AMS 598 Project 4 Report

Brandon D'Anna
108470479
November 15, 2022

1. Introduction

The scope of this project was to use the ADMM algorithm to run a logistic regression on a large dataset in order to obtain one set of consensus estimates for all coefficients. There were 10 data partition files, each containing the dependent variable y, and 25 explanatory variables. 10 processes were used, allowing for one file to be processed on one process. Gradient descent was used to obtain the logistic regression function. The output of the gradient descent logistic regression algorithm was tested locally for the project4_data_part1.csv data partition and converged in ~3,500 iterations. These results were matched with those from sklearn.

2. Methods

2.1 Functions

8 functions were created:

1. **chunks** - this function divides the total number of files in the directory into equal parts to be scattered to the various processes.

In [ ]:
```
def chunks(files, p):
    size = int(len(files)/p)
    for i in range(0, len(files), size):
        yield files[i:i + size]
```

1. **indvar and depvar** - these two functions took the inputted data file and returned numpy arrays. The former dropping the dependent varaible and adding in a column for the intercept and the latter keeping only the dependent variable.

In [2]:
```
def indvar(file):
    #remove dependent variable column from matrix and add in intercept column. transforms to numpy array
    df = pd.read_csv(file).drop(columns=['y'])
    df.insert(loc=0, column='x0', value=1)
    X = df.to_numpy()
    return X
def depvar(file):
    #keep only the dependent variable as a numpy array
    df = pd.read_csv(file,usecols=['y'])
    y = df['y'].ravel()[:, np.newaxis]
    return y
```

1. **logit and h_X** - the logit function simply takes the linear form of the estimates and puts them into the logit form in order to keep the outputted values between 0 and 1. h_X returns the probability by taking as inputs the dependent variables and their respective coefficients

In [ ]:
```
def logit(X):
    # function for binary response variable
    return 1 / (1 + np.exp(-X))
def h_X(beta, X):
    # returns the logit of the dot product of the weights and the features
    return logit(np.dot(X,beta))
```

1. **costfunc** - this funciton serves as the cost function needed to run the gradient descent algorithm. This is the function that, with the addition of the quadratic term composed of beta - betabar + u, is to be minimized. When the cost function converges, the optimal value of beta, the vector of coefficients, has been achieved.

In [ ]:
```
def costfunc(beta,betabar,u,rho,X,y):
    # returns the cost function for h(x)
    m = X.shape[0]
    ss = beta-betabar+u
    cost = -(1 / m) * (np.sum(y * np.log(h_X(beta,X)) + (1 - y) * np.log(1 - h_X(beta,X))) + (rho/2) * np.dot(ss.T,ss).item())
    return cost
```

1. **gradient** - this function takes the derivative at each beta for each iteration of the logistic regression.

In [ ]:
```
def gradient(beta,X,y):
    # returns the gradient of the cost function at each beta
    m = X.shape[0]
    return (1 / m) * np.dot(X.T, h_X(beta,X) - y)
```

1. **LogisticRegressionGD** - this function takes all of the previously definied component functions to create the full gradient descent algorithm. Here, beta and betabar are initialized as zero vectors if no other values are passed. This allows for an initial run and then subsequent runs will be performed in a while loop with a maximum number of iterations set. These subsequent runs will pass as input parameters the beta and betabar vectors outputted by the initial run.

In [ ]:
```
def LogisticRegressionGD(u,rho,X,y,learningrate,beta=None,betabar=None):

    if type(beta) != np.ndarray:
        beta = np.zeros((X.shape[1], 1))
        betabar = np.zeros((X.shape[1], 1))
    else:
        beta
        betabar

    cost_lst = []
    lastcost = None
    i = 0
    while True:
        gradients = gradient(beta,X,y)
        beta = beta - learningrate * gradients
        cost = costfunc(beta,betabar,u,rho,X,y)
        cost_lst.append(cost)
        i+=1
        if lastcost is not None and math.isclose(cost,lastcost):
            break
        elif i == 25000:
            break
            print('did not converge')
        lastcost = cost

    return beta
```

2.2 MPI Implementation

To implement the MPI, the **scatter()** function was used to distribute the files to the various worker processes from the root process at which the list of files was defined. At each process, the X and y vectors are defined alongside an initalization of the u vector. An initial pass of the gradient descent logistic regression is then run. Next, the **gather()** function is called to produce a list of beta values in the root process. Here, the mean of this list is calculated. A while loop is then implemented, starting with the broadcast of the consensus beta vector back to each process using the **bcast()** function. The u vectors are then updated and the logistic regression is run again. This repeats until each beta is equal to the betabar vector, or 10,000 iterations have been performed. This entire sequence is contained in a **run()** function which is called at the end of the python file.

3. Results

The final model is as follows:

$$= \frac{1}{1+exp(-1+-0.003X_1+.100X_2+-0.047X_3+0.007X_4+-0.305X_5+-0.002X_6+-0.00X_7+-0.005X_8+-0.000X_9+0.000X_{10}+0.002X_{11}+-0.001X_{12}+-0.002X_{13}+0.001X_{14}+-0.008X_{15}+-0.000X_{16}+-0.003X_{17}+-0.002X_{18}+-0.003X_{19}+0.007X_{20}+-0.001X_{21}+0.004X_{22}+-0.205X_{23}+0.205X_{24}+0.004X_{25})}$$

The β consensus vector is:

In [5]:
```
beta = [-1.002851064556674388e+00,
-3.370342983147195295e-03,
9.997711768048508918e-02,
-4.724346112246371326e-02,
6.597856814066970117e-03,
3.054461636567274985e-01,
-1.728896864096721858e-03,
3.710434519337973969e-04,
-4.712331100400225618e-03,
-4.954691505511417664e-04,
5.342229105142956829e-04,
2.226930676245514242e-03,
-1.163750491618185685e-03,
-1.947552321643868187e-03,
1.102127899471727623e-03,
-7.569304609748856230e-03,
-3.838548363332413638e-04,
-3.378737783898703474e-03,
-2.309068225499838478e-03,
-2.586880400289350880e-03,
7.109789115878469186e-03,
-1.118603462698329002e-03,
4.334800022412856058e-03,
-2.045952719687072485e-01,
2.054525608309289986e-01,
3.842990277740705122e-03]
```