AMRITA VISHWA VIDYAAPEETHAM
ETTIMADAI, COIMBATORE

Department - CSE-AI

Course - 21MAT204

Semester - 3

Instructor - Dr.Neethu Mohan

Group Number - 14

Group Members Name	Roll Number
M.Kalyana Sundaram	CB.EN.U4AIE21120
Kaushik Jonnada	CB.EN.U4AIE21122
Praneetha .K	CB.EN.U4AIE21147
Sarvesh ShashiKumar	CB.EN.U4AIE21163
Subikksha	CB.EN.U4AIE21167

PROJECT SUBMITTED FOR THE END SEMESTER EXAMINATION OF 21MAT204

EXTERNAL EXAMINER

INTERNAL EXAMINER

DECLARATION

We hereby declare that this project submitted to the Center for Computational Engineering and Networking is a record of the original work done under the guidance of Dr.Neethu Mohan.

Signature of the faculty

Group Members Name	Roll Number
M.Kalyana Sundaram	CB.EN.U4AIE21120
Kaushik Jonnada	CB.EN.U4AIE21122
Praneetha .K	CB.EN.U4AIE21147
Sarvesh ShashiKumar	CB.EN.U4AIE21163
Subikksha	CB.EN.U4AIE21167

Date:

Place: Ettimadai

ACKNOWLEDGEMENT

We would like to thank our professor Dr.Neethu Mohan who gave us her valuable suggestions and ideas when we needed them to work on our project on Logistic Regression with L1 Regularization

We are also grateful to our university, Amrita Vishwa Vidyapeetham for giving us the opportunity to work on this project. Last but not the least, we would like to thank our group members, as without their cooperation, we would not have been able to complete the project within the prescribed time.

Table Of Contents

Introduction

Logistic Regression is a statistical method used to fit a regression model when the response variable is binary. In logistic regression, the goal is to find the best coefficients, denoted by $\beta_0, \beta_1, ..., \beta_p$, that will separate the positive instances from the negative instances in the training set.

The logistic function, also known as the sigmoid function, is used to model the probability of the positive class given the feature values:

$$P(y=1|x) = \frac{1}{1+e^{-(\beta_0+\beta_1 x_1+\beta_2 x_2+...+\beta_p x_p)}}$$

Where x are the feature values and $\beta_0, \beta_1, ..., \beta_p$ are the coefficients.

To find the best coefficients, we need to maximize the likelihood of the observed data. The likelihood is defined as the probability of the observed data given the model. The negative log-likelihood is used as a cost function to minimize during the optimization process.

The cost function for logistic regression is given by:

$$J(\beta) = -\frac{1}{n}\sum_{i=1}^{n}[y_i log(P(y_i=1|x_i)) + (1-y_i) log(1-P(y_i=1|x_i))]$$

Where n is the number of instances in the training set.

This cost function is optimized using optimization algorithms like gradient descent, Newton-Raphson method or conjugate gradient optimization etc.

Once the optimal values of coefficients are found, the model can be used to predict the probability of the positive class for new instances. A threshold is chosen to convert the predicted probability into a binary output.

In summary, Logistic Regression is a supervised learning algorithm used to classify binary data by fitting a logistic function to the input features and the output variable, and then maximizing the likelihood of the observed data.

Cost Function:

The cost function in logistic regression is used to measure the difference between the predicted probabilities and the actual labels of the data, this is achieved by using a loss function such as the cross-entropy loss function.

The cross-entropy loss function is represented by the following equation:

For y = 1, Loss = $-\log(\hat{y})$

For y = 0, Loss = $-\log(1-\hat{y})$

The cost function is the average of the loss function over the entire training set:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}[-y^{(i)}log(\hat{y}^{(i)}) - (1-y^{(i)})log(1-\hat{y}^{(i)})]$$

The goal of training a logistic regression model is to find the set of weights that minimize the cost function. This is done by using optimization algorithms such as gradient descent to iteratively update the weights in the direction of the steepest decrease in the cost function.

In summary, the cost function in logistic regression is used to measure the difference between the predicted probabilities and the actual labels, it is calculated by an average of cross-entropy loss function over the entire training set, and the goal is to minimize the cost function to get the best set of weights for the model.

L1 Regularization

Logistic Regression is a statistical method that we use to fit a regression model when the response variable is binary. L1 regularization, also known as Lasso regularization, is a method to avoid overfitting by adding a penalty term to the cost function. The penalty term is the absolute value of the coefficients multiplied by a hyperparameter, lambda. This has the effect of shrinking the coefficients towards zero, which in turn can lead to some of the features being completely ignored by the model (i.e., the coefficients become exactly zero).

The L1 regularization term causes some coefficients to become exactly zero. This can be useful when we have a high number of features and we want to select a subset for our model. This method is known as feature selection.

Advantage of L1 regularization is that it is computationally efficient, and can be useful when we have a large number of features and we want to select a subset for our model.

A disadvantage of L1 regularization is that it is not differentiable, which makes it more difficult to optimize the cost function using gradient-based optimization algorithms. Additionally, L1 regularization can lead to unstable solutions and yield models that are difficult to interpret.

Overall, L1 regularization is a useful technique to prevent overfitting and to select a subset of features for a logistic regression model. However, it should be used with caution, as it can lead to unstable solutions and models that are difficult to interpret.

The goal of logistic regression is to find the best coefficients $\beta_0, \beta_1, \ldots, \beta_p$ that will separate the positive instances from the negative instances in the training set.

The logistic function is used to model the probability of the positive class given the feature values:

$$P(y=1|x) = \frac{1}{1+e^{-(\beta_0+\beta_1 x_1+\beta_2 x_2+...+\beta_p x_p)}}$$

Where x is the feature vector, $\beta_0, \beta_1, ..., \beta_p$ are the coefficients and e is the base of the natural logarithm.

To find the best coefficients, we need to maximize the likelihood of the observed data. The likelihood is defined as the probability of the observed data given the model. The negative log-likelihood is used as a cost function to minimize during the optimization process.

The cost function for logistic regression is given by:

$$J(\beta) = -\frac{1}{n}\sum_{i=1}^{n}[y_i log(P(y_i=1|x_i)) + (1-y_i) log(1-P(y_i=1|x_i))]$$

Where n is the number of instances in the training set.

With L1 regularization, we add a penalty term to the cost function, which is the absolute value of the coefficients multiplied by a hyperparameter lambda. This has the effect of shrinking the coefficients towards zero. The cost function for logistic regression with L1 regularization is given by:

$$J(\beta) = -\frac{1}{n}\sum_{i=1}^{n}[y_i log(P(y_i=1|x_i)) + (1-y_i) log(1-P(y_i=1|x_i))] + \lambda\sum_{j=1}^{p}|\beta_j|$$

Where λ is the regularization parameter, and it controls the strength of the regularization. A higher value of λ will result in smaller coefficients.

It is important to note that this cost function is not differentiable at $\beta_j = 0$, so optimization algorithms like gradient descent can not be used directly. Instead, sub-gradient descent is used to optimize the cost function.

L2 Regularization

L2 regularization, also known as Ridge regularization, is a method to avoid overfitting by adding a penalty term to the cost function. The penalty term is the sum of the squares of the coefficients multiplied by a hyperparameter, lambda. This has the effect of shrinking the coefficients towards zero, which in turn can lead to a simpler model.

The L2 regularization term causes the coefficients to be smaller, and it does not force any coefficients to be exactly zero. This can be useful when we want to balance the trade-off between a simpler model and a model that fits the data well.

The cost function for logistic regression with L2 regularization is given by:

$$J(\beta) = -\frac{1}{n}\sum_{i=1}^{n}[y_i log(P(y_i = 1|x_i)) + (1 - y_i)log(1 - P(y_i = 1|x_i))] + \frac{\lambda}{2}\sum_{j=1}^{p}\beta_j^2$$

Where λ is the regularization parameter, and it controls the strength of the regularization. A higher value of λ will result in smaller coefficients.
It is important to note that L2 regularization is differentiable so it can be optimized using optimization algorithms like gradient descent.

As L2 regularization causes the coefficients to be smaller, it can help reduce the variance of the model and prevent overfitting. However, it may not be as effective as L1 regularization in selecting a subset of features for the model.

Forward Propagation in Logistic Regression

In logistic regression, the goal is to learn a set of weights that can be used to make predictions about the probability of a particular input belonging to a certain class. Forward propagation is the process of using these weights to make a prediction for a given input. The following are the main steps involved in forward propagation for logistic regression:

The input features, represented by a column vector \mathbf{x}, are multiplied by the weight matrix \mathbf{W} (also a column vector) to obtain a weighted sum of the inputs:

$$\mathbf{z} = \mathbf{W}^T \mathbf{x}$$

This weighted sum is passed through the sigmoid function, represented by the function

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

to obtain a prediction for the probability that the input belongs to the positive class:

$$\hat{y} = \sigma(\mathbf{z}) = \frac{1}{1 + e^{-\mathbf{W}^T \mathbf{x}}}$$

The final prediction is made by thresholding the probability output obtained above by a certain threshold, usually 0.5.

In this way, Forward propagation in logistic regression is a simple matrix multiplication followed by a non-linear function (sigmoid) to make a prediction of the probability that the input belongs to a certain class.

Backpropagation in Logistic Regression

Backpropagation is an algorithm used to update the coefficients of a model during the training process. It is commonly used in neural networks but can also be applied to logistic regression. Backpropagation works by calculating the gradient of the cost function with respect to the coefficients and using it to update the coefficients in the opposite direction of the gradient.

In logistic regression, the cost function is the negative log-likelihood of the observed data given the model. The gradient of the cost function with respect to the coefficients is given by:

$$\frac{\partial J}{\partial \beta_j} = -\frac{1}{n}\sum_{i=1}^{n}[y_i - \hat{y_i}]x_j$$

Where $\hat{y_i}$ is the predicted probability of the positive class for the i-th instance in the training set, y_i is the actual label, x_j is the j-th feature value for the i-th instance, and n is the number of instances in the training set.

The coefficients are then updated using the gradient:

$$\beta_j = \beta_j - \alpha\frac{\partial J}{\partial \beta_j}$$

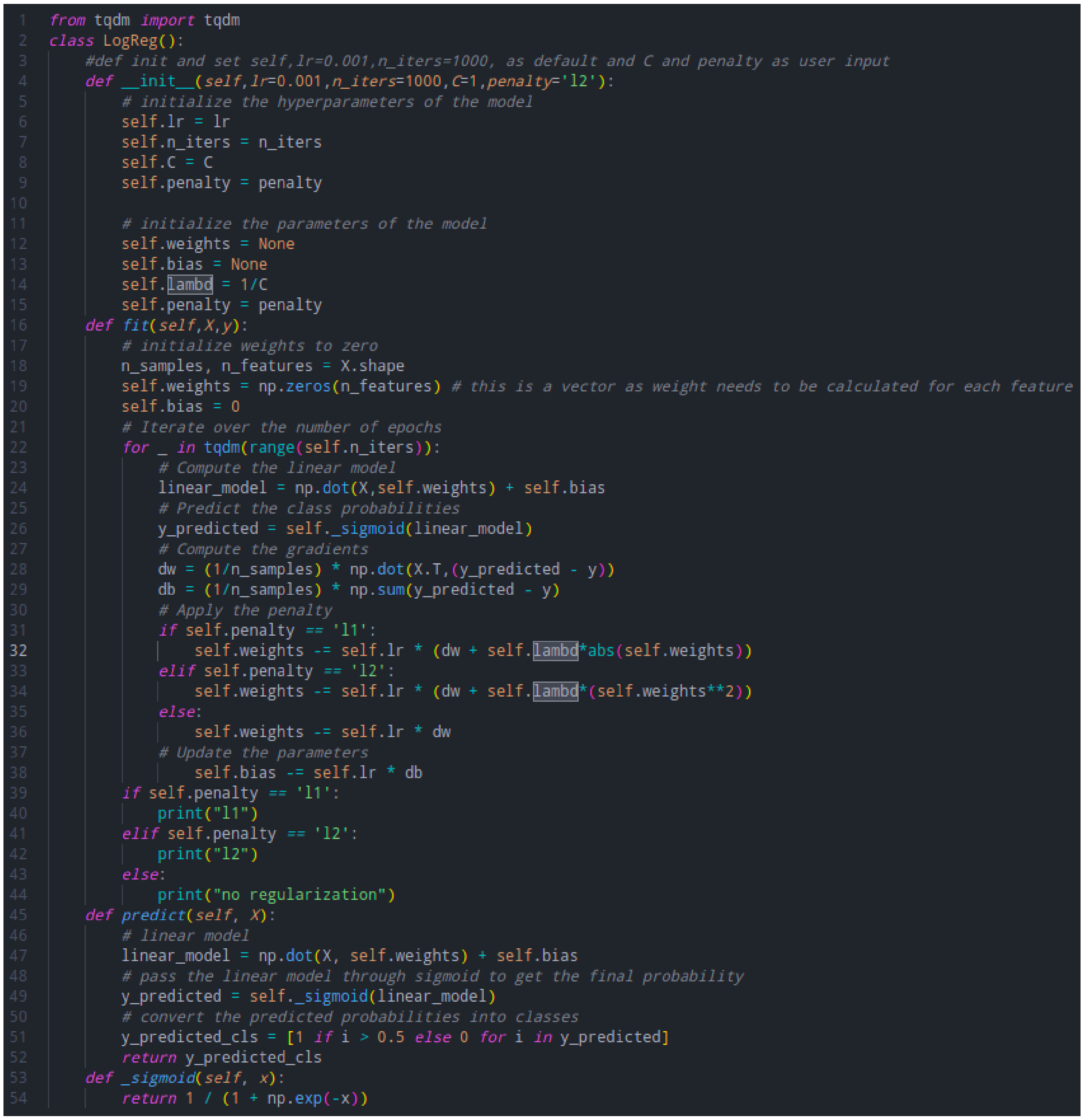
Where α is the learning rate, which controls the step size of the updates.

This process is repeated for a number of iterations until the cost function converges to a minimum.

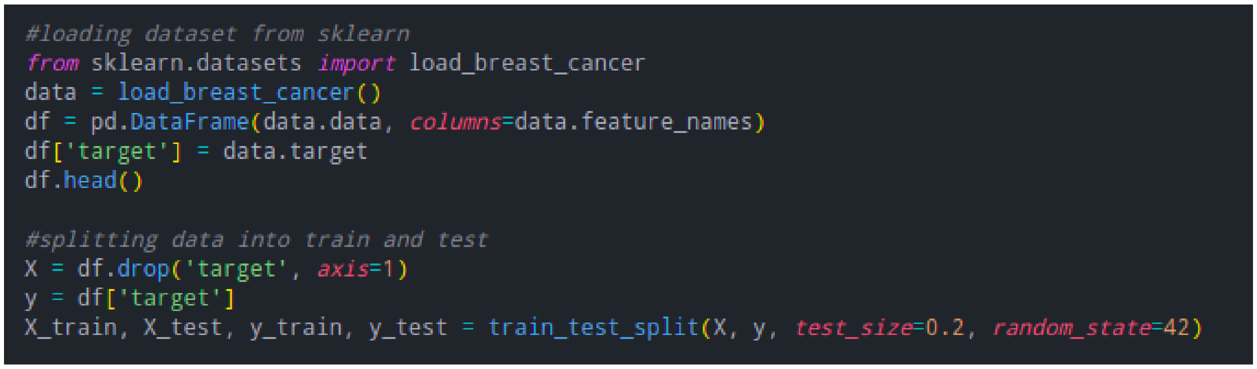
It's important to note that Backpropagation is a supervised learning algorithm, which means it uses labeled data to learn the model's parameters. Backpropagation is mainly used in neural networks but can also be used in logistic regression to optimize the cost function with respect to the coefficients.

Implementation

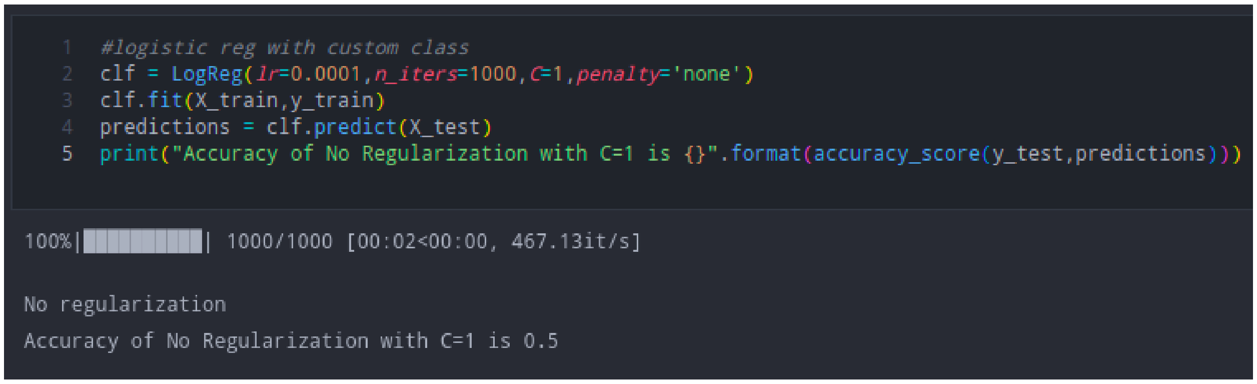
Implementation of Logistic Regression from Scratch:



```python
from tqdm import tqdm
class LogReg():
    #def init and set self,lr=0.001,n_iters=1000, as default and C and penalty as user input
    def __init__(self,lr=0.001,n_iters=1000,C=1,penalty='l2'):
        # initialize the hyperparameters of the model
        self.lr = lr
        self.n_iters = n_iters
        self.C = C
        self.penalty = penalty

        # initialize the parameters of the model
        self.weights = None
        self.bias = None
        self.lambd = 1/C
        self.penalty = penalty
    def fit(self,X,y):
        # initialize weights to zero
        n_samples, n_features = X.shape
        self.weights = np.zeros(n_features) # this is a vector as weight needs to be calculated for each feature
        self.bias = 0
        # Iterate over the number of epochs
        for _ in tqdm(range(self.n_iters)):
            # Compute the linear model
            linear_model = np.dot(X,self.weights) + self.bias
            # Predict the class probabilities
            y_predicted = self._sigmoid(linear_model)
            # Compute the gradients
            dw = (1/n_samples) * np.dot(X.T,(y_predicted - y))
            db = (1/n_samples) * np.sum(y_predicted - y)
            # Apply the penalty
            if self.penalty == 'l1':
                self.weights -= self.lr * (dw + self.lambd*abs(self.weights))
            elif self.penalty == 'l2':
                self.weights -= self.lr * (dw + self.lambd*(self.weights**2))
            else:
                self.weights -= self.lr * dw
            # Update the parameters
                self.bias -= self.lr * db
        if self.penalty == 'l1':
            print("l1")
        elif self.penalty == 'l2':
            print("l2")
        else:
            print("no regularization")
    def predict(self, X):
        # linear model
        linear_model = np.dot(X, self.weights) + self.bias
        # pass the linear model through sigmoid to get the final probability
        y_predicted = self._sigmoid(linear_model)
        # convert the predicted probabilities into classes
        y_predicted_cls = [1 if i > 0.5 else 0 for i in y_predicted]
        return y_predicted_cls
    def _sigmoid(self, x):
        return 1 / (1 + np.exp(-x))
```

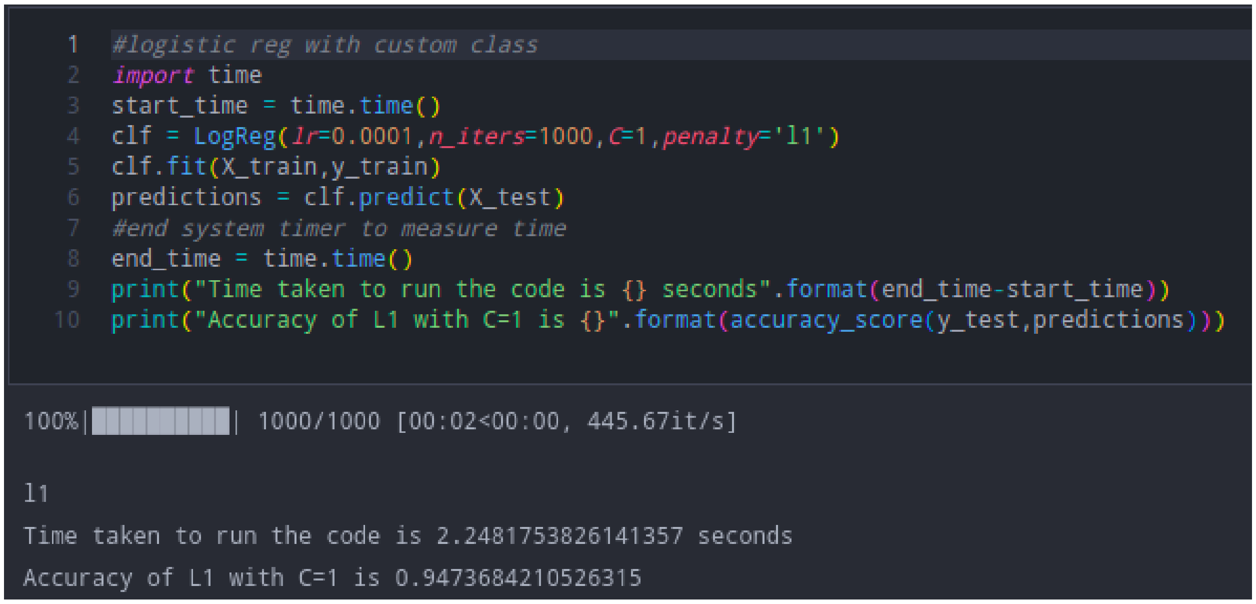


```python
#loading dataset from sklearn
from sklearn.datasets import load_breast_cancer
data = load_breast_cancer()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target
df.head()

#splitting data into train and test
X = df.drop('target', axis=1)
y = df['target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```



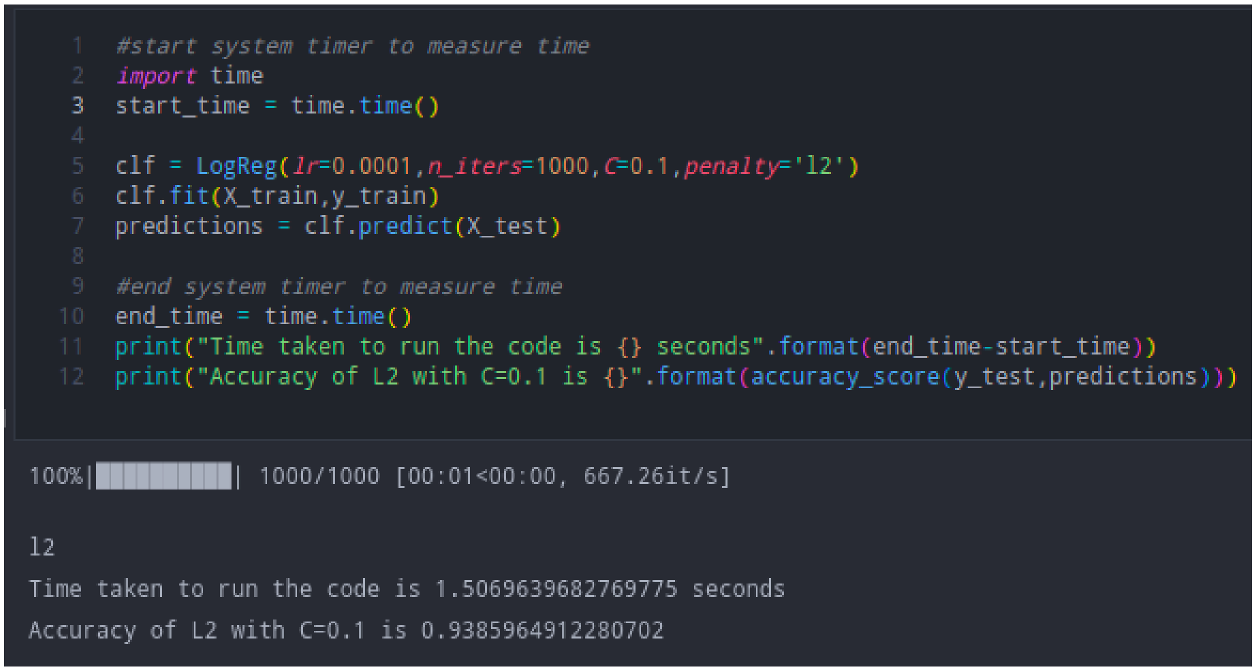
```python
#logistic reg with custom class
clf = LogReg(lr=0.0001,n_iters=1000,C=1,penalty='none')
clf.fit(X_train,y_train)
predictions = clf.predict(X_test)
print("Accuracy of No Regularization with C=1 is {}".format(accuracy_score(y_test,predictions)))
```

```
100%|██████████| 1000/1000 [00:02<00:00, 467.13it/s]

No regularization
Accuracy of No Regularization with C=1 is 0.5
```



```python
#logistic reg with custom class
import time
start_time = time.time()
clf = LogReg(lr=0.0001,n_iters=1000,C=1,penalty='l1')
clf.fit(X_train,y_train)
predictions = clf.predict(X_test)
#end system timer to measure time
end_time = time.time()
print("Time taken to run the code is {} seconds".format(end_time-start_time))
print("Accuracy of L1 with C=1 is {}".format(accuracy_score(y_test,predictions)))
```

```
100%|██████████| 1000/1000 [00:02<00:00, 445.67it/s]

l1
Time taken to run the code is 2.2481753826141357 seconds
Accuracy of L1 with C=1 is 0.9473684210526315
```



```
#start system timer to measure time
import time
start_time = time.time()

clf = LogReg(lr=0.0001,n_iters=1000,C=0.1,penalty='l2')
clf.fit(X_train,y_train)
predictions = clf.predict(X_test)

#end system timer to measure time
end_time = time.time()
print("Time taken to run the code is {} seconds".format(end_time-start_time))
print("Accuracy of L2 with C=0.1 is {}".format(accuracy_score(y_test,predictions)))
```

```
100%|██████████| 1000/1000 [00:01<00:00, 667.26it/s]

l2
Time taken to run the code is 1.5069639682769775 seconds
Accuracy of L2 with C=0.1 is 0.9385964912280702
```

Inferences

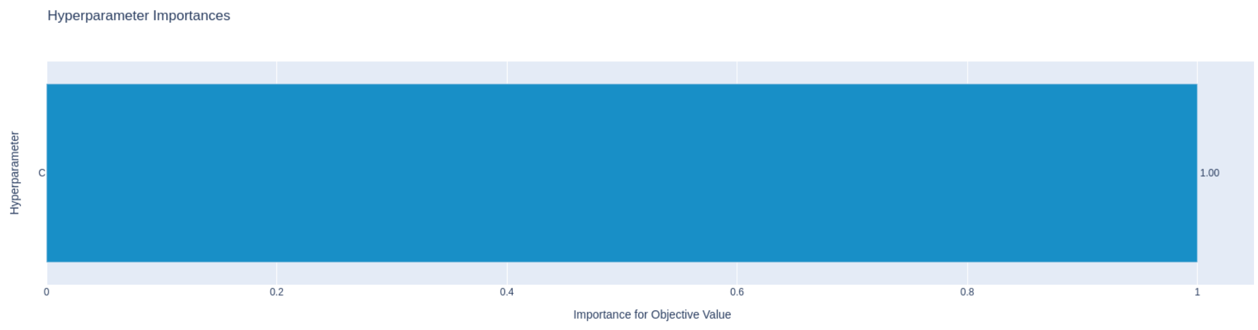
From the output we can clearly see that :

- C values contribute quite a amount to the Logistic Regression.
- What type of penalty we take is important for getting better results.
- Number of iterations will help the model to converge better.
- L2 is faster in this case compared to L1 and No Regularization.

Conclusion

By playing around with all the hyperparameters of Logistic regression , it is evident that with regression we arrive at better conclusions.

We also can prove that choosing the correct C value is important from the following plot.


Hyperparameter Importances
Hyperparameter
C
Importance for Objective Value