University of Moratuwa, Sri Lanka

Faculty of Engineering

Department of Electronic and Telecommunication Engineering

Semester 5

EN3150 – Pattern recognition

Assignment 02

Learning from data and related challenges and classification

G.K.M.I.D.Rajarathna

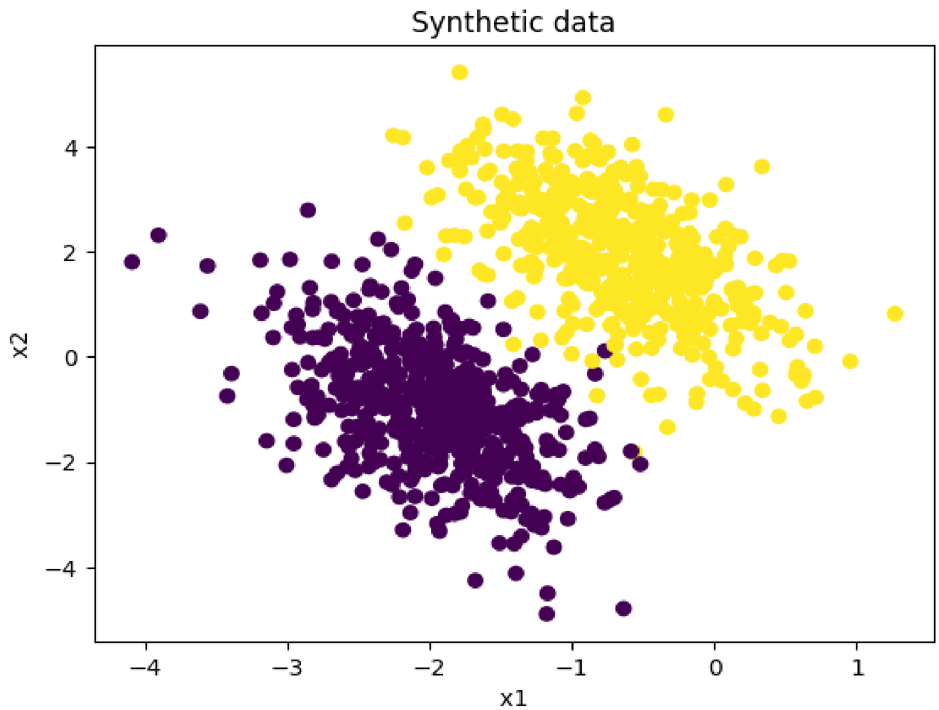
200500L

Logistic regression weight update process

Question 01

Data Generation and visualization

Used given code to generate the data set and visualize the generated data set. In this data set there are two target values and two features.



Question 02

Weight Update (Batch Gradient Descent)

Use batch gradient descent to iteratively optimize the model parameters $w0$, $w1$, and $w2$ in order to minimize the binary cross-entropy cost function with respect to the provided data. This approach allows us to find the optimal parameters through iterative updates.

Batch gradient decent method in general,

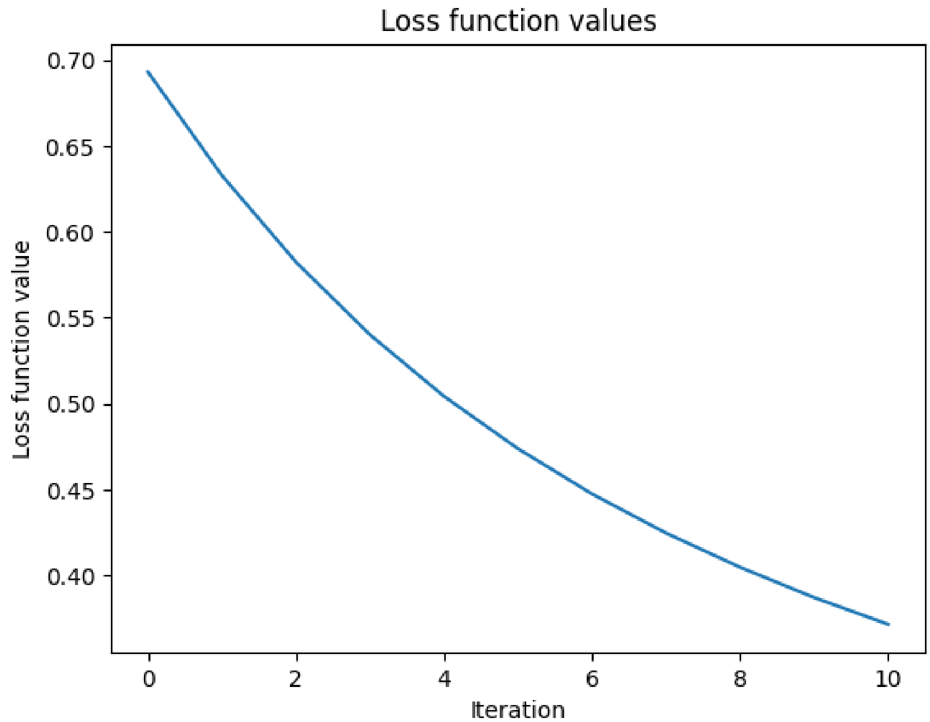
$$\boldsymbol{w}_{(t+1)} \leftarrow \boldsymbol{w}_{(t)} - \alpha \frac{1}{N} \left(\mathrm{sigm}(\boldsymbol{w}_{(t)}^{T}\boldsymbol{X}) - \boldsymbol{y}\right)\boldsymbol{X}.$$

Here, $\boldsymbol{X}$ is data matrix of dimension of $N \times (D+1)$. Here, N is total number of data samples and D is number of features. Now, $\boldsymbol{X}$ is given by

$$\boldsymbol{X} = \begin{bmatrix} x_{1,1} & x_{2,1} & \cdots & x_{D,1} \\ x_{1,2} & x_{2,2} & \cdots & x_{D,2} \\ \vdots & \vdots & \ddots & \vdots \\ x_{1,i} & x_{2,i} & \cdots & x_{D,i} \\ \vdots & \vdots & \ddots & \vdots \\ x_{1,N} & x_{2,N} & \cdots & x_{D,N} \end{bmatrix} = \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_i \\ \vdots \\ \boldsymbol{x}_N \end{bmatrix}.$$

Question 03

Now apply above gradient decent for 10 iterations with the generated data set.



Question 04

Weight Update Using Newton's Method

Gradient descent is a first-order optimization method. So it may converge slowly. To expedite convergence, we can include second-order information for the optimization (**Newton's Method)**. But this may lead to faster convergence, it may also introduce higher computational demands.

Newton's method,

$$\boldsymbol{w}_{(t+1)} \leftarrow \boldsymbol{w}_{(t)} - \left(\frac{1}{N}\,\boldsymbol{X}^T\boldsymbol{S}\boldsymbol{X}\right)^{-1}\left(\frac{1}{N}\,(\mathrm{sigm}(\boldsymbol{w}_{(t)}^T\boldsymbol{X}) - \boldsymbol{y})\boldsymbol{X}\right).$$

and is $\boldsymbol{S}$ given by

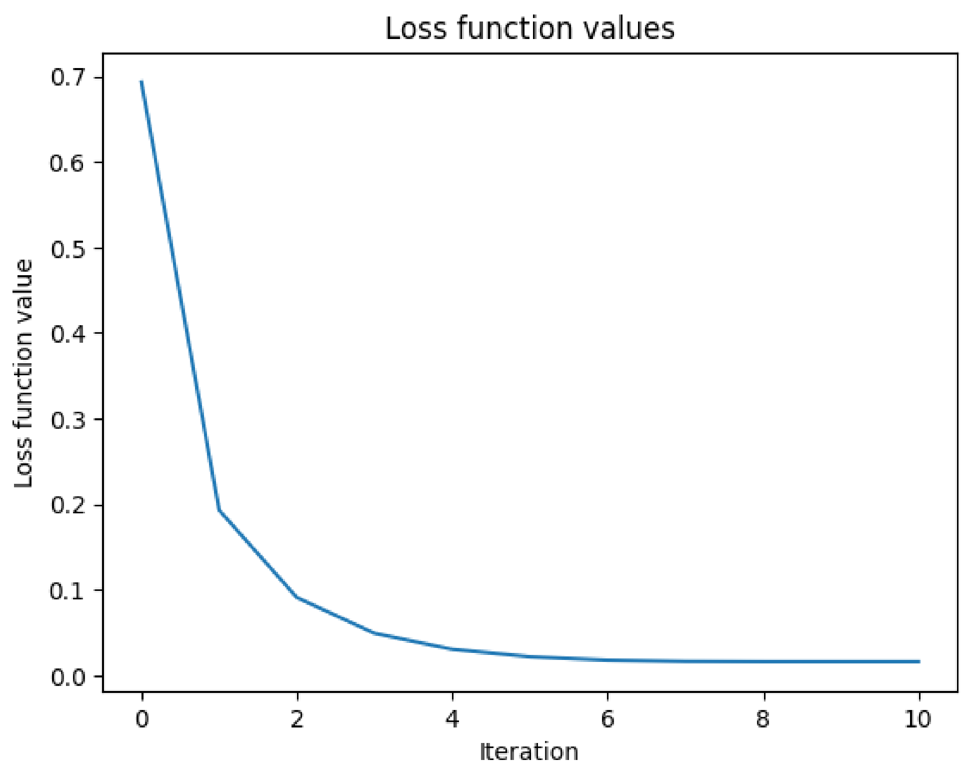
$$\boldsymbol{S} = \mathrm{diag}(s_1,\ s_2, \ldots, s_N),$$
$$\boldsymbol{s_i} = \left(\mathrm{sigm}(\boldsymbol{w}_{(t)}^T\boldsymbol{x}_i) - \boldsymbol{y}_i\right)\left(1 - \mathrm{sigm}(\boldsymbol{w}_{(t)}^T\boldsymbol{x}_i) - \boldsymbol{y}_i\right).$$

Question 05

Python implementation of newton's method

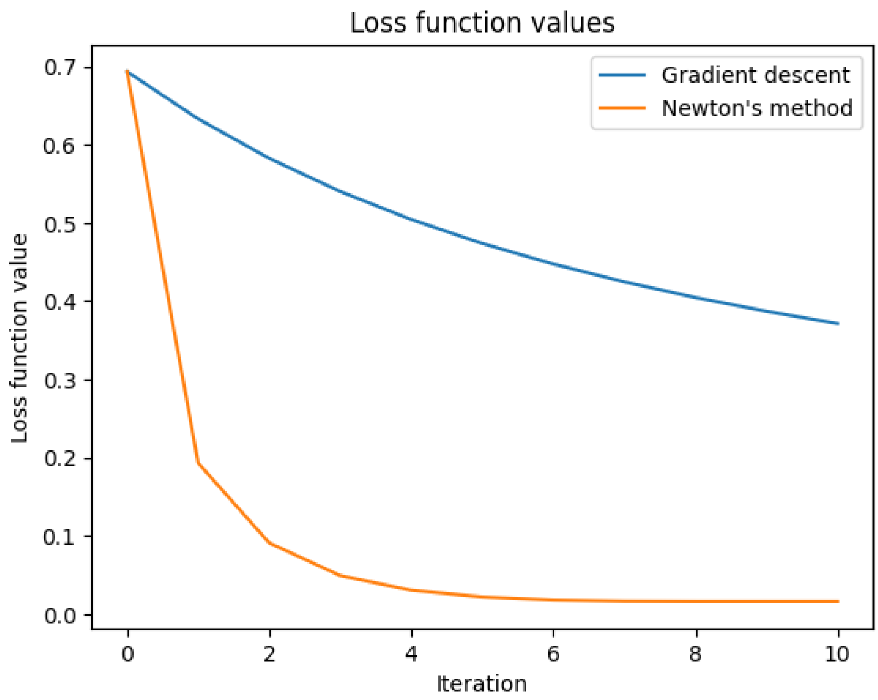
```
# Newton's method loop
for iteration in range(iterations):
    for sample_index in range(N):
        sample_matrix[sample_index, sample_index] = sigmoid(np.dot(weights_newton.T, X[sample_index, :].reshape(D + 1, 1))) - y[sample_index]

    gradient_newton = (ones_array.T @ sample_matrix @ X).T / N

    # Update sample_weights S
    for sample_index in range(N):
        sample_weights[sample_index, sample_index] = (sigmoid(np.dot(weights_newton.T, X[sample_index, :].reshape(D + 1, 1))) - y[sample_index]) * (1 - sigmoid

    # Calculate the Hessian matrix
    Hessian = (X.T @ sample_weights @ X) / N
    weights_newton = weights_newton - np.linalg.inv(Hessian) @ gradient_newton

    # Calculate the cost after the weight update
    updated_cost_newton = 0
    for sample_index in range(N):
        # Calculate the loss for each sample
        sample_loss = log_loss(y[sample_index], sigmoid(np.dot(weights_newton.T, X[sample_index, :].reshape(D + 1, 1)))).item()
        updated_cost_newton += sample_loss
    average_updated_cost_newton = updated_cost_newton / N
    cost_newton.append(average_updated_cost_newton)
```

This method led to relatively faster convergence, iteration results for 10 iteration as bellow,



Question 06

Comparison Between Gradient Descent and Newton's Method



Perform grid search for hyper-parameter tuning

Question 01

loading a dataset containing 7,000 images of handwritten digits, each measuring 28 x 28 pixels. These digits represent numbers from 0 to 9

Question 02

We use X = X[permutation] and y = y[permutation] to shuffle the data set in order to randomize the process.

Question 03

In this problem, we're tasked with multiclass classification using Lasso Logistic Regression, which minimizes the cross-entropy cost function while adding an L1 penalty. To implement it, we use LogisticRegression(penalty='l1', solver='liblinear', multi_class='auto'), adapting automatically for multiclass cases. Lasso is suitable as most data points have many zeros.

Our next step is creating a pipeline, a sequence of data processing steps for a streamlined ML workflow. This pipeline scales data with StandardScaler and employs Lasso Logistic Regression to find the best-fit model

```python
# Use lasso logistic regression and create a pipeline for scaling and classification
model = LogisticRegression(penalty='l1', solver='liblinear', multi_class='auto')
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('classifier', LogisticRegression(penalty='l1', solver='liblinear', multi_class='auto'))
])
param_grid = [
    {'classifier__C': np.logspace(-2, 2, 9)}
]
```

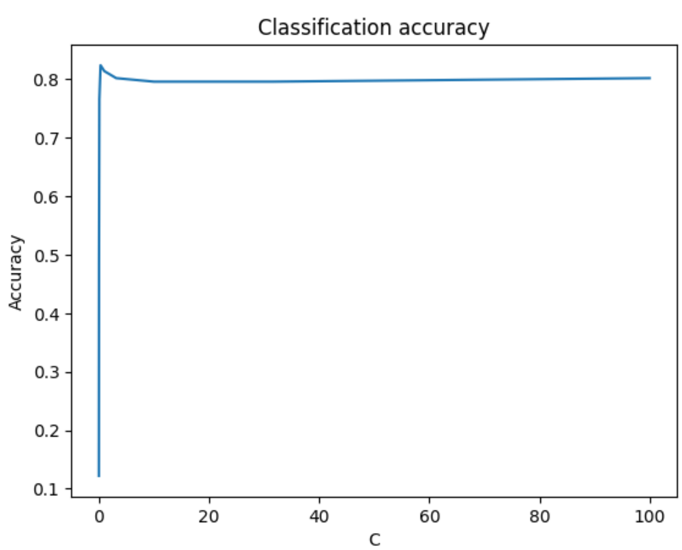
Question 04
Perform Grid Search

Lasso regression involves a hyperparameter (alpha) that influences model accuracy. To optimize this hyperparameter, we use a method called grid search. Grid search systematically tests different hyperparameter values to find the best-performing set, enhancing model accuracy. We'll define a grid of values and use grid search to select the optimal hyperparameter

```python
# Use GridSearchCV to perform a grid search over the range
grid_search = GridSearchCV(pipeline, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# find the best parameters
print("Best parameters: {}".format(grid_search.best_params_))
best_model = grid_search.best_estimator_
best_params = grid_search.best_params_
```

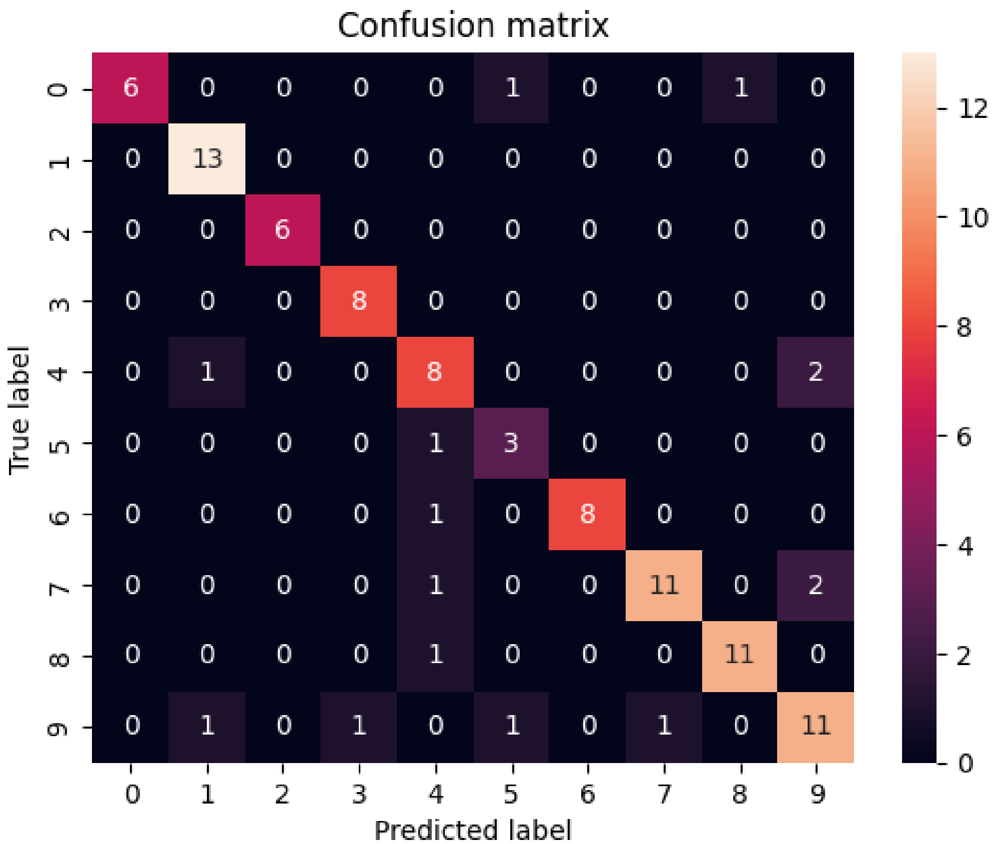
```
Best parameters: {'classifier__C': 0.31622776601683794}
```

Question 05



Question 06

```
Precision: 0.8588888888888888
Recall: 0.85518759018759
F1-score: 0.8526539913624311
```



Consistently high Precision, Recall, and F1-score (>0.8) suggest our model excels in classifying digits with few incorrect predictions. High Precision means few false positives, ensuring accurate predictions. High Recall implies capturing most relevant instances, while a balanced F1-score demonstrates overall model robustness

Logistic regressions

Question 01

$x_1 =$ number of hours studied

$x_2 = undergraduate\ GPA$

$y = 1\ or\ 0 : (1 - Student\ gets\ an\ A+, 0 - Student\ will\ not\ get\ an\ A+)$

$$P(y=1) = \frac{1}{1+e^{-(-6+0.05x_1+x_2)}}$$

a. There is a 37.7 % of probability to get A^+
b. Need 50 hours of study in order to achieve a 50% chance of receiving an A + in the class