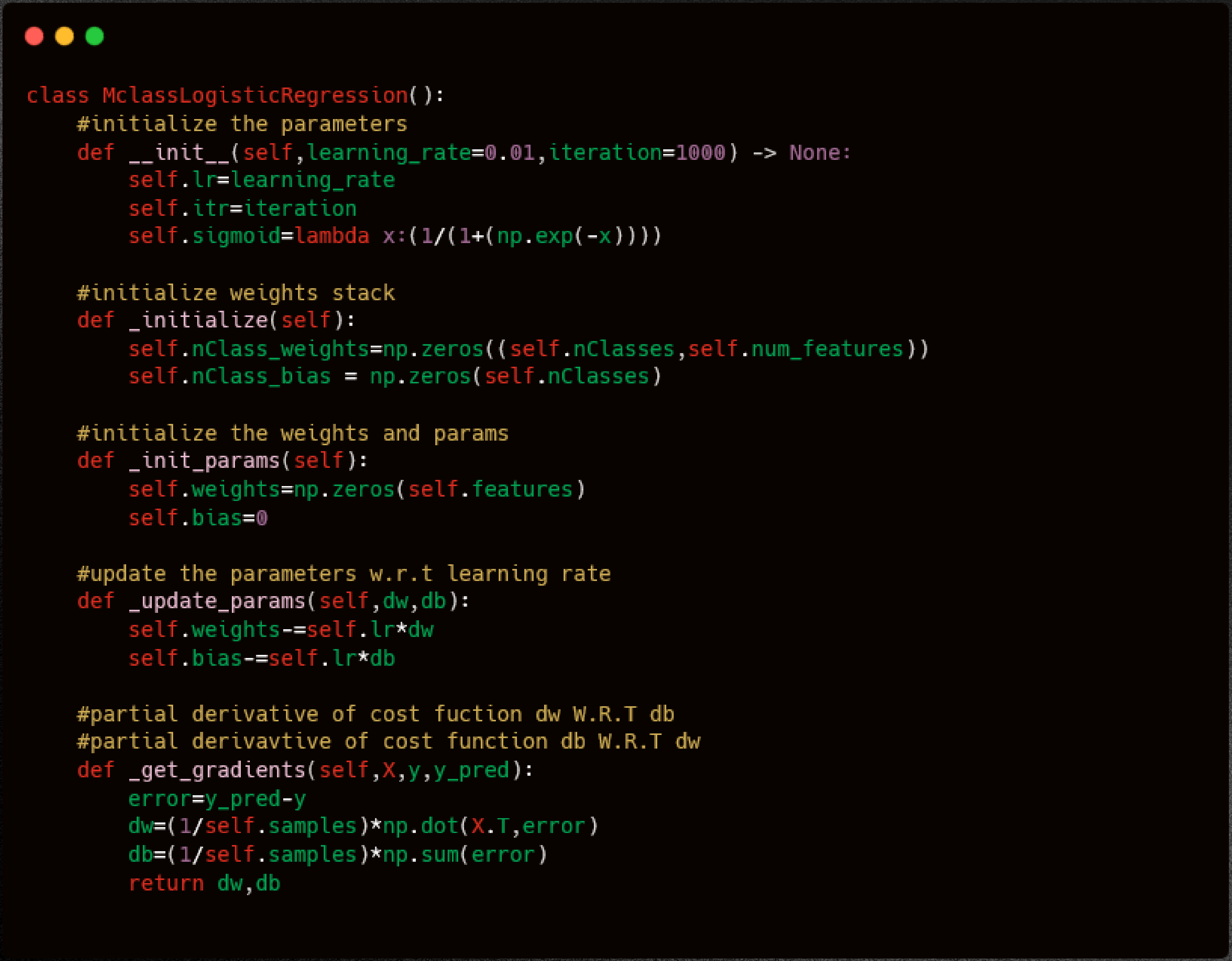
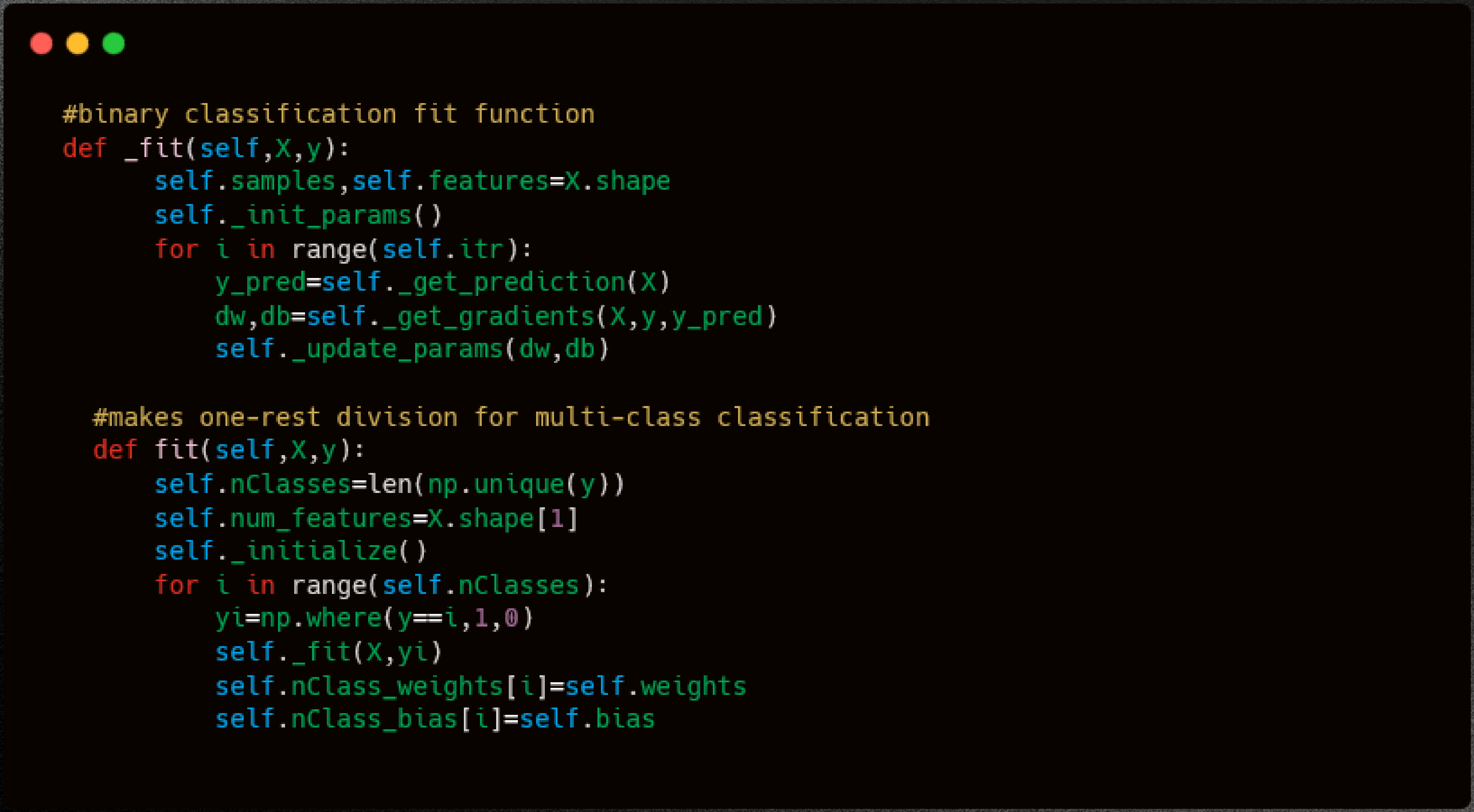
Multi-Class Classification - Using Logistic Regression

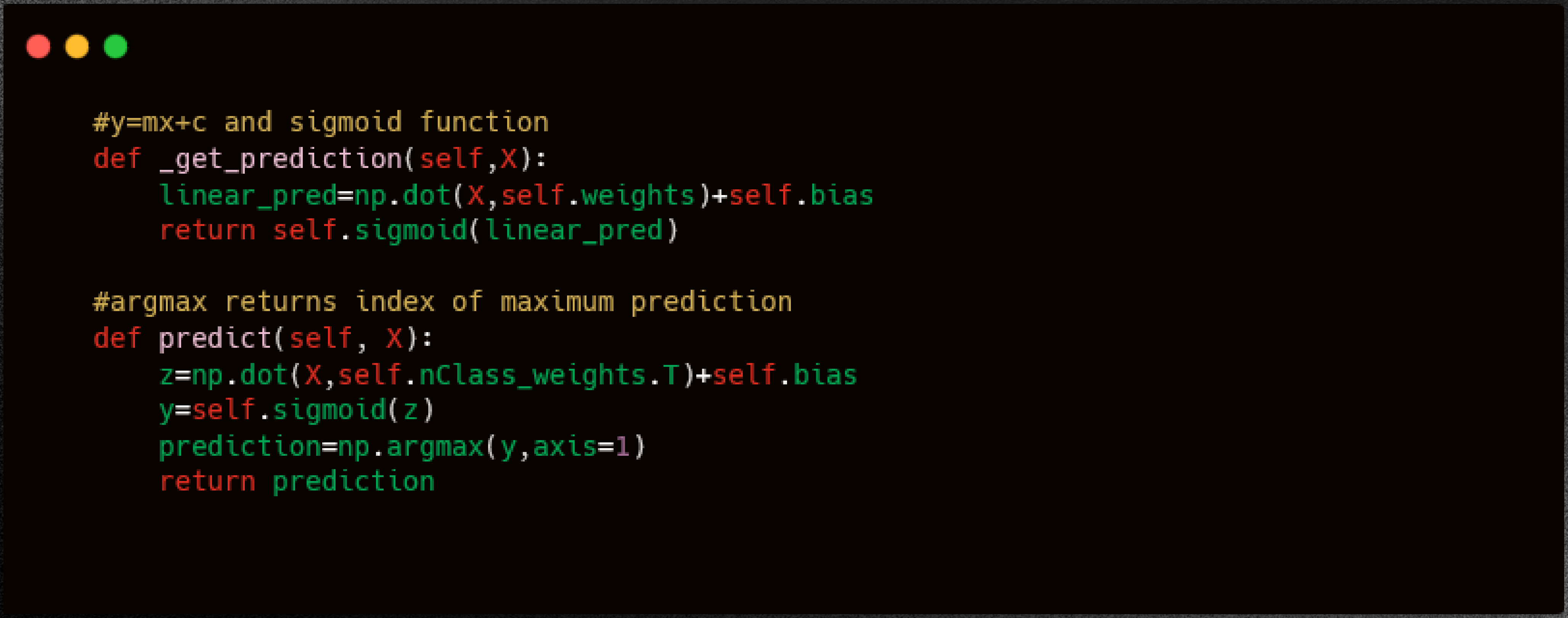
@abhishek kaddipudi

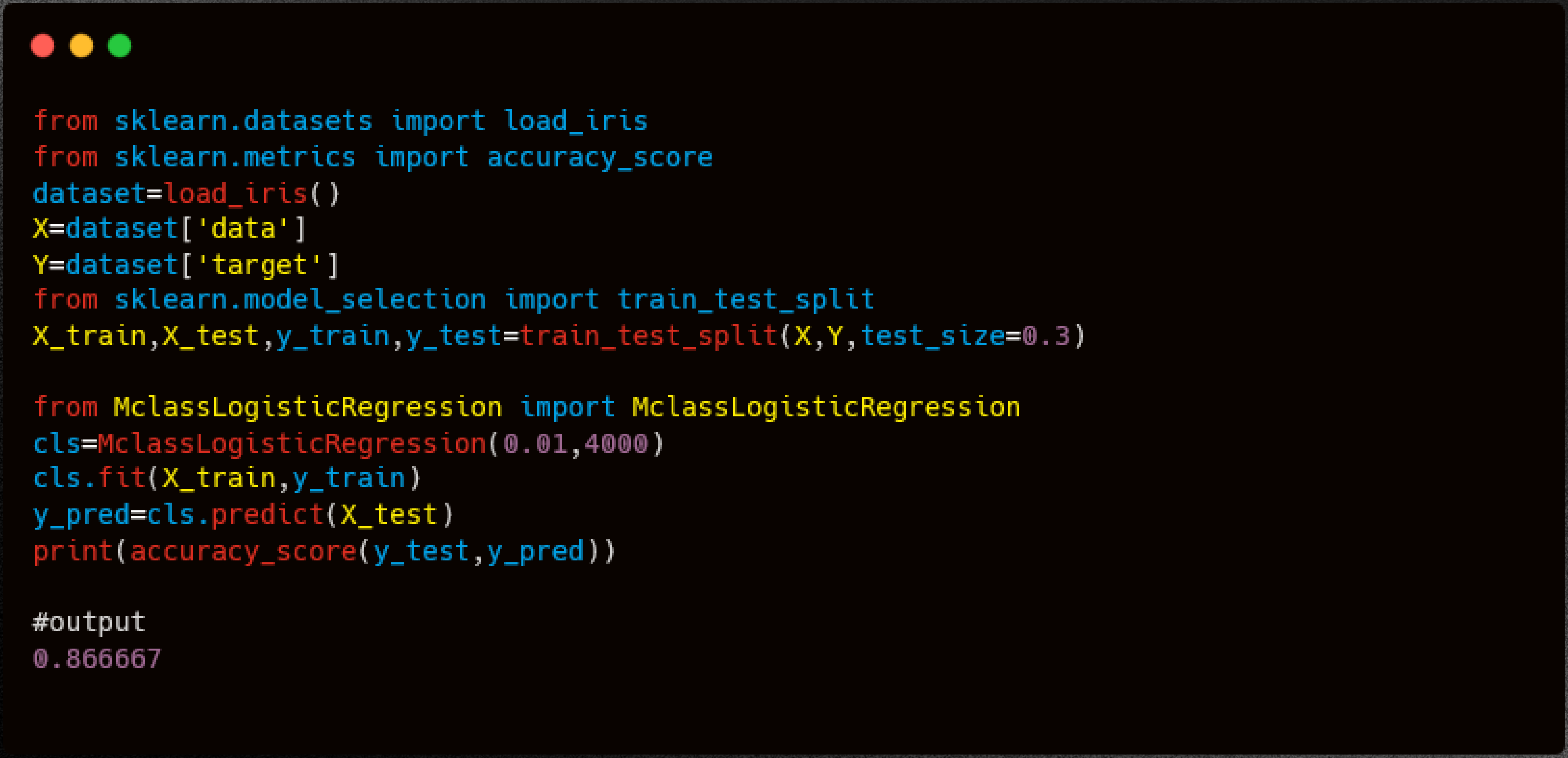


```python
class MclassLogisticRegression():
    #initialize the parameters
    def __init__(self,learning_rate=0.01,iteration=1000) -> None:
        self.lr=learning_rate
        self.itr=iteration
        self.sigmoid=lambda x:(1/(1+(np.exp(-x))))

    #initialize weights stack
    def _initialize(self):
        self.nClass_weights=np.zeros((self.nClasses,self.num_features))
        self.nClass_bias = np.zeros(self.nClasses)

    #initialize the weights and params
    def _init_params(self):
        self.weights=np.zeros(self.features)
        self.bias=0

    #update the parameters w.r.t learning rate
    def _update_params(self,dw,db):
        self.weights-=self.lr*dw
        self.bias-=self.lr*db

    #partial derivative of cost fuction dw W.R.T db
    #partial derivavtive of cost function db W.R.T dw
    def _get_gradients(self,X,y,y_pred):
        error=y_pred-y
        dw=(1/self.samples)*np.dot(X.T,error)
        db=(1/self.samples)*np.sum(error)
        return dw,db
```



```
#binary classification fit function
def _fit(self,X,y):
      self.samples,self.features=X.shape
      self._init_params()
      for i in range(self.itr):
          y_pred=self._get_prediction(X)
          dw,db=self._get_gradients(X,y,y_pred)
          self._update_params(dw,db)

  #makes one-rest division for multi-class classification
  def fit(self,X,y):
      self.nClasses=len(np.unique(y))
      self.num_features=X.shape[1]
      self._initialize()
      for i in range(self.nClasses):
          yi=np.where(y==i,1,0)
          self._fit(X,yi)
          self.nClass_weights[i]=self.weights
          self.nClass_bias[i]=self.bias
```


```
#y=mx+c and sigmoid function
def _get_prediction(self,X):
    linear_pred=np.dot(X,self.weights)+self.bias
    return self.sigmoid(linear_pred)

#argmax returns index of maximum prediction
def predict(self, X):
    z=np.dot(X,self.nClass_weights.T)+self.bias
    y=self.sigmoid(z)
    prediction=np.argmax(y,axis=1)
    return prediction
```



```
from sklearn.datasets import load_iris
from sklearn.metrics import accuracy_score
dataset=load_iris()
X=dataset['data']
Y=dataset['target']
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,Y,test_size=0.3)

from MclassLogisticRegression import MclassLogisticRegression
cls=MclassLogisticRegression(0.01,4000)
cls.fit(X_train,y_train)
y_pred=cls.predict(X_test)
print(accuracy_score(y_test,y_pred))

#output
0.866667
```