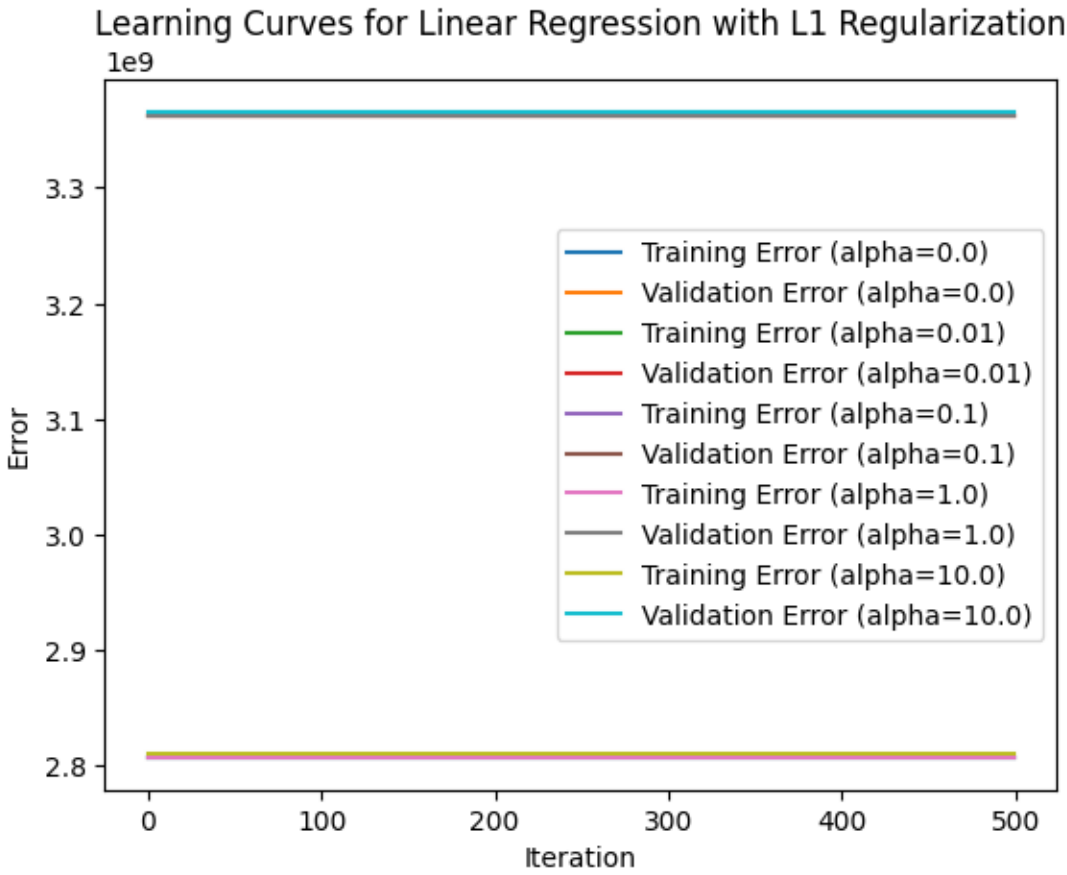
مازندرانیان – ۸۳۰۴۰۲۰۶۶ – تمرین ۴ یادگیری ماشین

سوال ۱) پیاده سازی **Linear Regression** به همراه **L1, L2 Regularization** روی دیتاست **Housing**

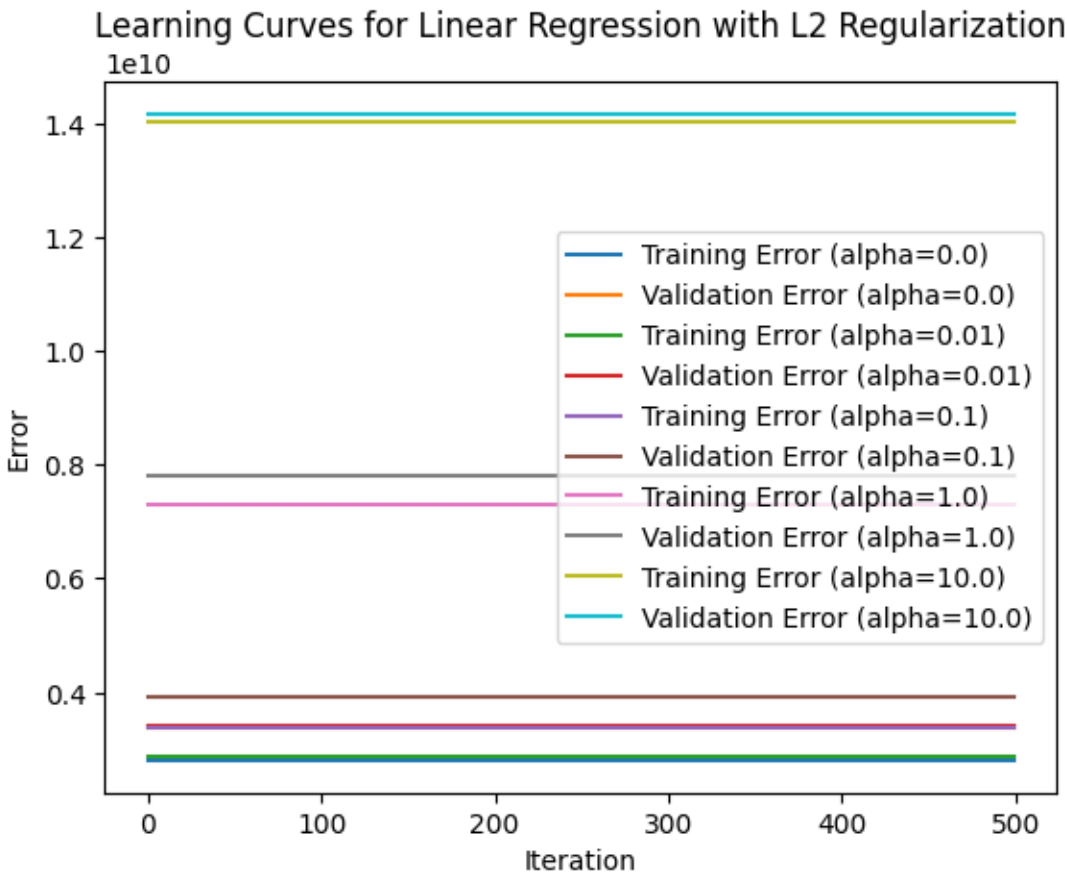
Linear Regression — Python

```python
class LinearRegression:
    def __init__(self, learning_rate=0.01, iterations=500, penalty=None, alpha=0.0):
        self.learning_rate = learning_rate
        self.iterations = iterations
        self.penalty = penalty
        self.alpha = alpha
        self.weights = None
        self.bias = None

    def _cost_function(self, X, y, weights, bias):
        num_samples = X.shape[0]
        predictions = np.dot(X, weights) + bias
        cost = (1 / (2 * num_samples)) * np.sum((predictions - y) ** 2)

        if self.penalty == 'l1':
            cost += self.alpha * np.sum(np.abs(weights))
        elif self.penalty == 'l2':
            cost += (self.alpha / (2 * num_samples)) * np.sum(weights ** 2)

        return cost

    def fit(self, X, y):
        num_samples, num_features = X.shape
        self.weights = np.zeros(num_features)
        self.bias = 0

        for _ in range(self.iterations):
            predictions = np.dot(X, self.weights) + self.bias
            dw = (1 / num_samples) * np.dot(X.T, (predictions - y))
            db = (1 / num_samples) * np.sum(predictions - y)

            if self.penalty == 'l1':
                dw += self.alpha * np.sign(self.weights)
            elif self.penalty == 'l2':
                dw += (self.alpha / num_samples) * self.weights

            self.weights -= self.learning_rate * dw
            self.bias -= self.learning_rate * db

    def predict(self, X):
        return np.dot(X, self.weights) + self.bias
```

از تابع cost_function_ برای ذخیره سازی cost به منظور plot کردن آن ها استفاده شده است و در قسمت fit جداگانه ارور ها را محاسبه میکنیم، براساس گرادیان محاسبه شده وزن ها و بایاس آپدیت می شوند.

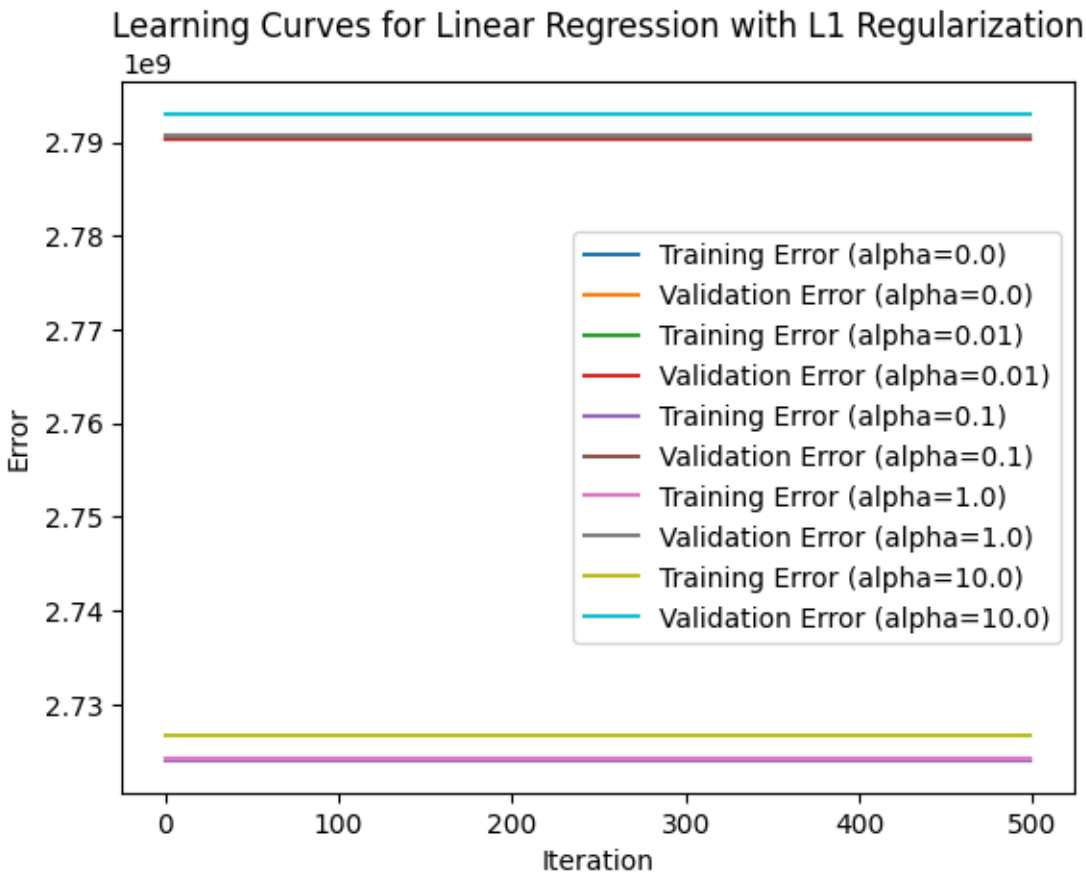


شکل ۱

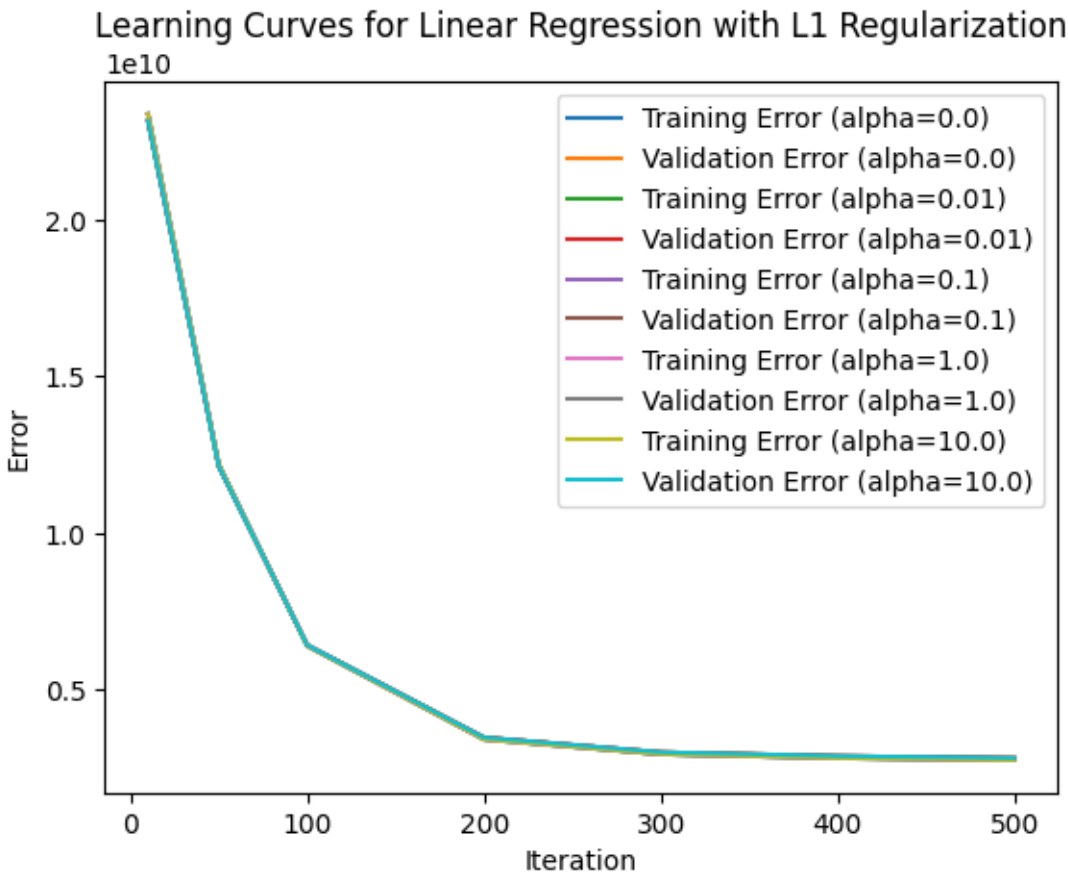


شکل ۲

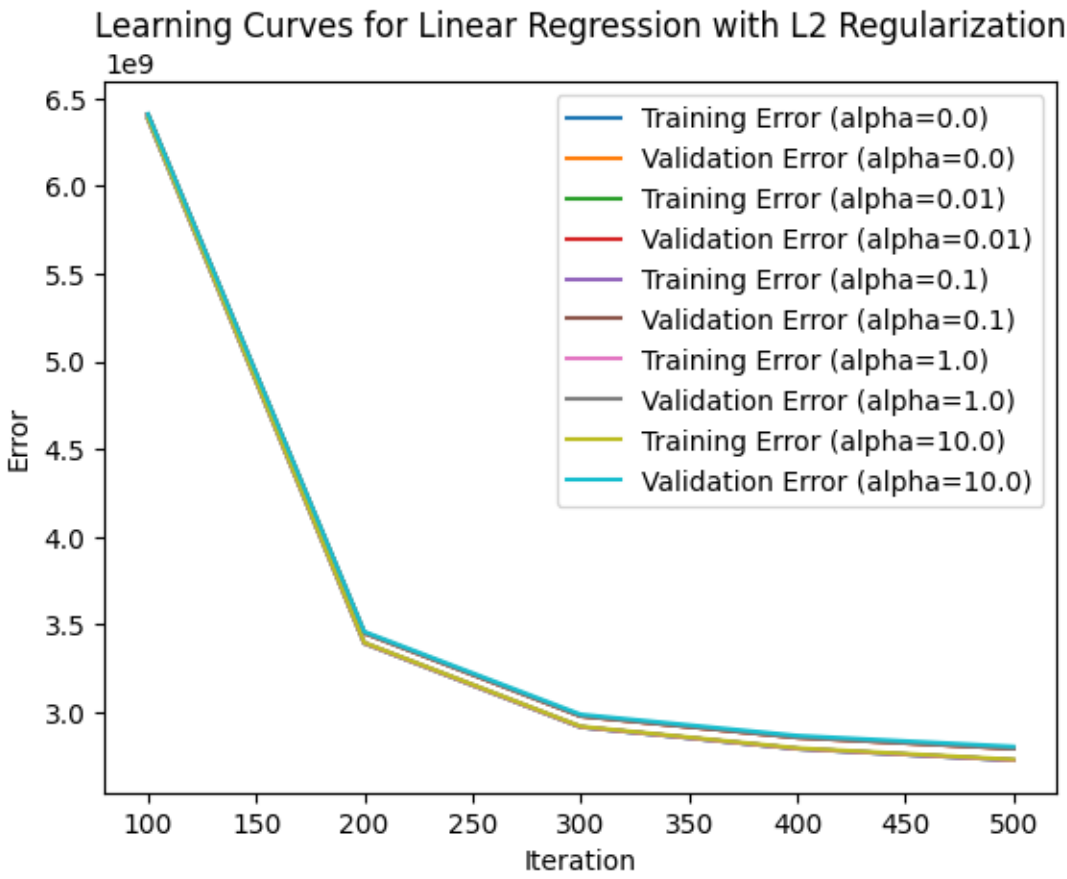
شکل های شماره ۱ و ۲ مربوط به دسته بندی نوع اول است که مشاهده می شود که مدل دچار overfit شده است.


Learning Curves for Linear Regression with L1 Regularization
1e9
2.79
2.78
2.77
2.76
2.75
2.74
2.73
Error
0
100
200
300
400
500
Iteration
Training Error (alpha=0.0)
Validation Error (alpha=0.0)
Training Error (alpha=0.01)
Validation Error (alpha=0.01)
Training Error (alpha=0.1)
Validation Error (alpha=0.1)
Training Error (alpha=1.0)
Validation Error (alpha=1.0)
Training Error (alpha=10.0)
Validation Error (alpha=10.0)

شکل ۳


Learning Curves for Linear Regression with L1 Regularization
1e10
2.0
1.5
1.0
0.5
Error
0
100
200
300
400
500
Iteration
Training Error (alpha=0.0)
Validation Error (alpha=0.0)
Training Error (alpha=0.01)
Validation Error (alpha=0.01)
Training Error (alpha=0.1)
Validation Error (alpha=0.1)
Training Error (alpha=1.0)
Validation Error (alpha=1.0)
Training Error (alpha=10.0)
Validation Error (alpha=10.0)

شکل ٤



شکل ۵

در نمودار های شماره ۴ و ۵ تعداد iteration ها را از ۵۰ تا ۵۰۰ در نظر گرفتیم.

فرمول محاسبه ی گرادیان و تابع ضرر در رگرسیون خطی در ادامه آمده است:

3. **Gradient Descent**

To minimize the cost function $J(w, b)$, the code uses **gradient descent**. The gradients for w and b are derived as follows:

Without Regularization:

- Gradient of w:

$$\frac{\partial J}{\partial w} = \frac{1}{m} X^\top \cdot (\hat{y} - y)$$

- Gradient of b:

$$\frac{\partial J}{\partial b} = \frac{1}{m} \sum_{i=1}^{m} (\hat{y}_i - y_i)$$

With Regularization:

- For **L1 Regularization**, add $\alpha \cdot \text{sign}(w)$ to the gradient of w.
- For **L2 Regularization**, add $\frac{\alpha}{m} \cdot w$ (or $\alpha \cdot w$ depending on scaling) to the gradient of w.

شکل ۶

2. **Cost Function**

The cost function measures how well the model predicts the target values y. For linear regression without regularization, the **Mean Squared Error (MSE)** is used:

$$J(w, b) = \frac{1}{2m} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2$$

To account for regularization:

- **L1 Regularization (Lasso)** adds a penalty proportional to the absolute value of the weights:

$$J(w, b) = \frac{1}{2m} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2 + \alpha \sum_{j=1}^{n} |w_j|$$

- **L2 Regularization (Ridge)** adds a penalty proportional to the squared value of the weights:

$$J(w, b) = \frac{1}{2m} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2 + \frac{\alpha}{2} \sum_{j=1}^{n} w_j^2$$

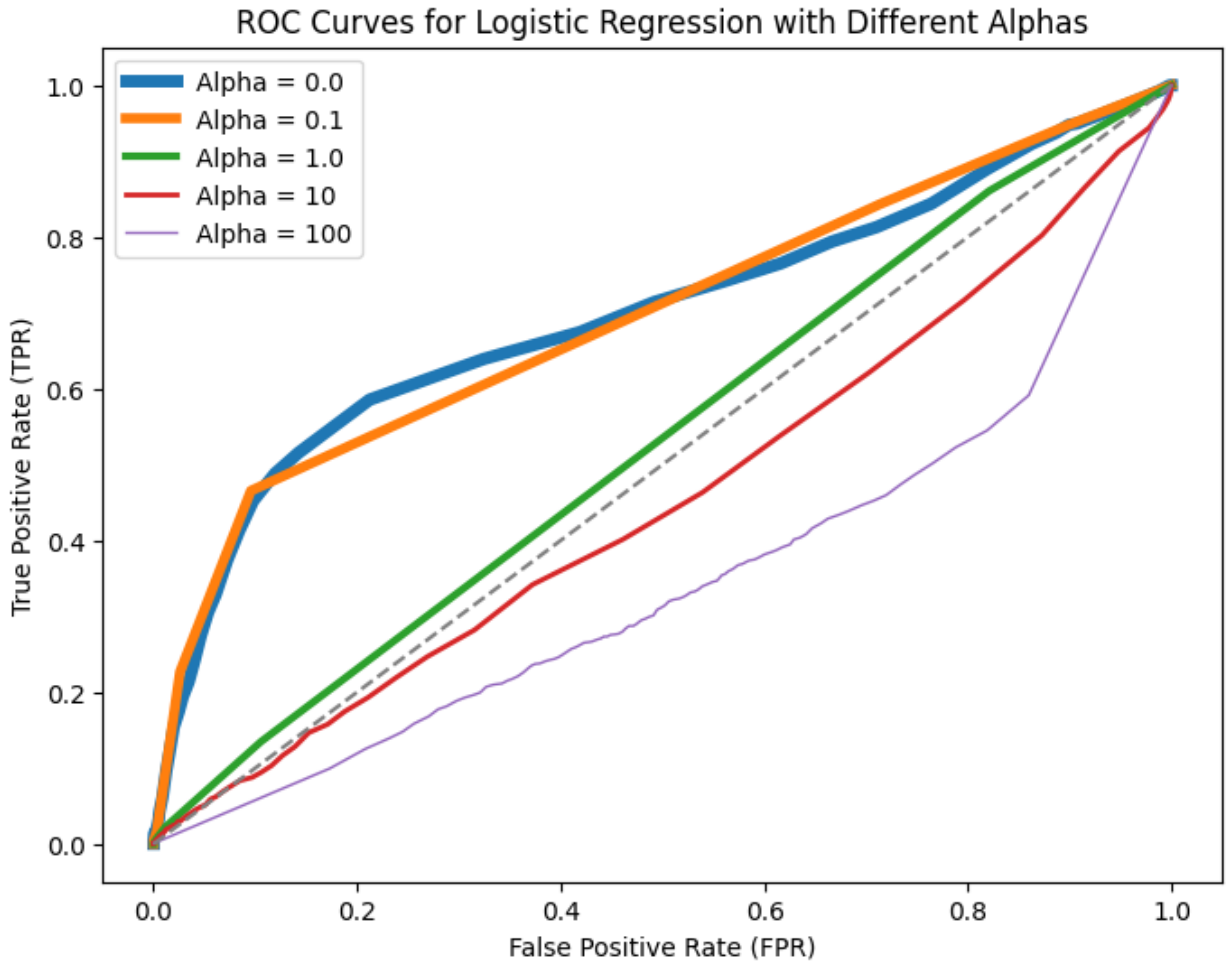
شکل ۷

سوال ۲) پیاده سازی **Logistic Regression** به همراه **L1, L2 Regularization** روی دیتاست **Credit Card**

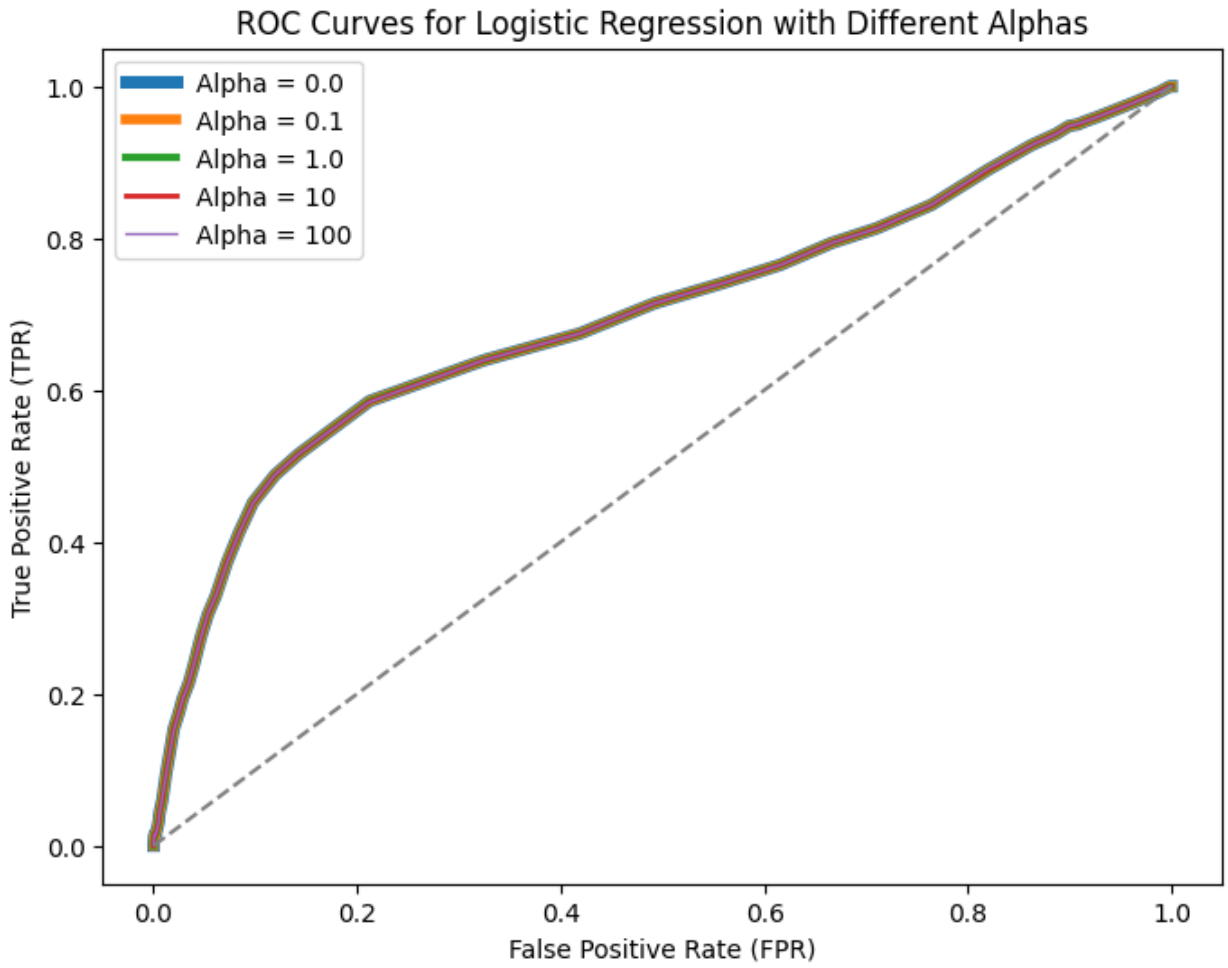
Logistic Regression — Python

```python
class LogisticRegression:
    def __init__(self, learning_rate=0.01, iterations=100, penalty=None, alpha=0.0):
        self.learning_rate = learning_rate
        self.iterations = iterations
        self.penalty = penalty
        self.alpha = alpha
        self.weights = None
        self.bias = None

    def _sigmoid(self, z):
        return 1 / (1 + np.exp(-z))

    def _cost_function(self, X, y, weights, bias):
        num_samples = X.shape[0]
        z = np.dot(X, weights) + bias
        predictions = self._sigmoid(z)
        cost = -(1 / num_samples) * np.sum(y * np.log(predictions) + (1 - y) * np.log(1 -
predictions))

        if self.penalty == 'l1':
            cost += self.alpha * np.sum(np.abs(weights))
        elif self.penalty == 'l2':
            cost += (self.alpha / (2 * num_samples)) * np.sum(weights ** 2)

        return cost

    def fit(self, X, y):
        num_samples, num_features = X.shape
        self.weights = np.zeros(num_features)
        self.bias = 0

        for _ in range(self.iterations):
            z = np.dot(X, self.weights) + self.bias
            predictions = self._sigmoid(z)
            dw = (1 / num_samples) * np.dot(X.T, (predictions - y))
            db = (1 / num_samples) * np.sum(predictions - y)

            if self.penalty == 'l1':
                dw += self.alpha * np.sign(self.weights)
            elif self.penalty == 'l2':
                dw += (self.alpha / num_samples) * self.weights

            self.weights -= self.learning_rate * dw
            self.bias -= self.learning_rate * db

    def predict(self, X):
        z = np.dot(X, self.weights) + self.bias
        predictions = self._sigmoid(z)
        return (predictions >= 0.5).astype(int)
```

در پیاده سازی Logistic Regression از تابع sigmoid به عنوان تابع prediction و محاسبه ارور استفاده می شود.

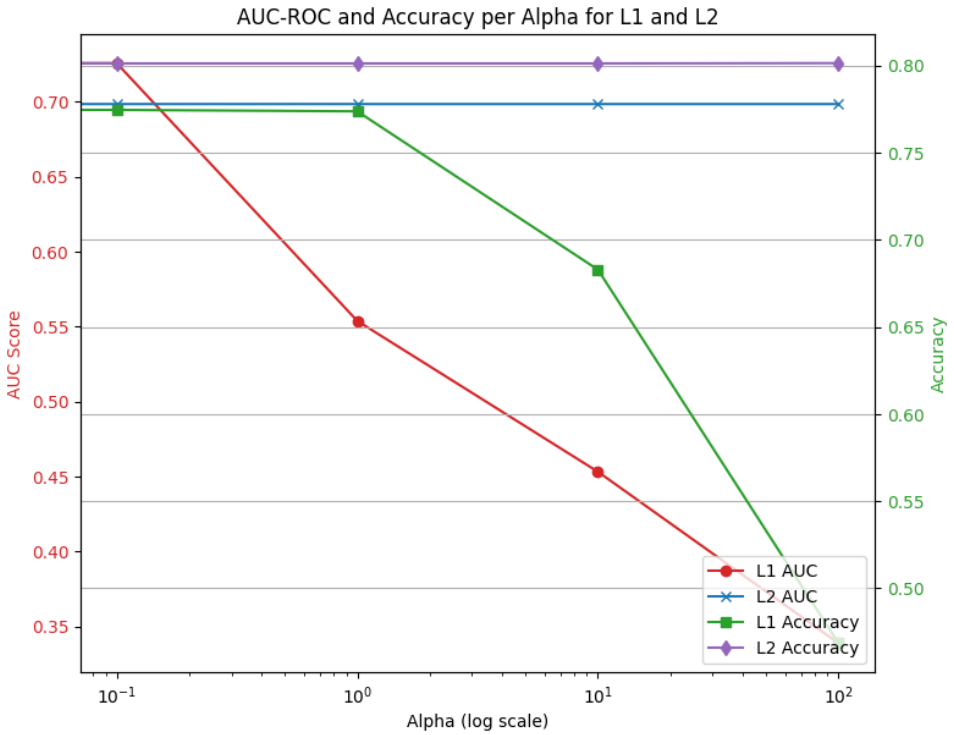


شکل ۸

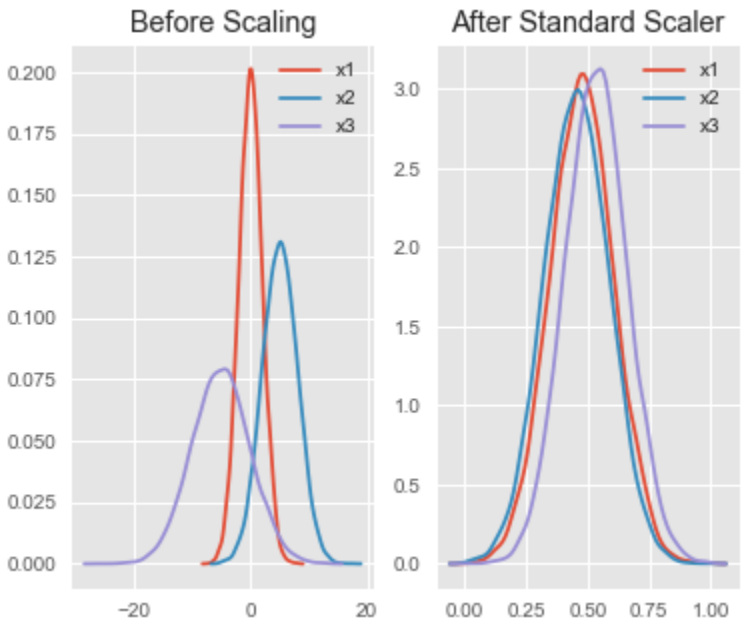


شکل ۹

شکل شماره ی ۸ مربوط به l1 و شماره ۹ مروبط به l2 می باشد.


AUC-ROC and Accuracy per Alpha for L1 and L2
AUC Score
Accuracy
Alpha (log scale)
L1 AUC
L2 AUC
L1 Accuracy
L2 Accuracy

شکل ۱۰


Before Scaling
After Standard Scaler
x1
x2
x3

شکل ۱۱