CS 412 Introduction to Machine Learning

Logistic Regression – Code Tutorial

Instructor: Wei Tang

Department of Computer Science
University of Illinois at Chicago
Chicago IL 60607

https://tangw.people.uic.edu
tangw@uic.edu

Slides credit: Martín Pellarolo

Data

```python
def generate_random_points(size=10, low=0, high=1):
    data = (high - low) * np.random.random_sample((size, 2)) + low
    return data
```

```python
N = 20 # number of samples in each class

X1 = generate_random_points(N, 0, 1)
y1 = np.ones(N)

X2 = generate_random_points(N, 1, 2)
y2 = np.zeros(N)

X = np.concatenate((X1, X2), axis=0)
y = np.concatenate((y1, y2), axis=0)
```

numpy.random.random_sample

random.random_sample(*size=None*)

Return random floats in the half-open interval [0.0, 1.0).

Results are from the "continuous uniform" distribution over the stated interval. To sample $Unif[a, b), b > a$ multiply the output of **random_sample** by *(b-a)* and add *a*:

```
(b - a) * random_sample() + a
```

Note

New code should use the `random` method of a `default_rng()` instance instead; please see the Quick Start.

Parameters: **size** : *int or tuple of ints, optional*

Output shape. If the given shape is, e.g., `(m, n, k)`, then `m * n * k` samples are drawn. Default is None, in which case a single value is returned.

Returns: **out** : *float or ndarray of floats*

Array of random floats of shape `size` (unless `size=None`, in which case a single float is returned).

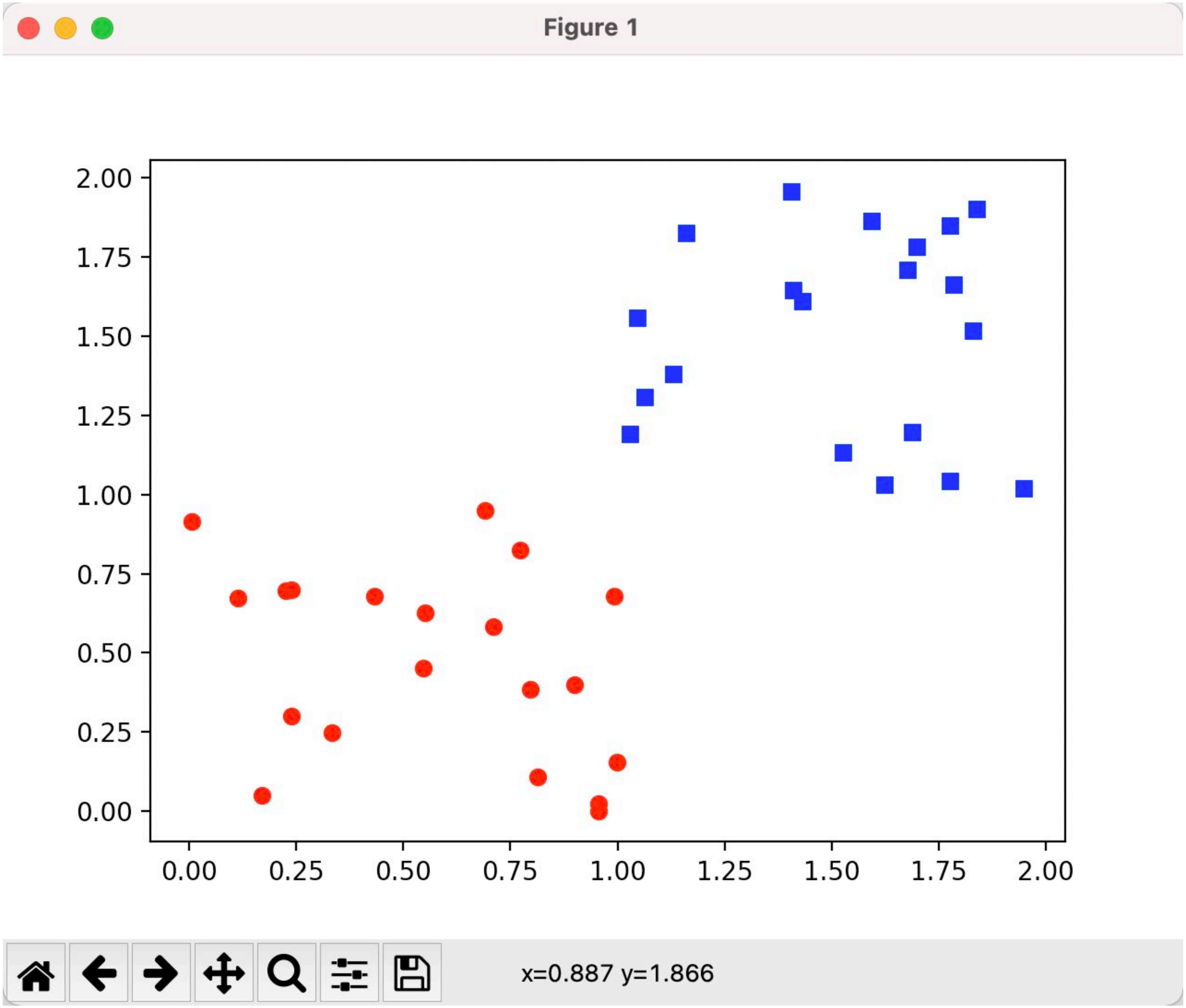
Figure 1
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0.00 0.25 0.50 0.75 1.00 1.25 1.50 1.75 2.00
x=0.887 y=1.866

Some useful functions

```python
def sigmoid(z):
    res = 1 / (1 + np.exp(-z))
    return res
```

```python
def add_intercept(X):
    intercept = np.ones((X.shape[0], 1))
    res = np.concatenate((intercept, X), axis=1)
    return res
```

```python
def loss(h, y):
    res = (-y * np.log(h) - (1 - y) * np.log(1 - h)).mean()
    return res
```

Learning

```python
def learning(X, y, lr=0.01, num_iter=100000):
    X = add_intercept(X)

    # weights initialization
    theta = np.zeros(X.shape[1])

    for i in range(num_iter):
        z = np.dot(X, theta)
        h = sigmoid(z)
        gradient = np.dot(X.T, (h - y)) / y.size
        theta -= lr * gradient

        if (i % 10000 == 0):
            z = np.dot(X, theta)
            h = sigmoid(z)
            print(f'loss: {loss(h, y)} \t')
    return theta
```

Prediction

```python
def predict(X, theta, threshold=0.5):
    X = add_intercept(X)
    prob = sigmoid(np.dot(X, theta))
    res = prob >= threshold
    return res
```

The complete pipeline

```
N = 20 # number of samples in each class

X1 = generate_random_points(N, 0, 1)
y1 = np.ones(N)

X2 = generate_random_points(N, 1, 2)
y2 = np.zeros(N)

X = np.concatenate((X1, X2), axis=0)
y = np.concatenate((y1, y2), axis=0)

theta = learning(X, y)

y_preds = predict(X, theta)

print("Average classification accuracy:", (y_preds == y).mean())

plt.plot(X1[:,0], X1[:,1], 'ro', X2[:,0], X2[:,1], 'bs')
# x2 = a*x1 + b
a = -theta[1]/theta[2]
b = -theta[0]/theta[2]
plt.plot(np.array([0,2]), np.array([0,2])*a+b, "g-")
plt.show()
```

Results

```
loss: 0.691823834433296З
loss: 0.12193492747304553
loss: 0.08102355257708946
loss: 0.0644580154145817
loss: 0.05511618732298669
loss: 0.0489784090151211
loss: 0.0445705016555993
loss: 0.04121452216444617
loss: 0.03855167714157677
loss: 0.036372816541204736
Average classification accuracy: 1.0
```

Results

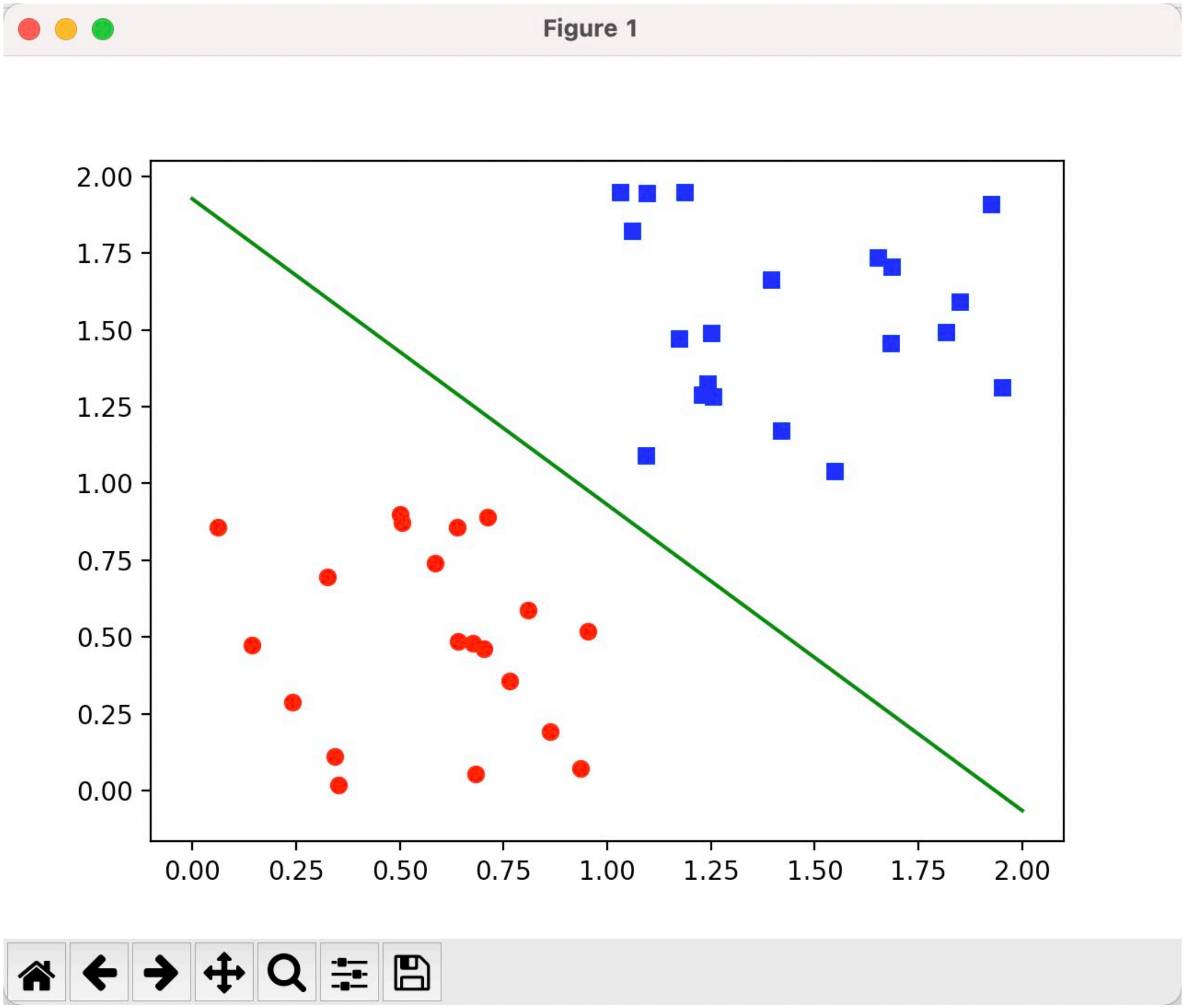

Figure 1
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

Exploration

How is the decision boundary changed during the learning process?

What will happen if the data are not linearly separable?

What is the effect of using different learning rates?