```
class LogisticRegression:

    def __init__(self, lr = 0.01, lamb = 10, num_iter = 1000, fit_intercept = True):
        self.lr = lr
        self.lamb = lamb
        self.num_iter = num_iter
        self.fit_intercept = fit_intercept

    def __add_intercept(self, X):
        '''This function adds an offset term'''
        intercept = np.ones((X.shape[0], 1))
        return np.concatenate((intercept, X), axis=1)

    def __sigmoid(self, z):
        '''***Define Sigmoid Function here. z is a vector***'''

    def fit(self, X, y):
        if self.fit_intercept:
            X = self.__add_intercept(X)

        # weights initialization
        self.theta = '''***Initialize theta. What should be the dimension of theta?***'''

        for i in range(self.num_iter):
            z = ... '''Implement x^T(theta)'''
            h = self.__sigmoid(z)

            '''***[Critical Step] - Using the formula in question prompt define gradient in vector form***'''
            gradient = ...

            '''***[Critical Step] - Update theta using the gradient and learning rate***'''
            self.theta -= ...

    def predict_prob(self, X):
      ''' This function outputs the predicted probablities'''
        if self.fit_intercept:
            X = self.__add_intercept(X)

        return self.__sigmoid(np.dot(X, self.theta))

    def predict(self, X, threshold):
      ''' This function outputs the predicted labels'''
        # usually we take threshold as 0.5
        return self.predict_prob(X) >= threshold
```

```
'''***Select an appropriate lambda parameter***'''
model = LogisticRegression(lamb = ...)
model.fit(X_train, y_train)

# Generate Predictions
preds = model.predict(X_test, threshold = 0.5)

# accuracy
'''***As you try different lambda observe and report how accuracy changes***'''
(preds == y_test).mean()
```