machine learning hw2 report

b02902080 資工四 郭傳駿

Logistic Regression Function

- required method

```
def grad(X,Y,L,e):
1.    global _weight,_b
2.    out = expit(np.dot(X,_weight) + _b)
3.    #loss = ((-1)*(np.dot(Y.T,np.log(out))+np.dot((1-
      Y).T,np.log(1-out))).sum()) / X.shape[0]
4.    partial_w = (-1)*np.dot(X.T,Y - out) + L * _weight
5.    partial_b = (-1)*(Y - out).sum()
6.    _weight -= e*partial_w
7.    _b -= e*partial_b
```

- the function takes four parameters
  X: feature input, Y: label, L: lambda, e:eta
- line1: the function takes two global variables
  _weight: weights(numpy array), _b: bias(float)
- line2: compute the logistic output as out
- line3: loss is always Nan in the beginning of training so I commented it
- line4: compute error caused by weight; regularise the weights by L
- line5: compute error caused by bias
- line6: update weights
- line7: update bias

the learning rate is set at 0.1 at first and then decays by 0.9995x after each iteration

---

Probabilistic Generative Model

- for the second method, I implemented the generative model introduced in class (classification)
- I chose **multivariate gaussian** as the probability distribution
- in calculating inverse of the covariance matrix, I encountered problems regarding singular matrix, my solution is to add small floating point numbers along the diagonal (1e-9), then its possible to get the inverse
- for calculating prior probabilities for both class on each data instance, I use the multivariate_normal.pdf() function from scipy.stats
  -> `mvnorm.pdf(X[i,:], spam_miu, sigma, True)`

it takes four parameters:
X[i,:]: input, spam_miu: the mean of spam mail distribution, sigma: shared covariance matrix, option for accepting singular matrix

- by calling the above function it seem to work whether the covariance matrix is singular, but without adding along the diagonal it would still raise error about matrix not positive semi-definite
- the classification is done by comparing prior probability for both classes, and assign the test case to the one with higher probability
- the reason not to calculate posterior probability is to simplify the calculation

---

Comparison between two methods

- Logistic regression preforms generally better than the generative method.
- On the public leaderboard, my logistic approach can get up to 93% correctness, the generative method gets about 87%.
- It is not easy to describe what distribution is the data in, so it might be hard to find the best probabilistic model to use.
- One interesting discovery is that generative model's performance on the public set is quite close to that on private set with 1% difference, while performance of logistic regression dropped more than 3% when switched to private set!

Additional Experiment: Neural Network

Since the logistic regression method has a upper bound of performance of approx. 93% accuracy, I tried to implement a simple neural network which is in essence, a deeper logistic unit.

SEGMENTS OF THE CODE:

```
#X: input, Y: target
    X = matrix[:,:-1]
    Y = matrix[:,-1:]
    X = np.append(X,np.ones((X.shape[0],1)),axis = 1)
    #synapse
    syn0 = 2 * np.random.rand(feature+1,hid) - 1
    syn1 = 2 * np.random.rand(hid,1) - 1
    for i in range(1000000):
        l0 = X
        l1 = non_linear(np.dot(l0,syn0))
        l2 = non_linear(np.dot(l1,syn1))
        l2_error = Y - l2
        if(i%1000) == 0:
            print np.mean(np.abs(l2_error))
        l2_delta = l2_error * non_linear(l2,deriv = True)
        l1_error = l2_delta.dot(syn1.T)
        l1_delta = l1_error * non_linear(l1,deriv = True)
        syn1 += l1.T.dot(l2_delta)
        syn0 += l0.T.dot(l1_delta)
```

```python
def non_linear(x,deriv = False):
    if deriv is True:
        return (x)*(1-x)
    return expit(x)
```

EXPLANATION:

- line1: layer0 is the input
- line2: layer1 is input multiplied by synapse0 weights and passed through activation function(sigmoid in this case)
- line3: layer 2 is value in layer 1 multiplied by synapse1 weights passed to sigmoid
- line4: compute prediction distance between label Y
- line5&6: print loss each 1000 iterations
- line7: weight l2 error with derivatives, reduce error of high confidence predictions
- line8: compute how much each node value in l1 contributed to the error in l2, essentially "backpropagation"
- line9: weight l1 error with derivatives
- line10&11 update synapse0&1 weights

DISCUSSION:

- the number of nodes in the hidden layer (l1) is set to 6
- the neural network has a unstable performance, sometimes it can converge while the error is large. I think the reason is that in computing l1,l2 deltas, the error is weighted by the derivative (slope) of the sigmoid function. This means the error is weighted with a number close to zero also when the prediction is furtherest from the target value, and so the synapse wights only get very little updates.
- In overall, the 3-layer neural network doesn't outperform logistic regression, 91% on public and 87% on private.

參考:

the design of the neural network is referenced from https://iamtrask.github.io/2015/07/12/basic-python-network/