RA1911030010030.MLCore.Assignment3.Nitish

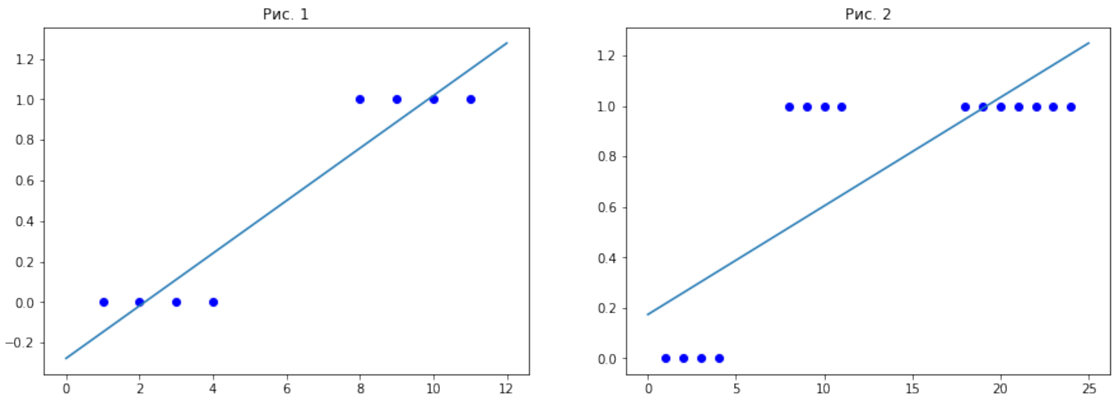
September 13, 2021

[1]:

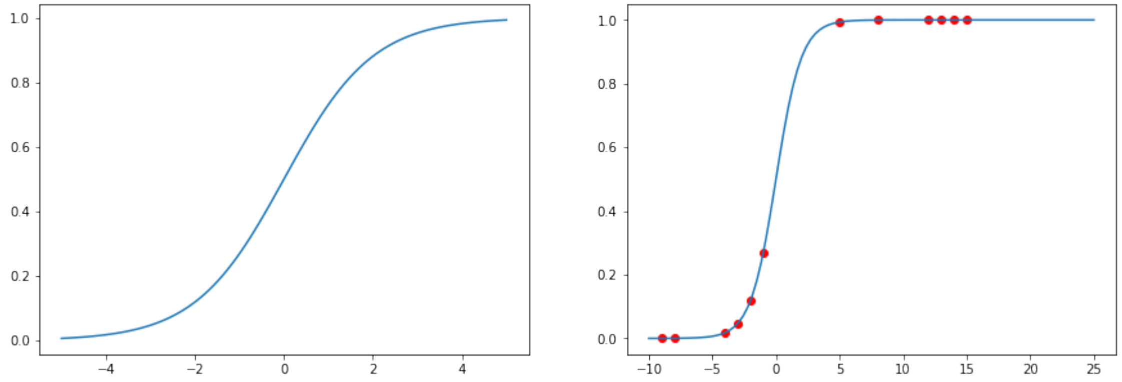
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib

from sklearn.linear_model import LinearRegression
from sklearn.datasets import make_blobs
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
```

[2]:

```
fig, ax = plt.subplots(1, 2, figsize=(15,5))

X = np.array([[1],[2],[3],[4],[8],[9],[10],[11]])
Y = np.array([0, 0, 0, 0, 1, 1, 1, 1])
model = LinearRegression()
model.fit(X, Y)
x_values = np.linspace(0, 12, 100)
y_values = [model.intercept_ + x * model.coef_[0] for x in x_values]
ax[0].plot(x_values, y_values)
ax[0].scatter(X, Y, c='b');
ax[0].title.set_text('  . 1')

X = np.array([[1],[2],[3],[4],[8],[9],[10],[11], [18], [19], [20], [21], [22],␣
 ↪[23], [24]])
Y = np.array([0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
model = LinearRegression()
model.fit(X, Y)
x_values = np.linspace(0, 25, 100)
y_values = [model.intercept_ + x * model.coef_[0] for x in x_values]
ax[1].plot(x_values, y_values)
ax[1].scatter(X, Y, c='b');
ax[1].title.set_text('  . 2')
```

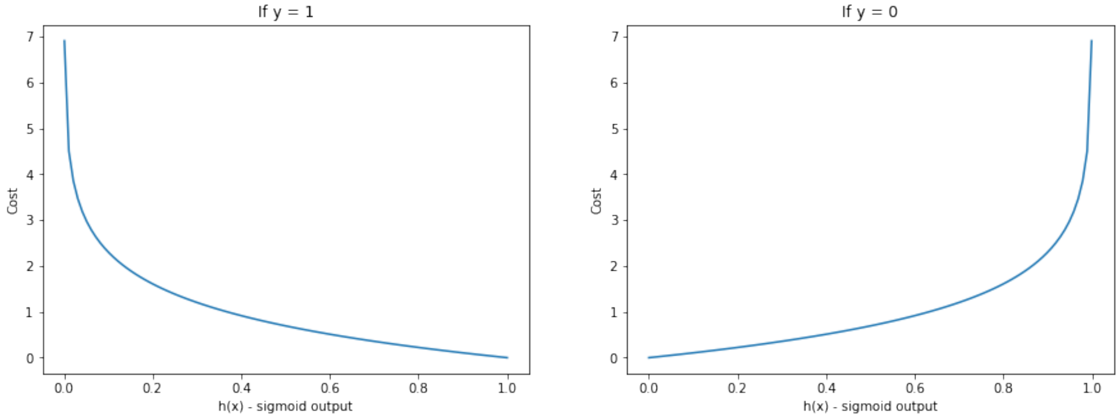


[3]:
```
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

fig, ax = plt.subplots(1, 2, figsize=(15,5))

x_values = np.linspace(-5, 5, 100)
y_values =  [sigmoid(x) for x in x_values]

ax[0].plot(x_values, y_values);

X = np.array([[-1],[-2],[-3],[-4],[-8],[-9],[5],[8], [12], [13], [14], [15]])
Y = np.array([0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1])

x_values = np.linspace(-10, 25, 100)
y_values =  [sigmoid(x) for x in x_values]

ax[1].plot(x_values, y_values);
ax[1].scatter(X, [sigmoid(x) for x in X], c='r');
```



[4]:

```
# hx - sigmoid values between (0, 1)

fig, ax = plt.subplots(1, 2, figsize=(15,5))

def if_y_1(hx):
    return -np.log(hx)

x_values = np.linspace(0.001, 1, 100)
y_values = [if_y_1(hx) for hx in x_values]

ax[0].plot(x_values, y_values)
ax[0].title.set_text('If y = 1');
ax[0].set_xlabel('h(x) - sigmoid output')
ax[0].set_ylabel('Cost')

def if_y_0(hx):
    return -np.log(1 - hx)

x_values = np.linspace(0, 0.999, 100)
y_values = [if_y_0(hx) for hx in x_values]

ax[1].plot(x_values, y_values)
ax[1].title.set_text('If y = 0');
ax[1].set_xlabel('h(x) - sigmoid output')
ax[1].set_ylabel('Cost');
```



1 Implementaion Logistic Regression

[5]:

```
class LogisticRegression:

    def __init__(self,lr=0.1,n_iters=1000):
        self.lr = lr
        self.n_iters = n_iters
        self.weights = None
        self.bias = None

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

        for _ in range(self.n_iters):
            linear_model = np.dot(X, self.weights) + self.bias
            hx = self._sigmoid(linear_model)

            dw = (X.T * (hx - y)).T.mean(axis=0)
            db = (hx - y).mean(axis=0)

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

    def predict(self,X):
        linear_model = np.dot(X,self.weights) + self.bias
        y_predicted = self._sigmoid(linear_model)
        y_predicted_cls = [1 if i > 0.5 else 0 for i in y_predicted]
        return y_predicted_cls

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

2 Use model

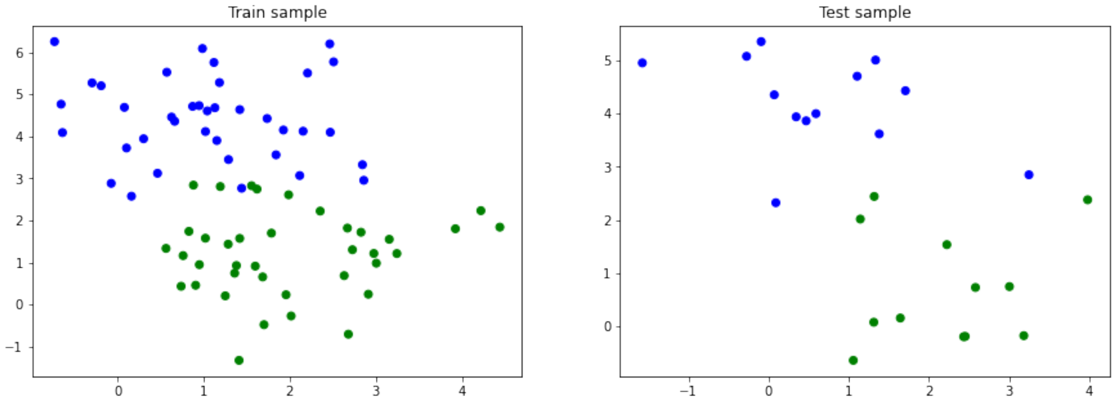
[6]:

```
X, y = make_blobs(n_samples=100, centers=2, n_features=2, random_state=0)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,␣
 ↪random_state=0)
```

[7]:

```
fig, ax = plt.subplots(1, 2, figsize=(15,5))

color = ['blue' if l == 0 else 'green' for l in y_train]
ax[0].scatter(X_train[:, 0], X_train[:, 1], c=color, label='1')
ax[0].title.set_text('Train sample')
```

```python
color = ['blue' if l == 0 else 'green' for l in y_test]
ax[1].scatter(X_test[:, 0], X_test[:, 1], c=color, label='1');
ax[1 ].title.set_text('Test sample')
```

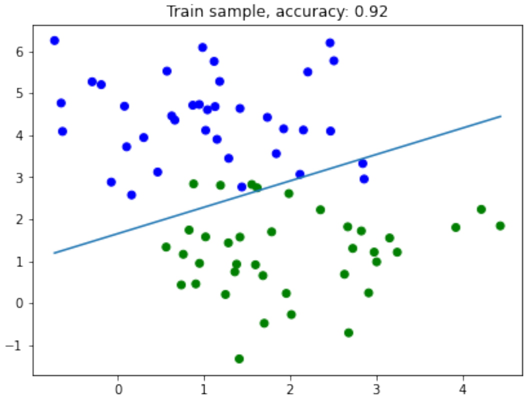


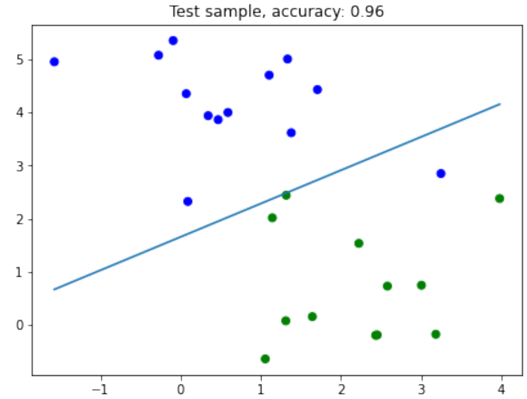
[8]:

```python
model = LogisticRegression()
model.fit(X_train, y_train);
```

[9]:

```python
fig, ax = plt.subplots(1, 2, figsize=(15,5))

x_values = np.linspace(X_train[:, 0].min(), X_train[:, 0].max(), 100)
y_values = [(-model.bias - model.weights[0]*x) / model.weights[1] for x in␣
 ↪x_values]
color = ['blue' if l == 0 else 'green' for l in y_train]
ax[0].scatter(X_train[:, 0], X_train[:, 1], c=color, label='1')
ax[0].plot(x_values, y_values)
ax[0].title.set_text('Train sample, accuracy: {}'.
 ↪format(accuracy_score(y_train, model.predict(X_train))))

x_values = np.linspace(X_test[:, 0].min(), X_test[:, 0].max(), 100)
y_values = [(-model.bias - model.weights[0]*x) / model.weights[1] for x in␣
 ↪x_values]
color = ['blue' if l == 0 else 'green' for l in y_test]
ax[1].scatter(X_test[:, 0], X_test[:, 1], c=color, label='1');
ax[1].plot(x_values, y_values)
ax[1].title.set_text('Test sample, accuracy: {}'.format(accuracy_score(y_test,␣
 ↪model.predict(X_test))))
```


Train sample, accuracy: 0.92
6
5
4
3
2
1
0
−1
0
1
2
3
4


Test sample, accuracy: 0.96
5
4
3
2
1
0
−1
0
1
2
3
4