```
    ## --------------------------------------------------------------

    return numgrad
```

Check that the numerically computed gradients are sufficiently close to the gradient computed analytically.

```
[ ]: # Create a dataset with 100 examples, each with 8 random features.
inputSize, numClasses = 8, 2
instances, labels = randn(8, 100), randint(0, numClasses, 100, dtype = np.uint8)

# Randomly initialize parameters
W = 0.01 * randn(numClasses, inputSize)
b = np.zeros((numClasses, 1))

decay = 0.001
cost, dW, db = softmaxCost(W, b, numClasses, inputSize, decay, instances,␣
 ↪labels)
W_numGrad = computeNumericalGradient(lambda x: softmaxCost(x, b, numClasses,␣
 ↪inputSize,
                                                             decay, instances,␣
 ↪labels), W)

# Use this to visually compare the gradients side by side.
print(np.stack((W_numGrad.ravel(), dW.ravel())).T)

# Compare numerically computed gradients with those computed analytically.
diff = norm(W_numGrad - dW) / norm(W_numGrad + dW)
print(diff)

# The difference should be small.
# In our implementation, these values are usually less than 1e-7.
```

4.3 3. Experimental Evaluations (10 points)

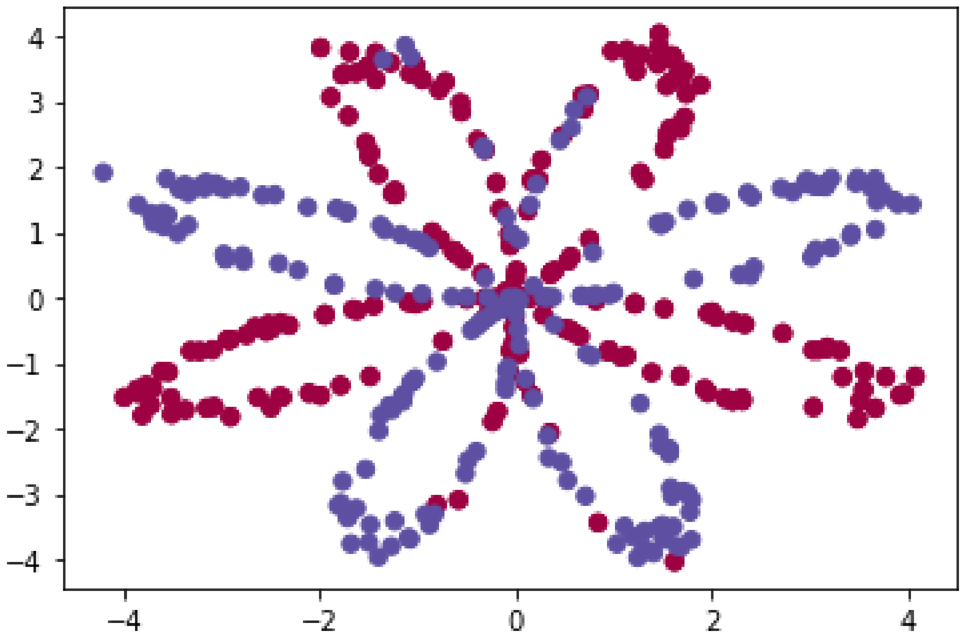
In this part, you will train and evaluate the LR model on 3 artificial datasets: *shapes*, *flower*, and *spiral*.

```
[16]: def softmaxExercise(instances, labels, numClasses):
    """Train and evaluate the LR model.
    Args:
        instances - the D x N input matrix, where each column data[:,n]␣
 ↪corresponds to a single sample.
        labels - an N x 1 matrix containing the labels corresponding for the␣
 ↪input data.
```

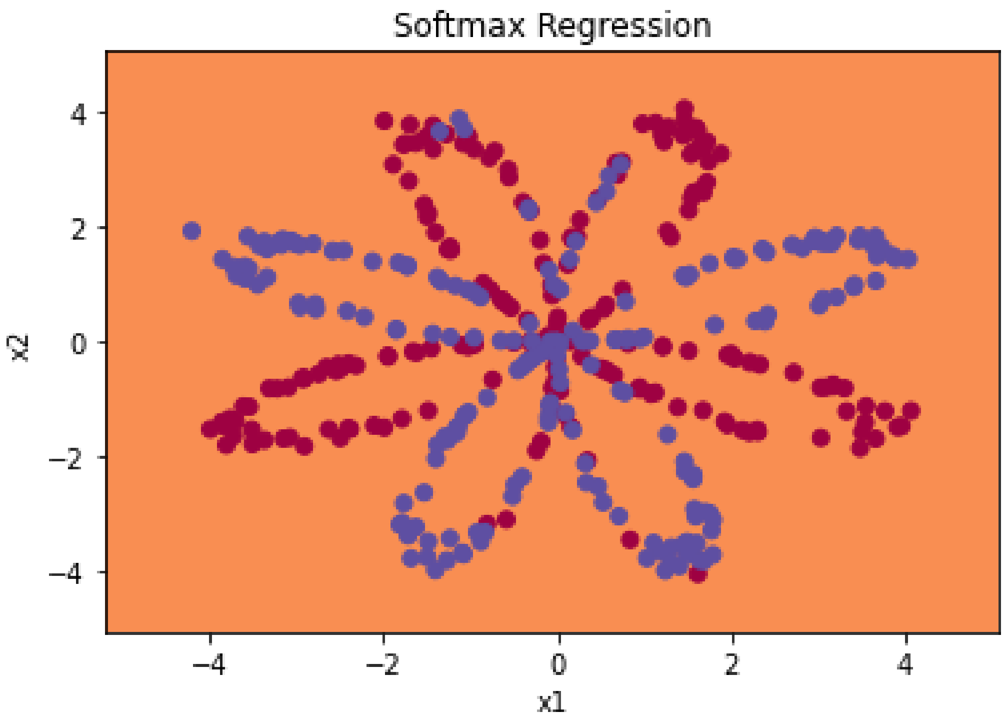
```
        numClasses - the number of classes.

    Returns:
        W, b - the trained parameters.
    """

    inputSize = instances.shape[0]

    # Randomly initialize parameters.
    W = 0.01 * randn(numClasses, inputSize)
    b = np.zeros((numClasses, 1))

    # Set hyper-parameters.
    learning_rate = 1.0
    decay = 0.001
    num_epochs = 200

    # Gradient descent loop.
    for epoch in range(num_epochs):
        # Call softmaxCost to compute cost and gradients.
        cost, dW, db = softmaxCost(W, b, numClasses, inputSize, decay,␣
↪instances, labels)
        if epoch % 10 == 0:
            print("Epoch %d: cost %f" % (epoch, cost))

        # Update parameters.
        W += -learning_rate * dW
        b += -learning_rate * db


    ## Testing on training data.
    #
    #  You should now test your model against the training examples.
    #  To do this, you will call softmaxPredict, which should return
    #  predictions given a softmax model and the input data.

    pred = softmaxPredict(W, b, instances)

    acc = np.mean(labels == pred)
    print('Accuracy: %0.3f%%.' % (acc * 100))

    # Accuracy is the proportion of correctly classified images
    # After 200 epochs, the results for our implementation were:
    #
    # Spiral Accuracy: 53.3%
    # Flower Accuracy: 47.0%
    return W, b
```

4.3.1 Evaluate the LR model on the *flower* dataset.

[17]:
```
instances, labels, numClasses = utils.load_flower_dataset()
W, b = softmaxExercise(instances, labels, numClasses)

# Plot the decision boundary.
utils.plot_decision_boundary(lambda x: softmaxPredict(W, b, x), instances,␣
 ↪labels)
plt.title("Softmax Regression")
plt.savefig('flower-boundary.jpg')
plt.show()
```

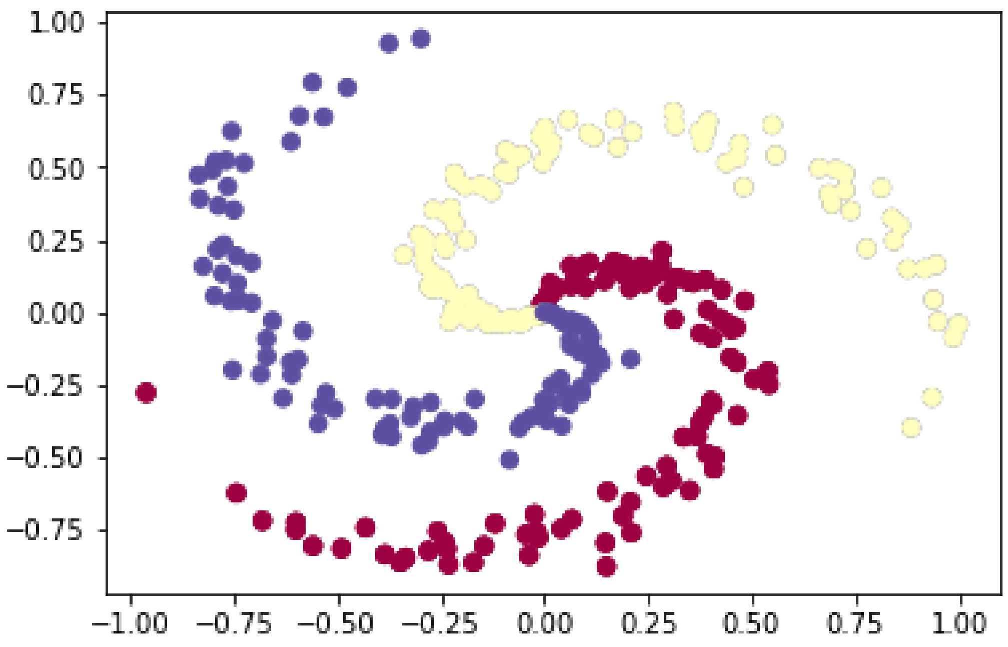


```
Epoch 0: cost 0.000000
Epoch 10: cost 0.000000
Epoch 20: cost 0.000000
Epoch 30: cost 0.000000
Epoch 40: cost 0.000000
Epoch 50: cost 0.000000
Epoch 60: cost 0.000000
Epoch 70: cost 0.000000
Epoch 80: cost 0.000000
Epoch 90: cost 0.000000
Epoch 100: cost 0.000000
Epoch 110: cost 0.000000
Epoch 120: cost 0.000000
Epoch 130: cost 0.000000
Epoch 140: cost 0.000000
Epoch 150: cost 0.000000
Epoch 160: cost 0.000000
Epoch 170: cost 0.000000
Epoch 180: cost 0.000000
Epoch 190: cost 0.000000
(400,)
Accuracy: 50.000%.
(1038240,)
```

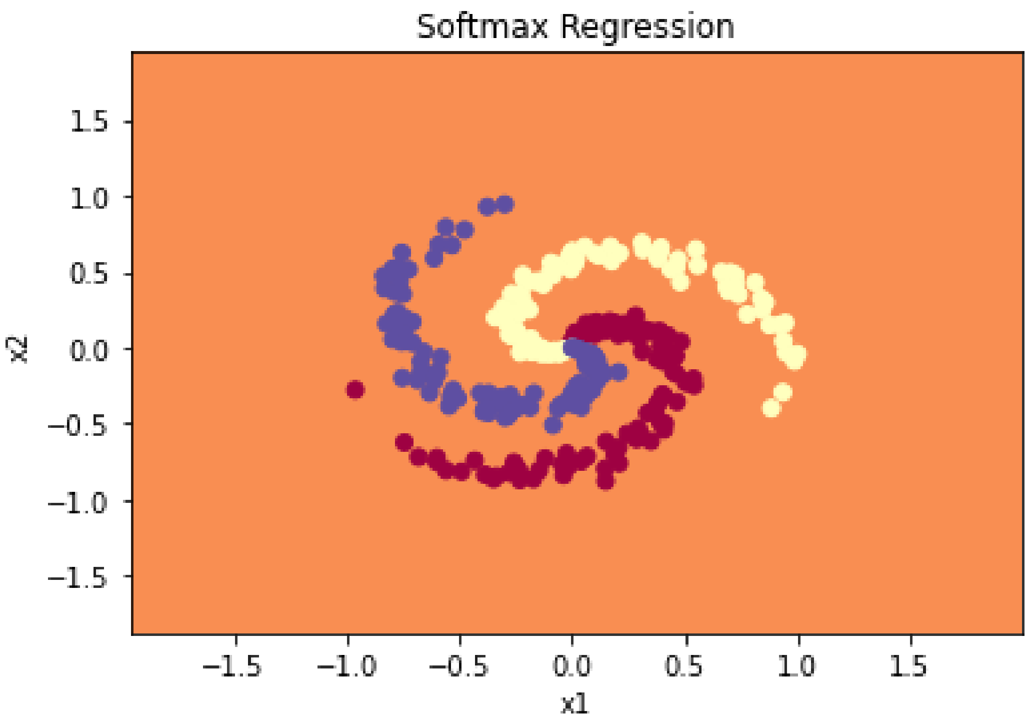


4.3.2 Evaluate the LR model on the *spiral* dataset.

[19]:
```
instances, labels, numClasses = utils.load_spiral_dataset()
W, b = softmaxExercise(instances, labels, numClasses)

# Plot the decision boundary.
utils.plot_decision_boundary(lambda x: softmaxPredict(W, b, x), instances,␣
 ↪labels)
plt.title("Softmax Regression")
plt.savefig('spiral-boundary.jpg')
plt.show()
```



```
Epoch 0: cost 0.000000
Epoch 10: cost 0.000000
Epoch 20: cost 0.000000
Epoch 30: cost 0.000000
Epoch 40: cost 0.000000
Epoch 50: cost 0.000000
Epoch 60: cost 0.000000
Epoch 70: cost 0.000000
Epoch 80: cost 0.000000
Epoch 90: cost 0.000000
Epoch 100: cost 0.000000
Epoch 110: cost 0.000000
Epoch 120: cost 0.000000
Epoch 130: cost 0.000000
Epoch 140: cost 0.000000
Epoch 150: cost 0.000000
Epoch 160: cost 0.000000
Epoch 170: cost 0.000000
Epoch 180: cost 0.000000
Epoch 190: cost 0.000000
Accuracy: 33.333%.
```



4.3.3 Evaluate the LR model on the *shapes* dataset (10 points)

[ ]:
```
instances, labels, numClasses = # YOUR CODE HERE

W, b = softmaxExercise(instances, labels, numClasses)
```

[ ]:
```

```

4.4 4. Bonus (20 + 20 points)

- Use the *sklearn.linear_model.LogisticRegression* class from sklearn to run the 3 experimental evaluations above.

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

- Modify the data generation functions to create examples that have only two labels and write a second version of the assignment that implements logistic regression for binary classification. Evaluate it on the binary classification dataset for `flower` and compare with softmax regression.
- Anything extra goes here.

[ ]:
```

```

4.5 5. Analysis (10 points)

Include here a nicely formatted report of the results, comparisons. Include explanations and any insights you can derive from the algorithm behavoir and the results. This section is important, so make sure you address it appropriately.

```
[ ]:
```