Machine Learning(IDC410) Report: Programming Exercise 2

Rimjhim Goel
Roll No: MS18133

Contents

1 Question 1: Generating the data set **2**

2 Question 2: Logistic regression **2**

3 Question 3: **2**
3.1 Impact of variation of n and θ on logistic regression . . . 2
3.1.1 Variation of θ . . . 2
3.1.2 Variation of n . . . 3
3.2 Derivation of partial derivative of cost function . . . 4

4 Question 4: L1 and L2 regularization **4**
4.1 L1/Lasso regression . . . 5
4.2 L2/Ridge regression . . . 5

5 List of files submitted **6**

1 Question 1: Generating the data set

```python
import numpy as np
from scipy.stats import bernoulli
import matplotlib.pyplot as plt

def gen_data(theta,n,m):
    X = np.random.randn(n,m+1)
    X[:,0]=1      # first column set to 1
    beta = np.random.randn(m+1,1)     # random coefficients
    bern = np.transpose(bernoulli.rvs(size=n,p=theta))  #Bernoulli distribution
    Y = 1/(1+ np.exp(-(np.matmul(X,beta))))

    for i in range(len(Y)):  #classification of Y
        if Y[i]>0.5:
            Y[i] = 1
        else:
            Y[i]= 0
    for i in range(len(Y)):  # flipping according to Bernoulli distribution
        if bern[i]==1:
            Y[i] = 1-Y[i]

    return X,beta,Y
```

2 Question 2: Logistic regression

Log likelihood/ cost function -

$$logC = \Sigma_{n=1}^{N} y_n log(\sigma(\beta^T X_n)) + (1 - y_n) log(1 - \sigma(\beta^T X_n))$$

Gradient-

$$\frac{\partial logC}{\partial \beta} = X^T(Y - Y_{predicted})$$

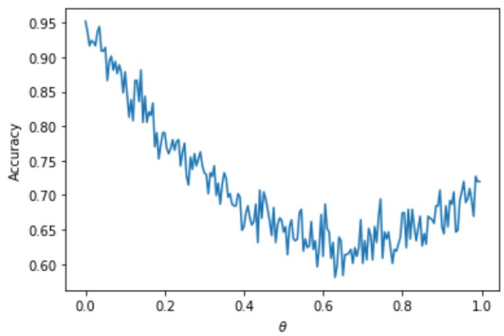
```python
def logistic_regression(X,Y,k,tau,lam):
    z = float(len(X))
    b = np.random.randn(int(len(beta)),1)  #random initialisation of the parameters
    previous_cost = 0   #initial cost set to 0
    cost = np.array([])


    for i in range(k):# k is no. of iterations
        y_pred = 1.0/(1+ np.exp(-(np.matmul(X,b))))# predicted y
        temp = np.subtract(y_pred,Y) # Y-Xb
        temp2 = np.subtract(np.ones_like(Y),Y)
        # Cost function/ Cross Entropy
        current_cost = -(1/z)*np.matmul(np.transpose(Y),np.log(y_pred,out=np.zeros_like(y_pred), where=(y_pred!=0))) - np.matmul(np.transpose(temp2),np.log(temp2,out=np.zeros_like(temp2), where=(temp2!=0)))
        cost = np.append(cost,current_cost)
        if abs(current_cost-previous_cost)<=tau:  # tau - threshold condition on change in cost function
            break

        previous_cost = current_cost
        grad =  (1/z)*(np.matmul(np.transpose(X),temp))# gradient calculation
        b = b - lam*grad #improved b, delta is step size/learning parameter
    return(y_pred,cost[-1])
```

3 Question 3:

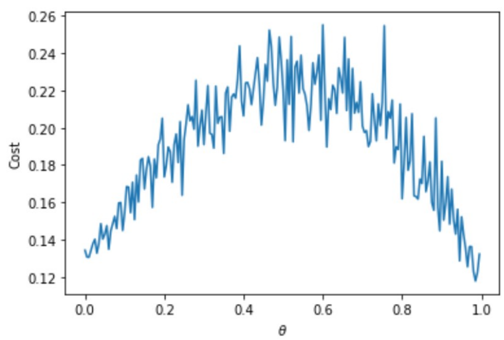
3.1 Impact of variation of n and θ on logistic regression

3.1.1 Variation of θ

For investigating how θ affects the accuracy of the model, other variables(n,m,X,β) were kept fixed for each theta value. $y_{predicted}$ is classified as 1 if probability is greater than 0.5 and 0 otherwise. Accuracy of the



(a) Accuracy vs θ



(b) Cost vs θ

model is calculated as-

$$Accuracy = 1 - \frac{\Sigma_{i=1}^{n}(y_i - y_{predicted_i})}{n}$$

where n is the total no. of elements in the y vector.
Since Bernoulli distribution varies in each iteration, accuracy is averaged over a number of iterations for each θ.

For $\theta \to 0$ and $\theta \to 1$, the cost function is minimum. Accuracy is maximum for $\theta \to 0$ and then starts reducing and is minimum for $\theta \to 0.5$ and then starts increasing again. This is because for $\theta \to 0.5$, the randomness in the data is maximum as probability of flipping and not flipping is equal. For $\theta = 0$ and $\theta = 1$, either none of the values are flipped or all are flipped. This means that the randomness in the data is less as compared to the θ values in between. Hence, the accuracy is maximum for $\theta \to 0$ and then again increases for $\theta \to 1$.

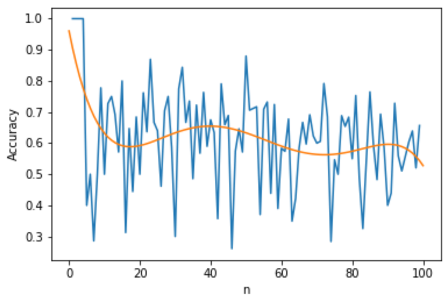
3.1.2 Variation of n

For observing the effect of n on the model, θ was kept zero i.e. no random flipping due to noise. There are lot of fluctuations in the accuracy because of inherent randomness due to random initialisation of β in the logistic regression model. However, the accuracy is averaged over a number of iterations for each n.

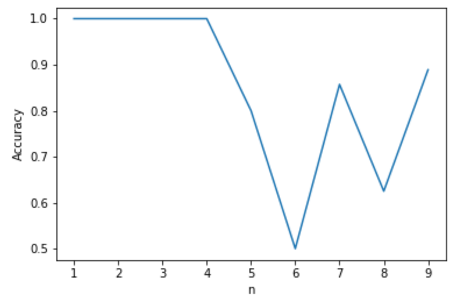
The plot shows accuracy = 1 for very small values of n. This is because it is easier to fit lesser data points. A perfect fit does not imply efficient machine learning. Small testing data sets underestimate the parameters and lead to overfitting. Such algorithms produce poor results when applied on training sets.

Thus an optimum sample size is necessary to represent all the parameters.

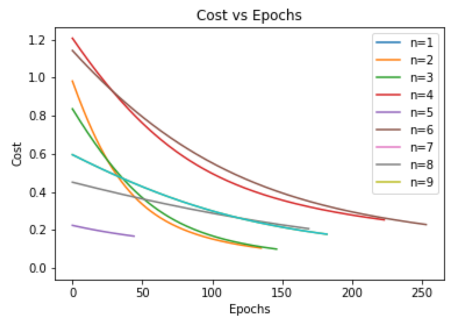
The cost converges to a minimum value for each n but there is no relation between n and the number of epochs it takes to converge.



(a) Accuracy vs n



(b) Accuracy for small n



(c) Cost vs Epochs

3.2 Derivation of partial derivative of cost function

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

Let p_n be the probability of being in a certain class.

$$p_n = \sigma(\beta^T X_n)$$

Maximum likelihood principle:

$$p(y_n) = p_n^{y_n}(1 - p_n)^{(1-y_n)}$$

$$p(y_n) = \sigma(\beta^T X_n)^{y_n}(1 - \sigma(\beta^T X_n)^{(1-y_n)}$$

Likelihood function:

$$C = \Pi_{n=1}^{N} \sigma(\beta^T X_n)^{y_n}(1 - \sigma(\beta^T X_n))^{(1-y_n)}$$

Log likelihood:

$$logC = \Sigma_{n=1}^{N} y_n log(\sigma(\beta^T X_n)) + (1 - y_n) log(1 - \sigma(\beta^T X_n))$$

Maximising the log likelihood principle:

$$\frac{\partial logC}{\partial \beta} = \Sigma_{n=1}^{N} y_n \frac{1}{\sigma(\beta^T X_n)} \frac{\partial \sigma(\beta^T X_n)}{\partial \beta} - (1 - y_n) \frac{1}{1 - \sigma(\beta^T X_n)} \frac{\partial \sigma(\beta^T X_n)}{\beta}$$

$$= \Sigma_{n=1}^{N} y_n (1 - \sigma(\beta^T X_n)) x_n - (1 - y_n) \sigma(\beta^T X_n) x_n$$

$$\frac{\partial logC}{\partial \beta} = X^T (Y - Y_{predicted})$$

4 Question 4: L1 and L2 regularization

Regularization is used to avoid overfitting of the data by adding additional terms to the cost function. These additional terms shrink the estimated parameters towards zero. Regularisation is essentially adding bias to the model weights so that it does not overfit.

4.1 L1/Lasso regression

L1 norm is added

$$\|\beta\| = |\beta_1| + |\beta_2| + \cdots + |\beta_{n+1}|$$

$$Cost = Cost + \lambda\Sigma_{d=1}^{m+1}|\beta_d|$$

$$\frac{\partial Cost}{\partial \beta} = -X^T(Y - X\beta) + \lambda I^{'} sign(\beta)$$

where $I^{'}$ is an identity matrix of size $m+1$ and the first element is zero (since β_0 is not penalised) and λ is the tuning parameter which represents the amount of regularisation. Larger the value of λ, greater the penalty on the parameter estimates which shrinks the estimates towards zero.
L1 regularisation reduces the number of features (β) by eliminating not-so-important features. This is used with data sets having large number of features to reduce complexity.

```python
#lasso regression L1
def logistic_regression_L1(X,Y,k,tau,lam,alpha):
    z = float(len(X))
    b = np.random.randn(int(len(beta)),1)  #random initialisation of the parameters
    previous_cost = 0   #initial cost set to 0
    cost = np.array([])

    b_L2 = 0

    for i in range(1,len(b)):
        b_L2 += abs(b[i])  # L1 norm

    I_L2 = np.ones((len(beta),1))
    I_L2[:,0]=0   # I' matrix
    I_L2 = I_L2*np.sign(b)

    for i in range(k):# k is no. of iterations

        y_pred = 1.0/(1+ np.exp(-(np.matmul(X,b))))# predicted y

        temp = np.subtract(y_pred,Y) # Y-Xb
        temp2 = np.subtract(np.ones_like(Y),Y)
        # cost/cross entropy calculation
        current_cost = -(1/z)*np.matmul(np.transpose(Y),np.log(y_pred,out=np.
    zeros_like(y_pred), where=(y_pred!=0))) - np.matmul(np.transpose(temp2),np.log(
    temp2,out=np.zeros_like(temp2), where=(temp2!=0)))
        # for L1 regression
        current_cost = current_cost + (1/z)*(alpha*b_L2)
        cost = np.append(cost,current_cost)

        if abs(current_cost-previous_cost)<=tau:  # tau - threshold condition on
    change in cost function
            break
        previous_cost = current_cost

        grad =  (1/z)*(np.matmul(np.transpose(X),temp))# gradient calculation
        # gradient calculation for L1 regression, alpha is tuning parameter
        grad = grad - alpha*I_L2
        b = b - lam*grad #improved b, delta is step size/learning parameter
    return(y_pred,cost[-1])
```

4.2 L2/Ridge regression

square of L2 norm is added

$$\|\beta\|^2 = |\beta_1^2| + |\beta_2^2| + \cdots |\beta_{n+1}^2|$$

$$Cost = Cost + \frac{\lambda}{2}\Sigma_{d=1}^{m+1}|\beta_d^2|$$

$$\frac{\partial Cost}{\partial \beta} = -X^T(Y - X\beta) + \lambda I^{'}\beta$$

L2 regularisation does not eliminate β, it just makes their contribution smaller. This is used when the parameters are highly correlated.

```
#ridge regression L2

def logistic_regression_L2(X,Y,k,tau,lam,alpha):
    z = float(len(X))
    b = np.random.randn(int(len(beta)),1)  #random initialisation of the parameters
    previous_cost = 0   #initial cost set to 0
    cost = np.array([])
    iter_ =[]

    b_L2 = 0
    for i in range(1,len(b)):
        b_L2 += b[i]*b[i]   #square of L2 norm

    I_L2 = np.ones((len(beta),1))
    I_L2[:,0]=0     #I'matrix
    I_L2 = I_L2*b

    for i in range(k):# k is no. of iterations
        iter_.append(i)
        y_pred = 1.0/(1+ np.exp(-(np.matmul(X,b))))# predicted y
        temp = np.subtract(y_pred,Y) # Y-Xb
        temp2 = np.subtract(np.ones_like(Y),Y)
        # cost/cross entropy calculation
        current_cost = -(1/z)*np.matmul(np.transpose(Y),np.log(y_pred,out=np.zeros_like(y_pred), where=(y_pred!=0))) - np.matmul(np.transpose(temp2),np.log(temp2,out=np.zeros_like(temp2), where=(temp2!=0)))
        current_cost = current_cost + (1/z)*(alpha*b_L2)  #cost with L2 term
        cost = np.append(cost,current_cost)
        if abs(current_cost-previous_cost)<=tau:  # tau - threshold condition on change in cost function
            break
        previous_cost = current_cost
        #gradient calculation
        grad =  (1/z)*(np.matmul(np.transpose(X),temp))# gradient calculation
        # gradient for L2 regression, alpha is tuning parameter
        grad = grad - alpha*I_L2
        b = b - lam*grad #improved b, delta is step size/learning parameter
    return(y_pred,cost[-1])
```

.

5 List of files submitted

1. .ipynb notebook containing code
2. pdf report