University of Skövde
School of Informatics
Master of Data science
Haftamu Hailu Tefera
al17hafte@student.his.e

Data Mining - IT726A

Assignment 1: Classification of handwritten digits

The central objective of this assignment is to demonstrate how Logistic Regression is performed an implemented using python for classifying hand written digits.

Step 1: The sigmoid function

From the class lecture notes the sigmoid function for a logistic regression is given by

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

To implement this function I have used the following python code where z is replaced by x here

```
def sigmoid(x):
 return (1 / (1 + np.exp(-x)))
```

The output of this sigmoid is always between 0 and 1.

Step 2: Feature normalization:

To normalize the features of an input data set X to have mean 0 and standard deviation 1.This simplifies learning task our model by making the data to have the same scale. I solved this one using the following two lines of code.

```
def featureNormalization(X):
   mean = np.mean(X,0)
   std = np.std(X,0)
   X = (X-mean)/std
   return X
```

I used the numpy package and mean and standard deviation (std) built in function in order to scale the value of the different data sets to have mean 0 and standard deviation 1.

Step 3: Cost function

The cost function for a binary logistic regression for the whole datasets is given as follows (directly from the lecture 3)

$$cost(\hat{Y}, Y) = \frac{1}{m}\sum_{i=0}^{m} cost(\hat{y}_i, y_i)$$
$$= \frac{1}{m}\sum_{i=0}^{m} -log(\hat{y}_i) * y_i - log(1-\hat{y}_i) * (1-y_i)$$

Where Yhat for binary logistic regression is given by=**sigmoid(X * W +b)** where W is the weight and b is the bias.

```
def cost(X, W, b, Y):
  Yhat = sigmoid(np.dot(X,W) + b)
  cost = (-1.0/np.shape(X)[0]) * np.sum(Y * np.log(Yhat) + (1-Y) * (np.log(1-Yhat)))
  if np.isnan(cost):
    return np.inf
  return cost
```

Here I used the **dot** function of the numpy package in order to find the scalar products of the two matrices X and W. The sum built-in function calculates the overall sum of difference between of the predicted value (Yhat) and actual value of Y over the whole data set.
Then finally I compared whether the calculated cost value is infinity or not. If it is infinity it returns infinity otherwise it returns the calculated cost that we are paying for being wrong in the estimation.

Step 4: Calculating the gradients

In order to optimize the cost function we need to perform a derivation on the cost function with respect to the slope (W) and the bias (b) and results the following formulas.

$$\frac{\partial cost}{\partial W} = \frac{1}{m}\sum_{i=0}^{m} -x_i * (y_i - \hat{y}_i) \qquad \frac{\partial cost}{\partial b} = \frac{1}{m}\sum_{i=0}^{m} -(y_i - \hat{y}_i)$$

Code for the above two gradients:

```
def d_cost(X, W, b, Y):
  Yhat = sigmoid(np.dot(X,W) + b)
  dW = (np.dot(-X.T,(Y -Yhat)))/(1.0*np.shape(X)[0])
  db = (np.sum(-(Y-Yhat)))/(1.0*np.shape(X)[0])
  return (dW, db)
```

The problem is solved using numpy package and dot and sum method to calculate the gradient with respect to W and b. Lastly, the function returns a tuple (dW, db).

Step 5: Making predictions (Classify either to the positive example or negative example)

Here we are predicting the positive examples of the data values (digits) when the value of the sigmoid function is greater than 0.5 and negative example if the value is less than or equal to 0.5.

```
def classify(X, W, b):
    X == sigmoid(np.dot(X,W)+b)
    return [1 if x > 0.5 else 0 for x in X]
```

Here after we test each digit using the sigmoid function. If the value is greater than 0.5 it is classified in the positive class or class 1 otherwise it goes to the negative class which is class 0.

Step 6: Evaluation of the solution (Measuring the performance)

After we trained our model we need to test on the unseen data. So in this case to know the percentage of the correctly classified/predicted values we need to perform summation of over the whole unseen data as follows. This validation is performed based on the outcome of the classify function and added all the positive examples (>0.5) on the test data (Y_test).
Here goes the code.

```
y = classify(X_test ,W,b)
result= (np.sum(y == Y_test)/(1.0*np.shape(Y_test)[0]))
print("Model Accuracy=",round(result,1),'%')
```

And we run the classify_zeros () function I get the percentage of correctly classified samples. The model accuracy value falls from 85% to 92% but the most frequent once are fall from **90%-91.3%** Therefore, we can say the model is good enough to correctly classify digits as it many classification models have similar accuracy/performance range.

Step 7: Finding the best learning rate

Here the main task is to evaluate the performance of the model using different learning rate values and number of iterations in the **gradient_descent** () function. To see the impact of changing learning rate and number of iterations on the performance of the model and I registered performance values as below.

Iteration	α= 50	α= 10	α= 1	α= 0.5	0.1	0.01	0.001
1000	89.7%	90.7%	91.2%	91.8%	89.8%	89.9%	91.5%
2500	90.3 %	91.3 %	91.7%	91.2%	89.9%	90.4%	91.3%

Performance model over Iteration table

Which of the learning rates performed best and why was it the best? Would the result be the same if we run gradient descent for 5000 iterations, 1 billion iterations?

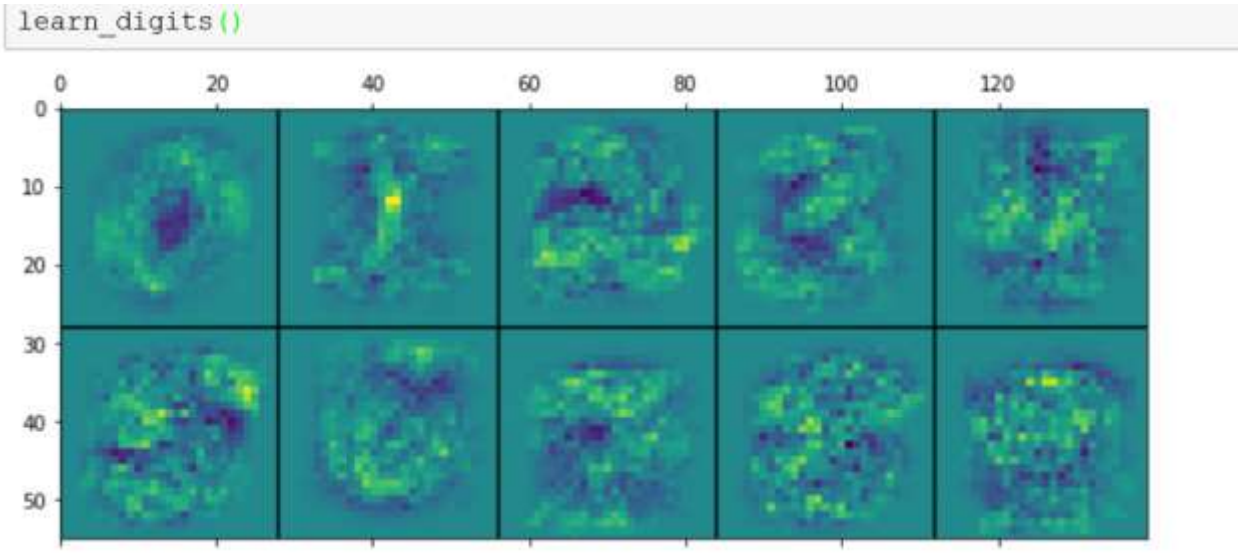
Above I try to register the performance of the model by varying the iteration and learning rate. I only registered performance of the model for 1000 and 2500 iterations because I found it the sigmoid function fits the data points perfectly. Therefore, using large number of iterations on the gradient_descent () that is 5000 and 1b will not bring any change on our performance of our model rather it will add extra cost, i.e. time of computation becomes too high. So for **1b number of iterations** the classify_zeros () function takes long time to display the result on the screen.

From the performance table given above it is obvious there is a slight difference on the performance values of our model by using different learning rates and iterations but in the eye of performance they are in the same category.

In summary, I can conclude if we use small learning rate(alpha) we need many number of iterations to fit the data points on the graph(sigmoid function) and I can say 1000 iterations is enough to fit the data.

Step 8: Learning to classify digits

When I run the learn_digits () function I found the following result.



From the displayed plot we can see that the dark pixels are located on the center of the image on the some of the images especially on the top and bottom left of the plot, this means this dark pixel is unlikely to represent the number and the classifier will have less confidence to put it into the positive class (which is 1).

On the top and bottom right corner of the image the dark pixel are to somehow scattered throughout the image and eventually they moved to peripheral(**I mean the part which is far from the center**) part of the plot . When we analyze the bright pixel it is almost distributed equally on the image on all sections of the plot. So yellow pixels are concentered on the central part of the image the classifier will have good confidence to put this into the positive example or class.

Reference

1. To solve this assignment I have used the lecture notes on logistic regression plus the linear regression code from lecture two.
2. https://www.python.org/
3. https://docs.scipy.org/doc/numpy-dev/user/quickstart.html