Part C – Logistic Regression

Learning Task 1: Classification model (LR1) using Logistic Regression with decision probability threshold from 0.3, 0.4, 0.5, 0.6, 0.7.

In [10]:
```
import numpy as np
import matplotlib.pyplot as plt
import math

def sigmoid(z):
    """
    Computes the sigmoid function for the input z.
    """
    return 1 / (1 + np.exp(-z))

class LogisticRegression:
    """
    Logistic Regression model.
    """
    def __init__(self, learning_rate=0.00001, num_iterations=10000):
        self.learning_rate = learning_rate
        self.num_iterations = num_iterations
        self.weights = None
        self.bias = None

    def fit(self, X, y, gd_type="batch", batch_size=32):
        """
        Fits the model to the training data using the specified GD algorithm.
        """
        # Initialize parameters
        n_samples, n_features = X.shape
        self.weights = np.zeros(n_features)
        self.bias = 0

        costs = []
```

```
        # Gradient descent
        dw=np.zeros(n_features)
        for i in range(self.num_iterations):
            if gd_type == "batch":
                # Batch gradient descent
                y_pred=np.zeros(n_samples)
                z_arr=np.zeros(n_samples)
                for j in range(n_samples):
                    z = np.dot(X[j], self.weights) + self.bias
    #                 print(z)
                    z_arr[j]=z
                    y_pred[j]=(sigmoid(z))

                for j in range(n_samples):
                    dw+=(y_pred[j]-y[j])*X[j]

                dw=(1/n_samples)*dw
                db = (1 / n_samples) * np.sum(y_pred - y)

                y_pred = np.clip(y_pred, 1e-15, 1-(1e-15))
                cost=0
                for j in range(n_samples):
                    cost+=(y[j]*np.log(y_pred[j])+(1-y[j])*np.log(1-y_pred[j]))
                cost=cost/n_samples
                costs.append(-cost)
#                 costs.append(-np.mean(y*np.log(y_pred) + (1-y)*np.log((1-y_pred))))

#                 print(y_pred)
```

```
            elif gd_type == "mini-batch":
                # Mini-batch gradient descent
                batch_indices = np.random.choice(n_samples, batch_size, replace=False)
                print(batch_indices)
                X_batch = X[batch_indices]
                y_batch = y[batch_indices]
                y_pred=np.zeros(batch_size)
                z_arr=np.zeros(batch_size)
                for j in range(batch_size):
                    z = np.dot(X_batch[j], self.weights) + self.bias
                    z_arr[j]=z
                    y_pred[j]=(sigmoid(z))

                for j in range(batch_size):
                    dw+=(y_pred[j]-y[j])*X[j]

                dw=(1/batch_size)*dw
                db = (1 / batch_size) * np.sum(y_pred - y_batch)

                y_pred = np.clip(y_pred, 1e-15, 1-(1e-15))
                cost=0
                for j in range(batch_size):
                    cost+=(y[j]*np.log(y_pred[j])+(1-y[j])*np.log(1-y_pred[j]))
                cost=cost/batch_size
                costs.append(-cost)

            else:
                # Stochastic gradient descent
                j = np.random.randint(n_samples)
```

```
            else:
                # Stochastic gradient descent
                j = np.random.randint(n_samples)

                z = np.dot(X[j], self.weights) + self.bias
                y_pred=(sigmoid(z))

                dw+=(y_pred-y[j])*X[j]
                db = np.sum(y_pred - y[j])

                y_pred = np.clip(y_pred, 1e-15, 1-(1e-15))
                cost=0
                cost+=(y[j]*np.log(y_pred)+(1-y[j])*np.log(1-y_pred))
                costs.append(-cost)


            # Update parameters
            self.weights -= self.learning_rate * dw
            self.bias -= self.learning_rate * db
#         print(costs,self.num_iterations)
        plt.scatter(np.array(range(self.num_iterations)),costs)
        plt.plot()

    def predict(self, X, threshold=0.5):
        """
        Predicts the class labels for the input data.
        """
        y_pred = sigmoid(np.dot(X, self.weights) + self.bias)
        y_pred[y_pred >= threshold] = 1
        y_pred[y_pred < threshold] = 0
        return y_pred.astype(int)
```

This is an implementation of a logistic regression algorithm using gradient descent to train the model.

The algorithm works as follows:

1) Import necessary libraries numpy and matplotlib.pyplot
2) Define the sigmoid function which is used as the activation function in logistic regression.
3) Define a LogisticRegression class which will contain the functions for training and predicting the model.
4) In the class constructor, initialize the learning rate, number of iterations, weights, and bias to None.
5) Define a fit function that takes in the input data, X, the target labels, y, the type of gradient descent (batch, mini-batch, or stochastic), and the batch size (if using mini-batch gradient descent).
6) Initialize the weights and bias to zeros and set up an array to store the costs.
7) For each iteration, compute the predicted output (y_pred) and the cost of the model. If using batch gradient descent, loop through all the training samples and compute the gradient for each sample. If using mini-batch gradient descent, randomly select a subset of the training samples and compute the gradient for that subset. If using stochastic gradient descent, randomly select one sample from the training set and compute the gradient for that sample.
8) Update the weights and bias based on the computed gradient using the learning rate.
9) Define a predict function that takes in the input data, X, and a threshold value. Predict the output based on the weights and bias and the sigmoid function. If the predicted output is greater than or equal to the threshold, set it to 1, else set it to 0.
10) Return the predicted output.
11) Overall, this code implements logistic regression using gradient descent and allows for different types of gradient descent to be used. It also computes and plots the cost of the model during training.

Learning Task1 Batch

In [11]:
```
Engineering_Task1(data)
X = data.drop(['diagnosis','id'], axis=1).values
y = data['diagnosis'].values
y[y=='M']=0
y[y=='B']=1
# shuffle the indices of the data
indices = list(range(len(X)))
random.shuffle(indices)

# calculate the split point
split_idx = int(0.67 * len(X))


# split the data into training and testing sets
train_X = X[indices[:split_idx]]
train_y = y[indices[:split_idx]]
test_X = X[indices[split_idx:]]
test_y = y[indices[split_idx:]]

model = LogisticRegression()
# print(train_y)
model.fit(train_X, train_y,"batch",)
thresholds = [0.3, 0.4, 0.5, 0.6, 0.7]
for threshold in thresholds:
    y_pred = model.predict(test_X, threshold=threshold)
    accuracy = np.mean(y_pred == test_y)
    print(f"Testing accuracy with threshold {threshold}: {accuracy:.2f}")
```

```
Testing accuracy with threshold 0.3: 0.91
Testing accuracy with threshold 0.4: 0.90
Testing accuracy with threshold 0.5: 0.92
Testing accuracy with threshold 0.6: 0.91
Testing accuracy with threshold 0.7: 0.91
```

This code performs binary classification on the Cancer dataset using logistic regression. The dataset is loaded into a pandas DataFrame object called "data" and preprocessed to prepare for training the logistic regression model.

The data is split into training and testing sets using 67% of the data for training and 33% for testing. The target variable, "diagnosis", is transformed into binary values (0 for malignant and 1 for benign). The indices of the data are shuffled randomly to ensure that the training and testing sets are representative of the entire dataset.

A LogisticRegression object is created, and the fit method is called to train the model using stochastic gradient descent with a learning rate of 0.0001. After training, the model is tested on the testing set using different threshold values (0.3, 0.4, 0.5, 0.6, and 0.7) to determine the accuracy of the predictions.

The algorithm used by logistic regression is as follows:

1) Initialize the parameters w and b to zeros or small random values.
2) Calculate the cost function using the current parameters w and b.
3) Calculate the gradients of the cost function with respect to the parameters w and b.
4) Update the parameters w and b using the gradients and a learning rate alpha.
5) Repeat steps 2-4 until convergence or a maximum number of iterations is reached.

The cost function used by logistic regression is the log loss or binary cross-entropy loss. The goal of logistic regression is to minimize this cost function to obtain the best parameters for the model.

Learning Task 2: Applying Feature Engineering Task 1 and Feature Engineering Task 2 and then build a classification model (LR2) using Logistic Regression with decision probability threshold from 0.3, 0.4, 0.5, 0.6, 0.7.

Learning Task2

```
Engineering_Task1(data_norm)
Engineering_Task2(data_norm)
X_norm = data_norm.drop(['diagnosis','id'], axis=1).values
y_norm = data_norm['diagnosis'].values
y_norm[y_norm=='M']=0
y_norm[y_norm=='B']=1
# shuffle the indices of the data
indices = list(range(len(X_norm)))
random.shuffle(indices)

# calculate the split point
split_idx = int(0.67 * len(X_norm))


# split the data into training and testing sets
train_X_norm = X_norm[indices[:split_idx]]
train_y_norm = y_norm[indices[:split_idx]]
test_X_norm = X_norm[indices[split_idx:]]
test_y_norm = y_norm[indices[split_idx:]]

model = LogisticRegression()
# print(train_y_norm)
model.fit(train_X_norm, train_y_norm,"Stochastic Gradient Descent")
thresholds = [0.3, 0.4, 0.5, 0.6, 0.7]
print('Using Stochastic Gradient Descent with Learning rate:0.0001')
for threshold in thresholds:
    y_pred_norm = model.predict(test_X_norm, threshold=threshold)
    accuracy = np.mean(y_pred_norm == test_y_norm)
    print(f"Testing accuracy with threshold {threshold}: {accuracy:.2f}")
```

This code performs logistic regression on a breast cancer diagnosis dataset to predict whether a tumor is benign or malignant. The dataset is normalized using mean normalization before splitting into training and testing sets. The logistic regression model is then trained using stochastic gradient descent with a learning rate of 0.0001. The model is evaluated using different threshold values, and the accuracy is reported for each threshold.

Algorithm:

1) Import necessary libraries.
2) Load the dataset and normalize the data using mean normalization.
3) Split the data into training and testing sets.
4) Create an instance of the LogisticRegression class.
5) Train the logistic regression model using stochastic gradient descent with a learning rate of 0.0001.
6) Evaluate the model's accuracy using different threshold values.
7) Print the accuracy for each threshold.