Machine Learning, 2018-19

Coursework Part II

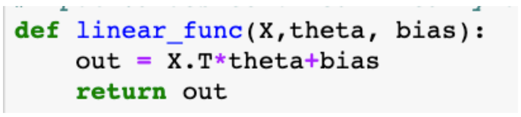
Report

1. Implementation write-up

Logistic regression:

Logistic regression is an algorithm used to solve supervised classification problems, measuring likelihood occurrence of an event.
It uses a linear function to represent input values; here inputs values are combined linearly using coefficients. **(see Figure 1)**

In the implementation, the linear function is defined as **linear_func**:



```python
def linear_func(X,theta, bias):
    out = X.T*theta+bias
    return out
```

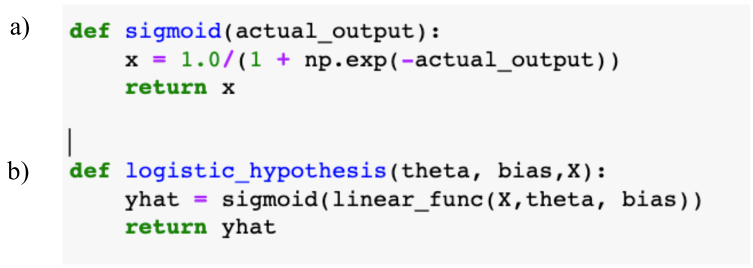
Figure 1, theta being the coefficients, and **X** the inputs.

The standard logistic function, is an S shaped curve which takes a real number and maps it into a value between 0 and 1. For Logistic regression uses a special case of the standard logistic function, called sigmoid **(see Figure 2(a)).**

The term "sigmoid" means S-shaped: see Figure 1.19(a) for a plot. It is also known as a squashing function, since it maps the whole real line to [0, 1], which is necessary for the output to be interpreted as a probability [1]

Passing the linear combination of the inputs, through the sigmoid function gives the logistic regression equation **(see Figure 2(b)).**

Figure 2



a)
```python
def sigmoid(actual_output):
    x = 1.0/(1 + np.exp(-actual_output))
    return x
```

b)
```python
def logistic_hypothesis(theta, bias,X):
    yhat = sigmoid(linear_func(X,theta, bias))
    return yhat
```

In order to find the best coefficients (thetas) for our function, these have been initialized with random values. In order to tell how well the algorithm is performing with thetas, a cost function is required (**Figure 3**). Loss is calculated for each single training example whereas cost is the average losses over entire training set.

```
#cost function
def cost_func(X, y, bias, theta):

    # cost when y = 1
    # -(1)*log(hypothesis) #therefore -log(hypothesis)
    y_1 = np.multiply(-y,np.log(logistic_hypothesis(theta, bias, X)))

    # cost when y = 0
    # (1-0)*log(0-hypothesis) #therefore -log(1- hypothesis)
    y_2 = np.multiply((1 - y), np.log(1 - logistic_hypothesis(theta, bias, X)))


    cost = np.sum(y_1 - y_2) / m
    return cost
```

Figure 3

In order to fit the parameters, need to find the theta which minimizes the cost function. Batch gradient descent algorithm (**Figure 4**) was implemented to do so, where the gradient is computed using the whole dataset. Parameters are updated repeatedly using a learning rate.

```
#Batch Gradient Descent function
def BGD(theta, X, y, bias, lr):
    gradient = 0
    bias1 = 0
    for i in range(0, m):

        gradient += (logistic_hypothesis(theta, bias, X[i]) - y[i])*X[i]
        bias1 += (logistic_hypothesis(theta, bias, X[i]) - y[i])


    theta_New = theta - lr *(1/m) * gradient
    bias_New = bias - lr *(1/m) * bias1

    return theta_New , bias_New
```

Figure 4

Finally, a logistic regression function was defined, which would call the batch gradient descent algorithm, the cost function and save the loss over each iteration in the cost_list array. (**Figure 4)**

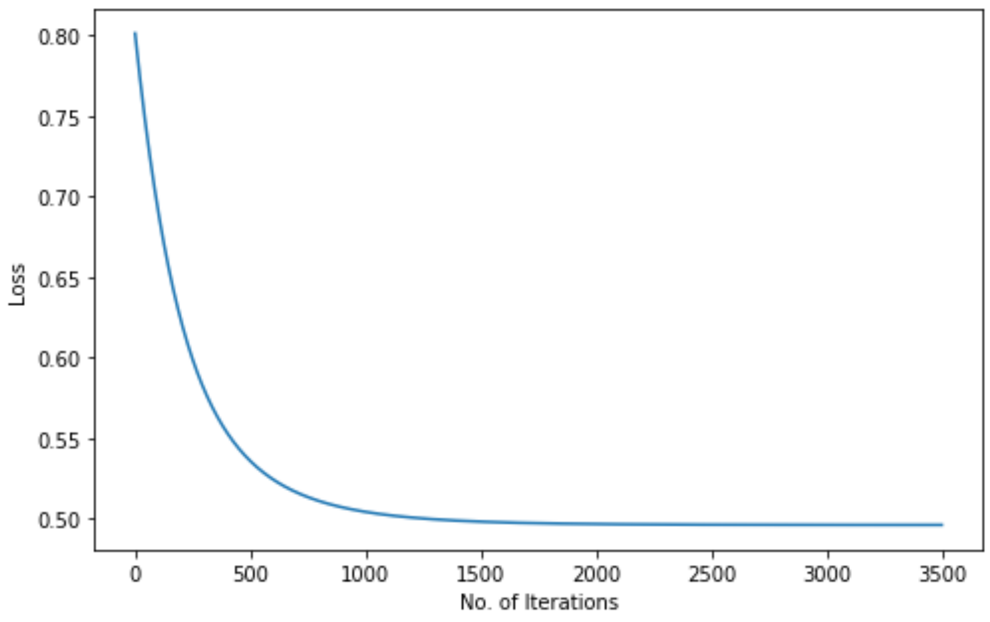
```
def logistic_regression(theta, X, y, bias, lr, iterations):

    #create an empty array to allocate a cost list
    cost_list = []

    for i in range(1, iterations+1):

        #update the theta and bias at each iteration using the Batch Gradient Descent function
        theta, bias = BGD (theta, X, y, bias, lr)
        #Calculate loss at each iteration
        loss = cost_func(X, y, bias, theta)
        #Update the list
        cost_list.append(loss)
        print ('Iteration: ',str(i),'--', 'Cost: ', str(loss),'--', 'Theta updated: ',str(theta))

    return cost_list
```

Figure 4

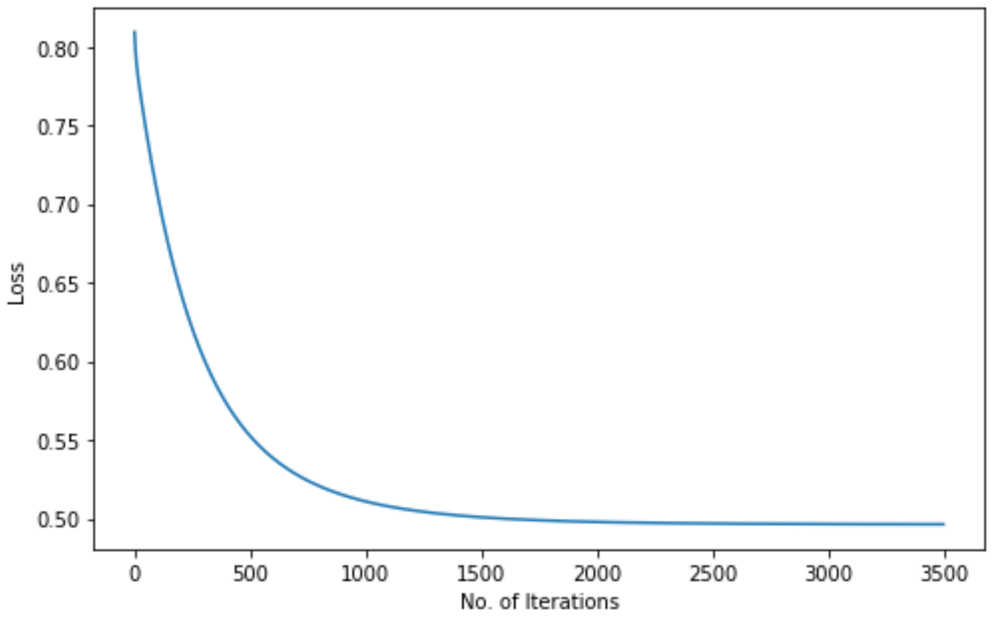
2. [question] After how many iterations, and for which learning rate (α) did your algorithm converge? Plot the loss function with respect to iterations to illustrate this point.

Algorithm converged after **3500 iterations** with a learning rate, **α = 0.04.**
Values for theta were randomly generated with **theta = np.random.uniform(-1,1,1)**;
The initial value of theta was **theta = -0.11961114** for this case.
The optimal theta value when converged is found to be **theta = 1.04212204**
Given the smoothness of the curve, is sign of a good learning rate.

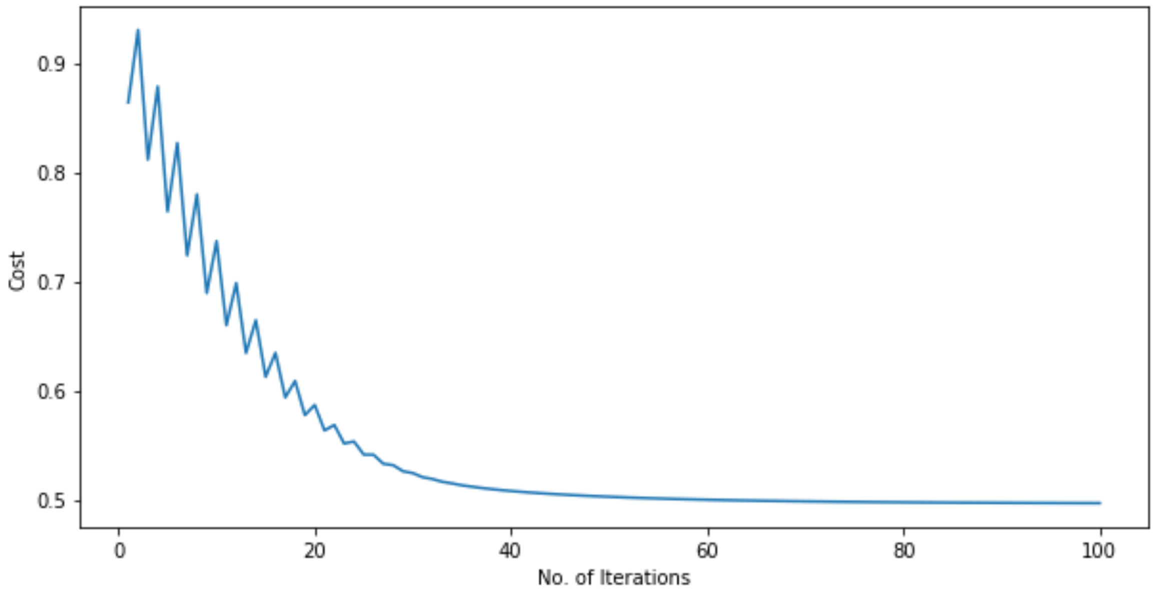
Loss per iteration



Another scenario, with theta value **theta = np.zeros(1)** with the same learning rate **α = 0.04** and 3500 iterations, it was resulted with optimal final theta value to be **theta = 1.0422377.**



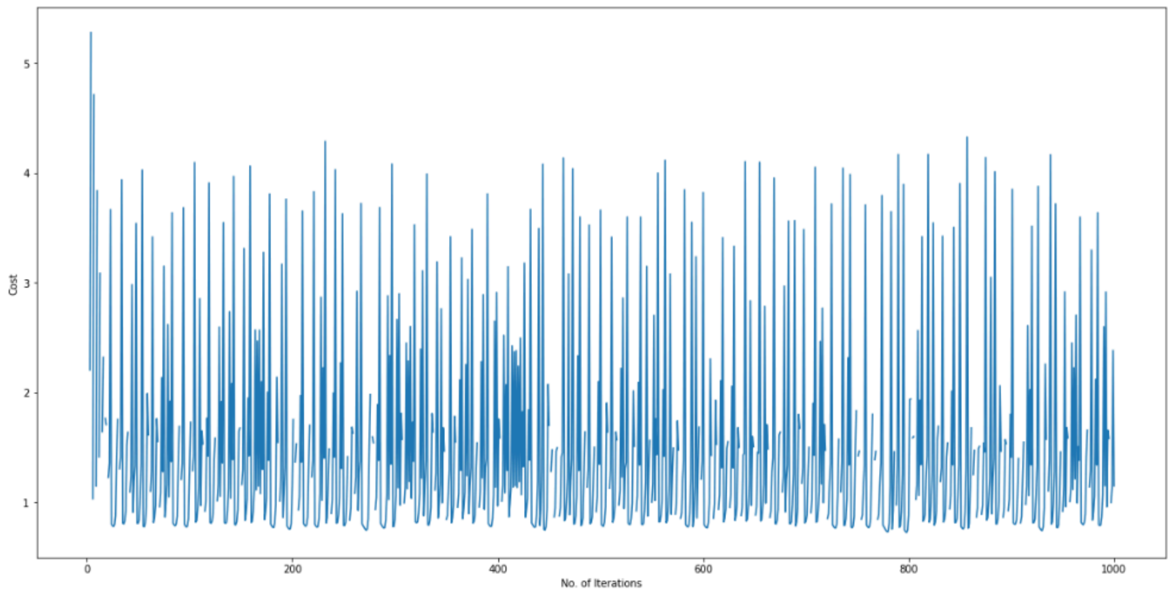
3. [question] What happens if α is too large? How does this affect the loss function? Plot the loss function with respect to iterations to illustrate this point. This point.



Plot 2

The above plot (**Plot2**) of the loss function over iterations, was result of using a higher learning rate, in this case **α = 1**. As the plot shows, the loss increases and decreases inconsistently having a zig-zag shape even though it converges eventually with fewer iterations.

When the learning rate is set to **α = 5**, the loss function is all over the place! (see **Plot 3** below) It is very inconsistent and it fails to converge. This means that is overshooting the minimum and never finding it, because of the high learning rate.



Plot 3

4. [question] Assume that you are applying logistic regression to the iris (flower) dataset, as in the previous assignment. Answer the following questions:

(a) How would your hypothesis function change in this case and why?

Logistic regression for this coursework we assumed labels were only binary $\mathbf{y^{(i)}} \in$ **{0,1}**, only 2 classes. With the iris flower dataset however we have to deal with 3 different classes.
For iris dataset, in the training set we'd have $\mathbf{\{(x^{(1)}, y^{(1)}),\ldots,(x^{(m)}, y^{(m)})\}}$ and $\mathbf{y^{(i)} \in \{1,\ldots,K\}}$, where K is the number of classes, in this case 3.

Given a new input **x**, we want our hypothesis to estimate the probability that **P(y = k | x; θ)** for each value of **k = 1,…,K**. Basically we want to estimate what is the probability that **y** equals to class ***i,*** given **x** parameterized by **θ**.
The hypothesis function is therefore as follow [2]

$$h_\theta(x) = \begin{bmatrix} P(y=1|x;\theta) \\ P(y=2|x;\theta) \\ \vdots \\ P(y=K|x;\theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{K} \exp(\theta^{(j)\top} x)} \begin{bmatrix} \exp(\theta^{(1)\top} x) \\ \exp(\theta^{(2)\top} x) \\ \vdots \\ \exp(\theta^{(K)\top} x) \end{bmatrix}$$

[2]

(b) How would you utilize your implementation of logistic regression in order to perform (multi-class) classification on the iris dataset? Include some pseudocode while discussing your approach

Since we already know how to do binary classification using our logistic regression, we can make it work for multiclass classification using One vs all (one vs rest) classification.

We need to turn this into 3 separate two class classification problems.

First with class1, which is of Iris Setosa, need to create a new temporary training set, where both class 2 and 3 are assigned as negative class and class 1 as positive.
Then train standard logistic regression classifier, need to fit the classifier $\boldsymbol{h_\theta^{1}(x)}$ where superscript 1 is for class 1.

Then with class 2, which is Iris Versicolor, we create a new temporary training set, where class 3 and 1 are assigned as negative and class 2 as positive.
Train the logistic regression classifier, to fit the $\boldsymbol{h_\theta^{2}(x)}$

Then with class 3, which is Virginica, we create a new temporary training set, where class 2 and 1 are assigned as negative and class 3 as positive.
Train the logistic regression classifier, to fit the $\boldsymbol{h_\theta^{3}(x)}$

To make a prediction in a new input X, we need to run all the 3 classifiers, $\boldsymbol{h_\theta^{1}(x)}$, $\boldsymbol{h_\theta^{2}(x)}$, $\boldsymbol{h_\theta^{3}(x)}$ on input X and pick the class which gives the highest prediction. [3]

References:
[1]https://doc.lagout.org/science/Artificial%20Intelligence/Machine%20learning/Machine%20Learning_%20A%20Probabilistic%20Perspective%20%5BMurphy%202012-08-24%5D.pdf
[2] http://deeplearning.stanford.edu/tutorial/supervised/SoftmaxRegression/
[3] http://www.holehouse.org/mlclass/06_Logistic_Regression.html