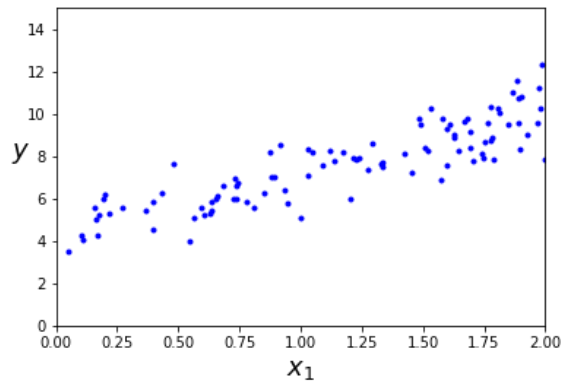
Artificial Intelligence Assignment 1
1번 : 선형회귀
20132651 Sungjae Lee

0. 데이터셋의 생성

In [1]:
```
import numpy as np
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
# 선형회귀를 위한 numpy, 시각화를 위한 matplotlib 패키지를 가져옵니다
```

In [2]:
```
X = 2*np.random.rand(100, 1)
y = 4 + 3*X + np.random.randn(100, 1)
# 특징 X 와 목표값 y 를 선형성이 있는 난수 집합으로 생성합니다
```

In [3]:
```
# 1. 화면 출력 확인
plt.plot(X, y, "b.")
plt.xlabel("$x_1$", fontsize = 18)
plt.ylabel("$y$", rotation = 0, fontsize = 18)
plt.axis([0, 2, 0, 15])
plt.show()
# X 와 y 에 대한 산점도 그래프를 그립니다
```



1. 정규 방정식을 사용한 선형회귀 접근

In [4]:
```
X_b = np.c_[np.ones((100, 1)), X]
theta_best = np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(y)
# numpy 의 linalg 를 사용하여 선형회귀를 진행합니다. 최적의 theta 값을 구합니다
```

In [5]:
```
# 2. theta_best 출력 확인
print(theta_best)
```

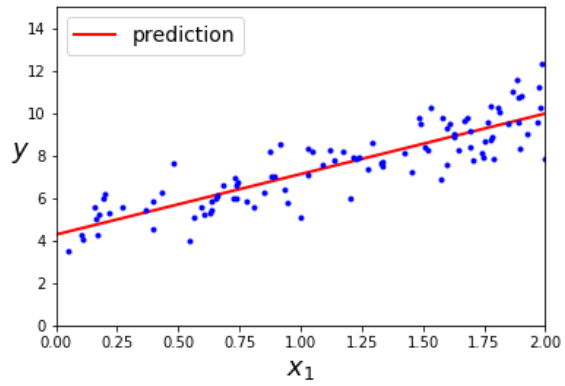
```
[[4.2803971]
 [2.8529455]]
```

In [6]:
```
X_new = np.array([[0], [2]])
X_new_b = np.c_[np.ones((2, 1)), X_new]
y_predict = X_new_b.dot(theta_best)
# X 값이 0 이거나 2 일때 y 값을 예측하여 y_predict 에 저장합니다
```

In [7]:
```
# 3. y_predict 출력 확인
print(y_predict)
```

```
[[4.2803971 ]
 [9.98628811]]
```

In [8]:
```
# 4. 화면 출력 확인
plt.plot(X_new, y_predict, "r-", linewidth = 2, label = "prediction")
plt.plot(X, y, "b.")
plt.xlabel("$x_1$", fontsize = 18)
plt.ylabel("$y$", rotation = 0, fontsize = 18)
plt.legend(loc = "upper left", fontsize = 14)
plt.axis([0, 2, 0, 15])
plt.show()
# 두 점에 대한 예측 y_predict 와 테스트 X값 X_new 를 통해 직선을 그립니다
# 이 직선은 X 를 특성으로 y 값을 예측하는 선형회귀 직선입니다
```



In [9]:
```
from sklearn.linear_model import LinearRegression
# 이번에는 sklearn 의 linear regression 패키지를 이용하여 선형회귀를 구현합니다
```

In [10]:
```
lin_reg = LinearRegression()
lin_reg.fit(X, y)
# fit 메소드를 이용해 X 특성에 대한 y 목표값을 예측합니다.
```

Out[10]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

In [11]:
```
# 5. lin_reg.intercept_, lin_reg.coef_ 출력 확인
print(lin_reg.intercept_, lin_reg.coef_)
# 생성된 LinearRegression 직선의 bias 와 weight 를 출력합니다
```

```
[4.2803971] [[2.8529455]]
```

In [12]:
```
# 6. lin_reg.predict(X_new) 출력 확인
print(lin_reg.predict(X_new))
# 해당 모델로 앞에 나온 테스트 데이터 X_new 를 예측합니다
```

```
[[4.2803971 ]
 [9.98628811]]
```

In [13]:
```
theta_best_svd, redisuals, rank, s = np.linalg.lstsq(X_b, y, rcond = 1e-6)
# 최적의 theta 값을 찾습니다
```

In [14]:
```
# 7. theta_best_svd 출력 확인
print(theta_best_svd)
```

```
[[4.2803971]
 [2.8529455]]
```

In [15]:
```
# 8. np.linalg.pinv(X_b).dot(y) 출력 확인
print(np.linalg.pinv(X_b).dot(y))
# 테스트 데이터 X_b 에 대한 theta 를 출력합니다
```

```
[[4.2803971]
 [2.8529455]]
```

2. 경사 하강법을 사용한 선형회귀 접근

In [16]:
```
eta = 0.1
n_iterations = 1000
m = 100
theta = np.random.randn(2, 1)
for iteration in range(n_iterations):
    gradients = 2/m * X_b.T.dot(X_b.dot(theta) - y)
    theta = theta - eta * gradients
# 최적값을 찾기 위해 학습률이 0.1인 Gradient Descent를 1000회 반복합니다
```

In [17]:
```
# 9. theta 출력 확인
print(theta)
# 최적의 theta 값을 출력합니다
```

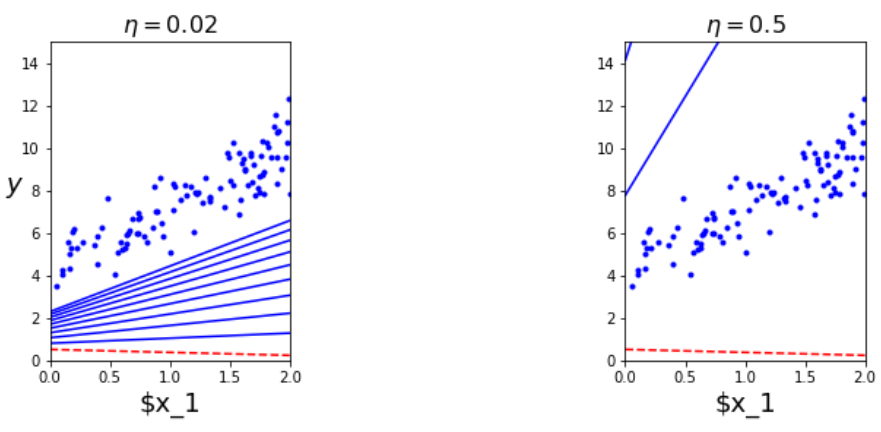
```
[[4.2803971]
 [2.8529455]]
```

In [18]:
```
print(X_new_b.dot(theta))
# 앞에서의 테스트셋 X_new_b 를 예측합니다
```

```
[[4.2803971 ]
 [9.98628811]]
```

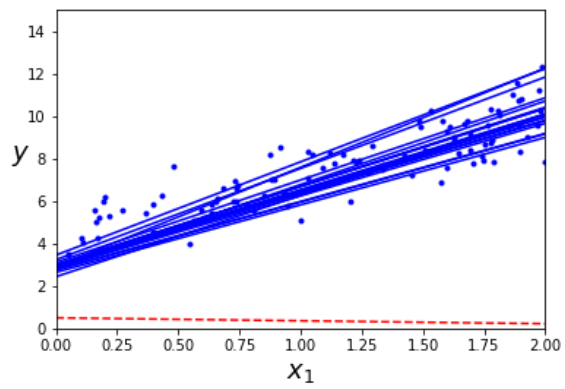
In [19]:
```
theta_path_bgd = []
def plot_gradient_descent(theta, eta, theta_path = None):
    m = len(X_b)
    plt.plot(X, y, "b.")
    n_iterations = 1000
    for iteration in range(n_iterations):
        if iteration < 10:
            y_predict = X_new_b.dot(theta)
            style = "b-" if iteration > 0 else "r--"
            plt.plot(X_new, y_predict, style)
        gradients = 2/m * X_b.T.dot(X_b.dot(theta) - y)
        theta = theta - eta * gradients
        if theta_path is not None:
            theta_path.append(theta)
    plt.xlabel("$x_1", fontsize = 18)
    plt.axis([0, 2, 0, 15])
    plt.title(r"$\eta = {}$".format(eta), fontsize = 16)
# theta 값의 변화에 따른 선형회귀 직선의 변화를 살펴보기 위해 함수를 생성합니다
```

In [20]:
```
# 11. 화면 출력 확인
np.random.seed(42)
theta = np.random.randn(2, 1)
plt.figure(figsize = (10, 4))
plt.subplot(131)
plot_gradient_descent(theta, eta = 0.02)
plt.ylabel("$y$", rotation = 0, fontsize = 18)
plt.subplot(133)
plot_gradient_descent(theta, eta = 0.5)
plt.show()
# 학습률이 0.02 일 때와 0.5 일 때의 선형회귀 직선의 이동 변화를 볼 수 있습니다
# 학습률이 너무 작으면 시간이 너무 오래걸리며, 학습률이 너무 높으면 수렴하지 못합니다
```



3. 스토캐스틱 경사 하강법을 사용한 선형회귀 접근

In [21]:
```
# 12. 화면 출력 확인
theta_path_sgd = []
m = len(X_b)
np.random.seed(42)
n_epochs = 50
t0, t1 = 5, 50
def learning_schedule(t):
    return t0/(t+t1)
theta = np.random.randn(2, 1)
for epoch in range(n_epochs):
    for i in range(m):
        if epoch == 0 and i < 20:
            y_predict = X_new_b.dot(theta)
            style = "b-" if i > 0 else "r--"
            plt.plot(X_new, y_predict, style)
        random_index = np.random.randint(m)
        xi = X_b[random_index:random_index + 1]
        yi = y[random_index:random_index + 1]
        gradients = 2 * xi.T.dot(xi.dot(theta) - yi)
        eta = learning_schedule(epoch * m + i)
        theta = theta - eta * gradients
        theta_path_sgd.append(theta)

plt.plot(X, y, "b.")
plt.xlabel("$x_1$", fontsize = 18)
plt.ylabel("$y$", rotation = 0, fontsize = 18)
plt.axis([0, 2, 0, 15])
plt.show()
# 스토캐스틱 경사 하강법을 이용하여 특정 샘플에 대해서 Gradient 를 계산하고 경사 하강을 진행합니다
# 이를 통해 배치 경사 하강 알고리즘에 비해 빠른 속도로 경사 하강이 진행됩니다
```



In [22]:
```
# 13. theta 출력 확인
print(theta)
# 최종적인 theta 값을 출력합니다
```

```
[[4.27380642]
 [2.86748498]]
```

In [23]:
```
from sklearn.linear_model import SGDRegressor
sgd_reg = SGDRegressor(max_iter = 50, penalty = None, eta0 = 0.1, random_state = 42)
# sklearn 패키지의 스토캐스틱 경사 하강 알고리즘을 가져옵니다
```

In [24]:
```
# 14. sgd_reg.fit(X, y.ravel()) 출력 확인
sgd_reg.fit(X, y.ravel())
# 가져온 모델을 X 와 y 를 이용하여 학습시킵니다
```

Out[24]:
```
SGDRegressor(alpha=0.0001, average=False, epsilon=0.1, eta0=0.1,
       fit_intercept=True, l1_ratio=0.15, learning_rate='invscaling',
       loss='squared_loss', max_iter=50, n_iter=None, penalty=None,
       power_t=0.25, random_state=42, shuffle=True, tol=None, verbose=0,
       warm_start=False)
```

In [25]:
```
# 15. sgd_reg.intercept_, sgd_reg.coef_ 출력 확인
print(sgd_reg.intercept_)
print(sgd_reg.coef_)
# 스토캐스틱 경사 하강 알고리즘을 통해 만들어진 모델의 bias 와 weight 를 출력합니다
```

```
[4.29513019]
[2.88698744]
```

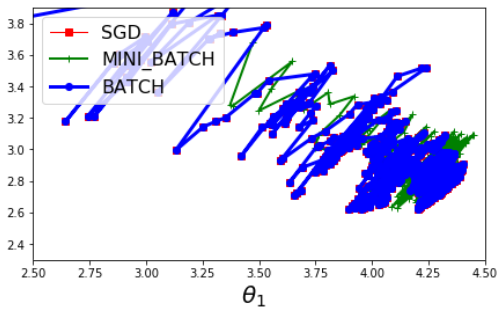
In [26]:
```
theta_path_mgd = []
n_iterations = 50
minibatch_size = 20
np.random.seed(42)
theta = np.random.randn(2, 1)
t0, t1 = 200, 1000
def learning_schedule(t):
    return t0/(t+t1)
t = 0
for epoch in range(n_iterations):
    shuffled_indices = np.random.permutation(m)
    X_b_shuffled = X_b[shuffled_indices]
    y_shuffled = y[shuffled_indices]
    for i in range(0, m, minibatch_size):
        t += 1
        xi = X_b_shuffled[i:i + minibatch_size]
        yi = y_shuffled[i:i + minibatch_size]
        gradients = 2/minibatch_size * xi.T.dot(xi.dot(theta) - yi)
        eta = learning_schedule(t)
        theta = theta - eta * gradients
        theta_path_mgd.append(theta)
# 미니 배치 경사하강법을 이용하여 모델을 생성합니다
# 이 때 최적의 theta 값을 저장합니다
```

In [27]:
```
# 16. theta 출력 확인
print(theta)
# 미니 배치 경사하강법의 최적 theta 값을 출력합니다
```

```
[[4.22470385]
 [2.84780161]]
```

In [28]:
```
theta_path_bgd = np.array(theta_path_bgd)
theta_path_sgd = np.array(theta_path_sgd)
theta_path_mgd = np.array(theta_path_mgd)
# 배치 경사 하강 / 스토캐스틱 경사 하강 / 미니 배치 경사 하강의 theta 진행을 저장합니다
```

In [29]:
```
plt.figure(figsize = (7, 4))
plt.plot(theta_path_sgd[:, 0], theta_path_sgd[:, 1], "r-s", linewidth = 1, label =
"SGD") plt.plot(theta_path_mgd[:, 0], theta_path_mgd[:, 1], "g-+", linewidth = 2,
label = "MINI_B ATCH")
plt.plot(theta_path_sgd[:, 0], theta_path_sgd[:, 1], "b-o", linewidth = 3, label =
"BATCH" )
plt.legend(loc = "upper left", fontsize = 16)
plt.xlabel(r"$\theta_0$", fontsize = 20)
plt.xlabel(r"$\theta_1$  ", fontsize = 20, rotation = 0)
plt.axis([2.5, 4.5, 2.3, 3.9])
plt.show()
# 각각 의 방법 이 어떤 경로로 최적의 theta 값을 찾아가는지 그래프로 표현합니다
```

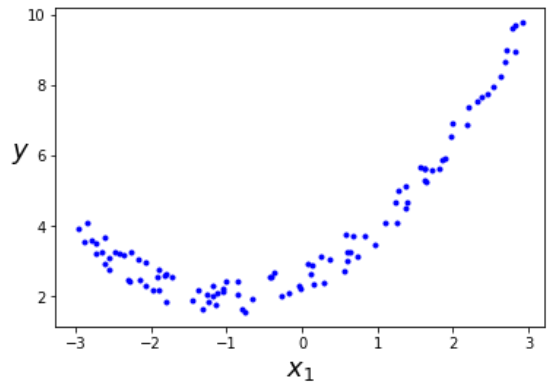


Artificial Intelligence Assignment 1
2번 : 다차항회귀
20132651 Sungjae Lee

In [1]:
```
import numpy as np
import numpy.random as rnd
import matplotlib.pyplot as plt
# 다차항회귀를 위한 numpy 와 시각화를 위한 matplotlib 패키지를 가져옵니다
```

In [2]:
```
np.random.seed(42)
m = 100
X = 6 * np.random.rand(m, 1) - 3
y = 0.5 * X**2 + X + 2 + np.random.rand(m, 1)
# 비선형 데이터를 난수를 활용하여 X 와 y로 생성합니다
```

In [3]:
```
# 1. 화면 출력 확인
plt.plot(X, y, "b.")
plt.xlabel("$x_1$", fontsize = 18)
plt.ylabel("$y$", rotation = 0, fontsize = 18)
plt.show()
# 생성한 데이터를 산점도 그래프로 표현합니다
```



In [4]:
```
from sklearn.preprocessing import PolynomialFeatures
# sklearn 의 PolynomialFeatures 를 활용하여 다차항회귀를 진행합니다
```

In [5]:
```
poly_features = PolynomialFeatures(degree = 2, include_bias = False)
X_poly = poly_features.fit_transform(X)
```

In [6]:
```
# 2. X[0] 출력 확인
print(X[0])
```

```
[-0.75275929]
```

In [7]:
```
# 3. X_poly[0] 출력 확인
print(X_poly[0])
```

```
[-0.75275929  0.56664654]
```

In [8]:
```
from sklearn.linear_model import LinearRegression
```

In [9]:
```
lin_reg = LinearRegression()
lin_reg.fit(X_poly, y)
```

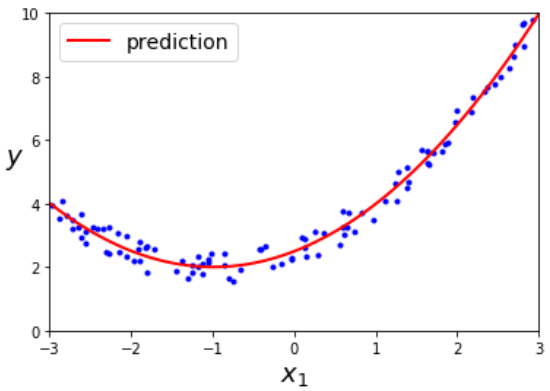
Out[9]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

In [10]:
```
# 4. lin_reg.intercept_, lin_reg.coef_ 출력 확인
print(lin_reg.intercept_)
print(lin_reg.coef_)
```

```
[2.49786712]
[[0.9943591  0.49967213]]
```

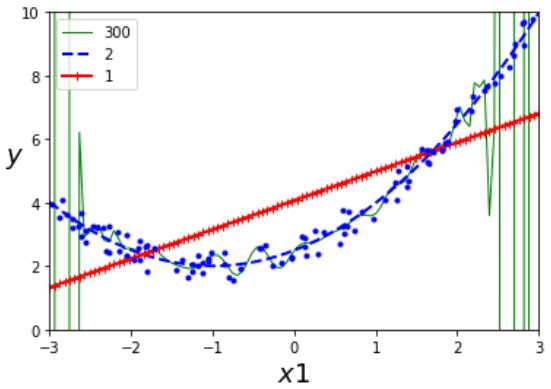
In [11]:
```
X_new = np.linspace(-3, 3, 100).reshape(100, 1)
X_new_poly = poly_features.transform(X_new)
y_new = lin_reg.predict(X_new_poly)
```

In [12]:
```
# 5. 화면 출력 확인
plt.plot(X, y, "b.")
plt.plot(X_new, y_new, "r-", linewidth = 2, label = "prediction")
plt.xlabel("$x_1$", fontsize = 18)
plt.ylabel("$y$", rotation = 0, fontsize = 18)
plt.legend(loc = "upper left", fontsize = 14)
plt.axis([-3, 3, 0, 10])
plt.show()
# 생성된 예측 곡선과 데이터를 비교하는 그래프를 그립니다
```



In [13]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
```

In [14]:
```
# 6. 화면 출력 확인
for style, width, degree in (("g-", 1, 300), ("b--", 2, 2), ("r-+", 2, 1)):
    polybig_features = PolynomialFeatures(degree=degree, include_bias=False)
    std_scaler = StandardScaler()
    lin_reg = LinearRegression()
    polynomial_regression = Pipeline([
        ("poly_features", polybig_features),
        ("std_scaler", std_scaler),
        ("lin_reg", lin_reg),
    ])
    polynomial_regression.fit(X, y)
    y_newbig = polynomial_regression.predict(X_new)
    plt.plot(X_new, y_newbig, style, label = str(degree), linewidth = width)
plt.plot(X, y, "b.", linewidth = 3)
plt.legend(loc = "upper left")
plt.xlabel("$x1$", fontsize = 18)
plt.ylabel("$y$", rotation = 0, fontsize = 18)
plt.axis([-3, 3, 0, 10])
plt.show()
# 차수가 1차, 2차, 300차 일 때 과소적합 및 과대적합 여부를 봅니다
# 2차 곡선을 이용한 예측이 가장 정확하게 데이터를 표현하는 점을 볼 수 있습니다
# 300차 곡선에서는 과대적합이, 1차 곡선에서는 과소적합이 발생합니다
```

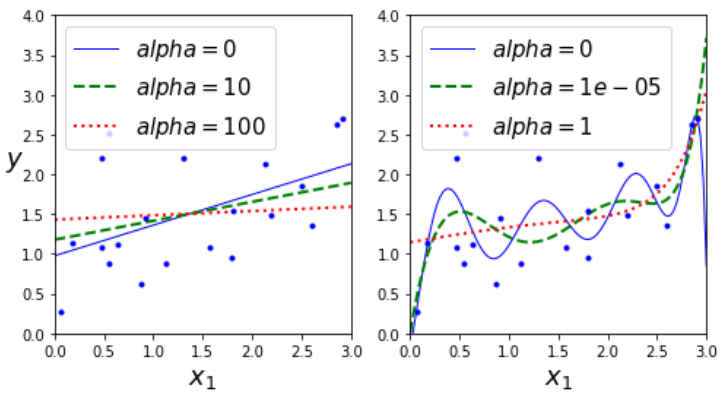


Artificial Intelligence Assignment 1
3번 : 규제
20132651 Sungjae Lee

In [1]:
```
import numpy as np
from sklearn.linear_model import Ridge
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures
from sklearn.preprocessing import StandardScaler

import matplotlib.pyplot as plt
%matplotlib inline
# 앞에서 사용한 numpy 와 sklearn, matplotlib 패키지를 가져옵니다
```

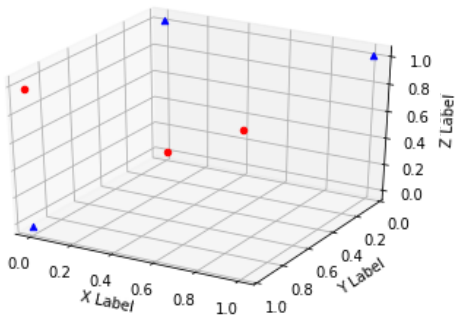
In [2]:

```
# 1. 화면 출력 확인
np.random.seed(42)
m = 20
X = 3 * np.random.rand(m, 1)
y = 1 + 0.5 * X + np.random.randn(m, 1)/1.5
X_new = np.linspace(0, 3, 100).reshape(100, 1)
def plot_model(model_class, polynomial, alphas, **model_kargs):
    for alpha, style in zip(alphas, ("b-", "g--", "r:")):
        model = model_class(alpha, **model_kargs) if alpha > 0 else LinearRegression()
        if polynomial:
            model = Pipeline([
                ("poly_features", PolynomialFeatures(degree = 10, include_bias = False)),
                ("std_scaler", StandardScaler()),
                ("regul_reg", model),
            ])
        model.fit(X, y)
        y_new_regul = model.predict(X_new)
        lw = 2 if alpha > 0 else 1
        plt.plot(X_new, y_new_regul, style, linewidth = lw, label = r"$alpha = {}$".format
(alpha))
    plt.plot(X, y, "b.", linewidth = 3)
    plt.legend(loc = "upper left", fontsize = 15)
    plt.xlabel("$x_1$", fontsize = 18)
    plt.axis([0, 3, 0, 4])

plt.figure(figsize = (8, 4))
plt.subplot(121)
plot_model(Ridge, polynomial = False, alphas = (0, 10, 100), random_state = 42)
plt.ylabel("$y$", rotation = 0, fontsize = 18)
plt.subplot(122)
plot_model(Ridge, polynomial = True, alphas = (0, 10**-5, 1), random_state = 42)
plt.show()
# 규제에 따른 예측선의 변화를 그래프를 이용해 살펴봅니다
# 알파값이 0일 때 과적합에 가까워지며 알파값이 높아지면 규제가 강해지는 것으로 보입니다
# 즉, 적절한 알파값을 활용하여 최적의 데이터 예측 모델을 생성하는 것이 중요합니다
```



Artificial Intelligence Assignment 1
4번 : 선형분리
20132651 Sungjae Lee

In [1]:
```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
np.random.seed(42)

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(0, 0, 0, c='r', marker = 'o')
ax.scatter(0, 1, 1, c='r', marker = 'o')
ax.scatter(1, 1, 1, c='r', marker = 'o')
ax.scatter(1, 0, 1, c='b', marker = '^')
ax.scatter(0, 0, 1, c='b', marker = '^')
ax.scatter(0, 1, 0, c='b', marker = '^')

ax.set_xlabel('X Label')
ax.set_ylabel('Y Label')
ax.set_zlabel('Z Label')

ax.set_ylim(1, 0)

plt.show()
# 6개의 점을 3차원 공간에 표현한 것으로, 각각의 점은 결과값에 따라 파란색 세모와 빨간색 원으로 구분하였습니다
```



선형분리 가능 여부와 그 이유

위의 훈련집합은 선형분리가 불가능합니다. 이를 간단하게 설명하기 위해서는 좌측 평면의 네 점을 살펴볼 수 있습니다. Y 축과 Z 축으로 이루어진 해당 평면은 (Y, Z) 의 평면으로 보았을 때, (0, 0), (1, 1) 의 빨간점과 (0, 1), (1, 0)의 파란점이 함께 존재하는 상황입니다. 이는 XOR 문제와 동일한 형태이며, 선형분리가 불가능함을 볼 수 있습니다. 평면적으로 선형분리가 불가능한 상태에서는 이를 확장하여 3차원 공간으로 만들었을 때 초평면을 이용한 분리가 불가능하다는 추론이 가능합니다. 그러므로 해당 훈련집합은 선형분리가 불가능함을 알 수 있습니다.

Artificial Intelligence Assignment 1
5번 : 행렬
20132651 Sungjae Lee

In [1]:
```
import numpy as np
```

In [2]:
```
A = np.matrix('1 -2 3 5; 2 2 -1 0; 3 0 1 2; 1 0 2 0')
print(A)
```

```
[[ 1 -2  3  5]
 [ 2  2 -1  0]
 [ 3  0  1  2]
 [ 1  0  2  0]]
```

In [3]:
```
# 2 * A 의 출력
2*A
```

Out[3]:
```
matrix([[ 2, -4,  6, 10],
        [ 4,  4, -2,  0],
        [ 6,  0,  2,  4],
        [ 2,  0,  4,  0]])
```

In [4]:
```
# A 의 전치행렬 Transposed Matrix 의 출력
print(A.T)
```

```
[[ 1  2  3  1]
 [-2  2  0  0]
 [ 3 -1  1  2]
 [ 5  0  2  0]]
```

In [5]:
```
# A 의 역행렬 Inversed Matrix 의 출력
print(A.I)
```

```
[[-0.23529412 -0.23529412  0.58823529 -0.05882353]
 [ 0.29411765  0.79411765 -0.73529412  0.32352941]
 [ 0.11764706  0.11764706 -0.29411765  0.52941176]
 [ 0.29411765  0.29411765 -0.23529412 -0.17647059]]
```

In [6]:
```
# A 행렬의 계수 rank 의 출력
from numpy.linalg import matrix_rank
print(matrix_rank(A))
```

```
4
```

In [7]:
```
# A 행렬의 행렬식 determinant 의 출력
from numpy.linalg import det
print(det(A))
```

```
34.00000000000001
```

In [8]:
```
# A 행렬의 고유 분해 eigen decomposition 의 출력
from numpy.linalg import eig
print(eig(A))
```

```
(array([ 5.52552524+0.j        , -1.52204833+1.31733645j,
       -1.52204833-1.31733645j,  1.51857142+0.j        ]), matrix([[-0.68465996+0.j        ,  0.619935  +0.j        ,
          0.619935  -0.j        ,  0.11199539+0.j        ],
        [-0.21666012+0.j        , -0.34875983-0.24694874j,
         -0.34875983+0.24694874j, -0.93998928+0.j        ],
        [-0.60547918+0.j        , -0.31379356-0.41033136j,
         -0.31379356+0.41033136j, -0.22854692+0.j        ],
        [-0.34306572+0.j        , -0.263929  +0.31075191j,
         -0.263929  -0.31075191j, -0.22725204+0.j        ]]))
```

In [9]:
```
# A 의 특이값 분해 singular value decomposition 의 출력
from numpy.linalg import svd
print(svd(A))
```

```
(matrix([[-0.8819521 ,  0.25706354, -0.24178316, -0.31244156],
        [ 0.05743106, -0.76320224, -0.26028922, -0.58862003],
        [-0.42722253, -0.58124436,  0.04756433,  0.69092224],
        [-0.19063859, -0.11660051,  0.93355667, -0.28023773]]), array([6.97059301, 3.8068416 , 1.85521105, 0.69063894]), matrix
([[-0.32126336,  0.26952747, -0.50380033, -0.75520197],
        [-0.8221184 , -0.53601693,  0.18911937,  0.03226532],
        [ 0.16919264, -0.01995035,  0.78137601, -0.60035603],
        [ 0.43850318, -0.7997767 , -0.31602313, -0.2611543 ]]))
```

Artificial Intelligence Assignment 1

6번 : 놈

20132651 Sungjae Lee

In [1]:
```
import numpy as np
from numpy.linalg import norm
```

In [2]:
```
# x의 1차, 2차, 3차 놈과 최대 놈 계산
x = np.array([3, -4, -1.2, 0, 2.3])

print(norm(x, 1))
print(norm(x, 2))
print(norm(x, 3))
print(norm(x, np.inf))
```

```
10.5
5.632938842203065
4.716120891176797
4.0
```

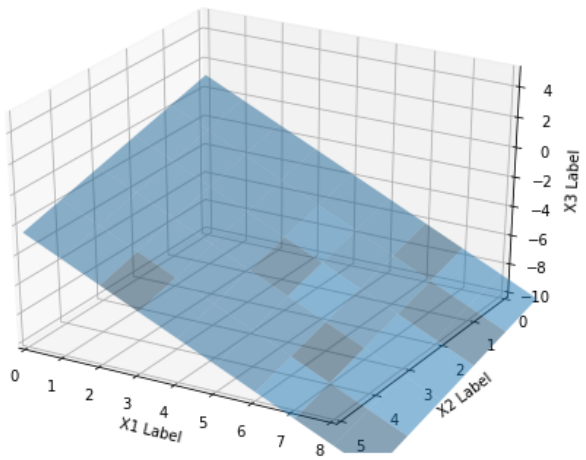
In [3]:
```
# x2의 프로베니우스 놈 계산
x2 = np.matrix('2 1; 1 5; 4 1')

print(norm(x2))
```

```
6.928203230275509
```

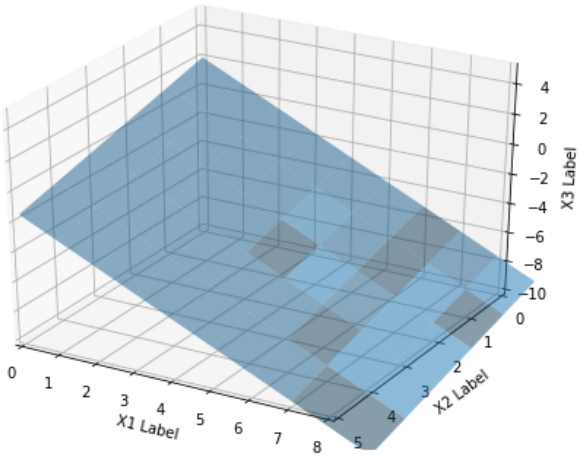
Artificial Intelligence Assignment 1
7번 : 퍼셉트론
20132651 Sungjae Lee

In [1]:
```
# 2-3 의 수식 : w = (1.2, 0.7, 1.0) // 임계값 T는 1.0 // 1.2x1 + 0.7x2 + 1.0x3 = 1.0
```

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
%matplotlib inline

x1, x2 = np.meshgrid(range(10), range(6))

x3 = 1.0 -1.2 * x1 - 0.7 * x2

plt3d = plt.figure(figsize=(8, 6)).gca(projection='3d')
plt3d.plot_surface(x1, x2, x3, alpha=0.5)

ax = plt.gca()

ax.set_xlabel('X1 Label')
ax.set_ylabel('X2 Label')
ax.set_zlabel('X3 Label')

ax.set_xlim(0, 8)
ax.set_ylim(5, 0)
ax.set_zlim(-10, 5)

plt.show()
# T = 1.0 에서의 결정평면을 3차원 공간에 그려보면 다음과 같습니다
```



In [3]:
```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
%matplotlib inline

x1, x2 = np.meshgrid(range(10), range(6))

x3 = 2.0 -1.2 * x1 - 0.7 * x2

plt3d = plt.figure(figsize=(8, 6)).gca(projection='3d')
plt3d.plot_surface(x1, x2, x3, alpha=0.5)

ax = plt.gca()

ax.set_xlabel('X1 Label')
ax.set_ylabel('X2 Label')
ax.set_zlabel('X3 Label')

ax.set_xlim(0, 8)
ax.set_ylim(5, 0)
ax.set_zlim(-10, 5)

plt.show()
# T = 2.0 에서의 결정평면을 3차원 공간에 그려보면 다음과 같습니다
# 차이점은 X3 축과 닿는 평면이 약간 상승했다는 점입니다. 이는 원점과 평면의 거리가 증가하며 발생한 것으로 보입니다
```

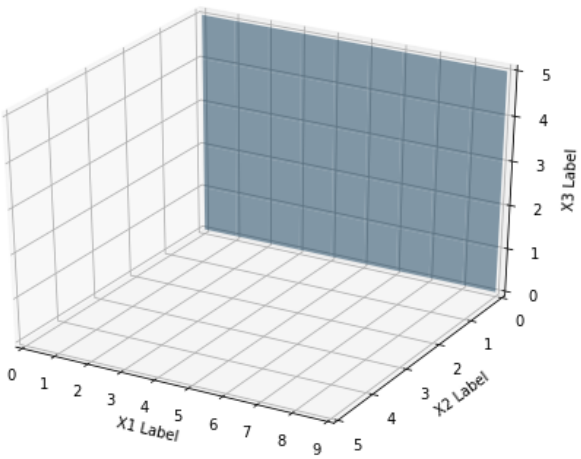


In [4]:
```
import numpy as np
from numpy.linalg import norm

# x의 1차, 2차, 3차 놈과 최대 놈 계산
x = np.array([1.2, 0.7, 1.0])
T = 1.0
T2 = 2.0
print('T = 1.0일 때, 거리 :',T/norm(x, 2))
print('T = 2.0일 때, 거리 :',T2/norm(x, 2))
# 임곗값의 변화로 인한 평면의 변화를 확인하기 위해 norm 과 임곗값 T 를 기반으로 원점과 평면의 거리를 계산하였습니다.
```

```
T = 1.0일 때, 거리 : 0.584206237836986
T = 2.0일 때, 거리 : 1.168412475673972
```

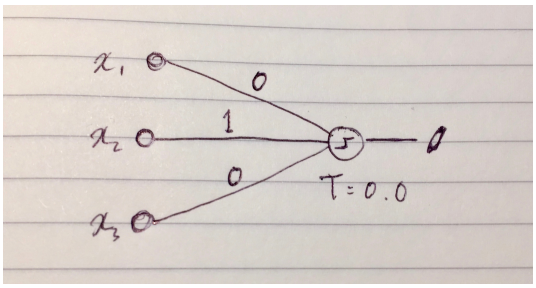
Artificial Intelligence Assignment 1
8번 : 결정평면_퍼셉트론
20132651 Sungjae Lee

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
%matplotlib inline

x1, x3 = np.meshgrid(range(10), range(6))

x2 = 0 * x1

plt3d = plt.figure(figsize=(8, 6)).gca(projection='3d')
plt3d.plot_surface(x1, x2, x3, alpha=0.5)

ax = plt.gca()

ax.set_xlabel('X1 Label')
ax.set_ylabel('X2 Label')
ax.set_zlabel('X3 Label')

ax.set_xlim(0, 9)
ax.set_ylim(5, 0)

plt.show()
# 해당 결정평면을 3차원 공간에 그려보면 다음과 같습니다
```



퍼셉트론의 weight 와 임곗값 T

위의 결정평면은 w = (0, 1, 0)^T 이고, T = 0.0 인 퍼셉트론에 의해 만들어진 것입니다. 그 이유는 우선 X1, X3 의 값에 상관없이 결정이 진행된다는 점이며, 결과적으로 X2 값이 0보다 클 때와 작을 때에 의해 y 값이 구분되기 때문입니다. 이를 퍼셉트론 구조로 표현하면 다음과 같습니다



Artificial Intelligence Assignment 1

9번 : 합성함수

20132651 Sungjae Lee

NO.

(1) 식 2.53에 따라 $i(x)$와 $h(x)$를 쓰시오.

$f(x) = g(h(x))$라 하면,

$h(x) = \frac{1}{4}(1-2x)^2 - 1$ 이다.

$f(x) = g(h(i(x)))$라 하면,

$i(x) = 1-2x$ 이다.

(2) 연쇄법칙을 이용하여 $f'(x)$를 구하시오.

연쇄법칙에 의하면 $f'(x) = g'(h(i(x))) \cdot h'(i(x)) \cdot i'(x)$

이므로, $g'(h(i(x)))$와 $h'(i(x))$와 $i'(x)$를 각각 구해 곱한다.

$h(i(x)) = A$로 치환하면,

$g'(h(i(x))) = g'(A) = 6A^2 - 6A$ 이다.

$h'(i(x)) = \frac{1}{2}(1-2x)$ 이다.

$i'(x) = -2$ 이다.

즉, $f'(x) = \{6(\frac{1}{4}(1-2x)^2-1)^2 - 6(\frac{1}{4}(1-2x)^2-1)\}\{\frac{1}{2}(1-2x)\}\{-2\}$

여기서 $\frac{1}{4}(1-2x)^2 = B$로 치환하면,

$f'(x) = \{6(B-1)^2 - 6(B-1)\}(2x-1)$

$= 6 \cdot (2x-1)(B^2 - 2B + 1 - B + 1)$

$= 6 \cdot (2x-1)(B^2 - 3B + 2)$

$= 6 \cdot (2x-1)(x^2 - x - \frac{7}{4})(x^2 - x - \frac{3}{4})$

$= 6 \cdot (2x-1)(x - \frac{3}{2})(x + \frac{1}{2})(x^2 - x - \frac{7}{4})$

(3) $f'(0) = -7.875$

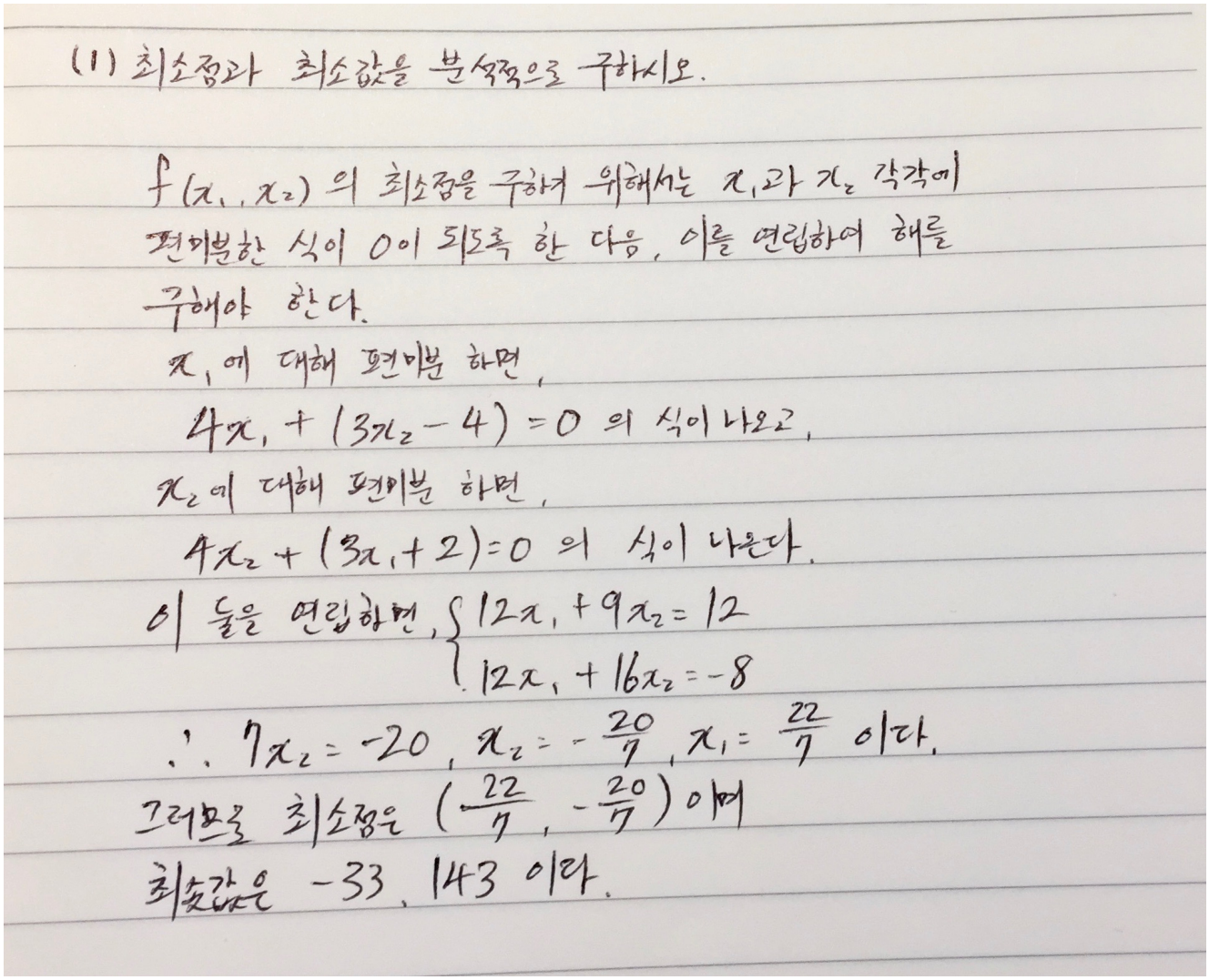
$f'(2.1) = 16.443$

morning glory

Artificial Intelligence Assignment 1

10번 : 경사하강법

20132651 Sungjae Lee



(1) 최소점과 최소값을 분석적으로 구하시오.

$f(x_1, x_2)$의 최소점을 구하기 위해서는 x_1과 x_2 각각에 편미분한 식이 0이 되도록 한 다음, 이를 연립하여 해를 구해야 한다.

x_1에 대해 편미분 하면,

$4x_1 + (3x_2 - 4) = 0$ 의 식이 나오고,

x_2에 대해 편미분 하면,

$4x_2 + (3x_1 + 2) = 0$ 의 식이 나온다.

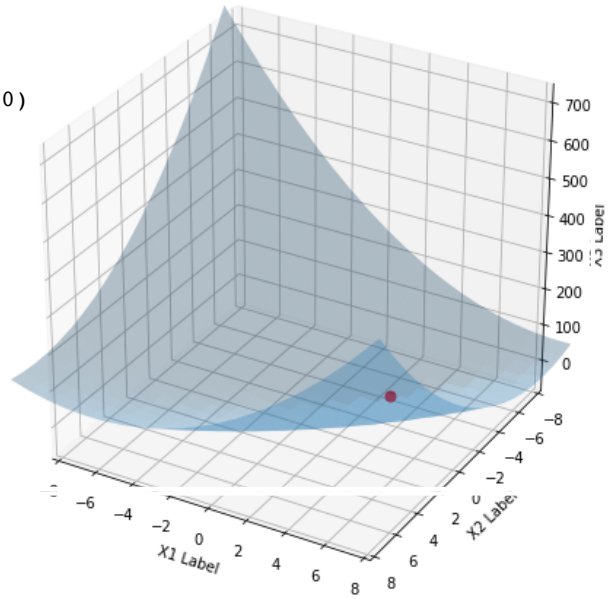
이 둘을 연립하면, $\begin{cases} 12x_1 + 9x_2 = 12 \\ 12x_1 + 16x_2 = -8 \end{cases}$

$\therefore 7x_2 = -20$, $x_2 = -\frac{20}{7}$, $x_1 = \frac{22}{7}$ 이다.

그러므로 최소점은 $(\frac{22}{7}, -\frac{20}{7})$ 이며

최솟값은 -33.143 이다.

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
%matplotlib inline

x1, x2 = np.meshgrid(range(-10, 10), range(-10, 10))
x3 = 2*(x1**2) + 3*x1*x2 +2*(x2**2) - 4*x1 + 2*x2 - 24

plt3d = plt.figure(figsize=(8, 8)).gca(projection='3d')
plt3d.plot_surface(x1, x2, x3, alpha=0.3)

ax = plt.gca()

ax.scatter(22/7, -20/7, -33.14, color='red', s = 50)

ax.set_xlabel('X1 Label')
ax.set_ylabel('X2 Label')
ax.set_zlabel('X3 Label')

ax.set_xlim(-8, 8)
ax.set_ylim(8, -8)

plt.show()
# 주어진 방정식을 3차원 공간에 그래프로 그려보았습니다
# 차후에 구하게 된 최솟값을 빨간색 점으로 표현하였습니다
```



In [2]:
```
a = (22/7)
b = (-20/7)
c = 2*(a**2) + 3*a*b +2*(b**2) - 4*a + 2*b - 24
c
# 편의를 위해 x1 축을 a, x2 축을 b, x3 축을 c 로 지정하고 진행하였습니다.
# 수식을 통해 구해진 최적의 a, b 값 (즉, 최적의 x1, x2 값) 을 수식에 대입하여
# 최저점 c (x3 값) 을 구하였습니다
```

Out[2]: -33.14285714285714

In [3]:
```
all_c = []
for i in range(-10, 10):
    a = (22 + i) / 7
    b = (-20) / 7
    c = 2*(a**2) + 3*a*b +2*(b**2) - 4*a + 2*b - 24
    all_c.append(c)
for i in range(-10, 10):
    a = (22) / 7
    b = (-20 + i) / 7
    c = 2*(a**2) + 3*a*b +2*(b**2) - 4*a + 2*b - 24
    all_c.append(c)
for i in range(-10, 10):
    a = (22 + i) / 7
    b = (-20 + i) / 7
    c = 2*(a**2) + 3*a*b +2*(b**2) - 4*a + 2*b - 24
    all_c.append(c)
# 주변값 탐색을 통해 수식의 전개 방식이 올바른 진행이었는지 검증하였습니다
```

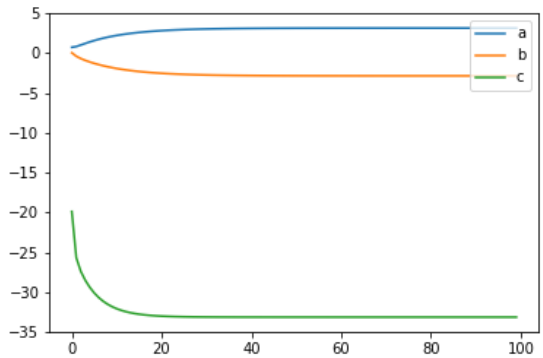
In [4]:
```
min(all_c)
# 주변값 탐색 결과 최소값이 수식 전개를 통한 최소값과 동일함을 보았습니다
```

Out[4]: -33.14285714285714

In [5]:
```
a = 1.0
b = 0.9
p = 0.1
all_a = []
all_b = []
all_c = []
all_i = []
for i in range(100):
    if i < 4:
        print('x', i, 'is', '(', a, ',', b, ')')
    c = 2*(a**2) + 3*a*b +2*(b**2) - 4*a + 2*b - 24
    a2 = p * (4*a + 3*b - 4)
    b2 = p * (4*b + 3*a + 2)
    a -= a2
    b -= b2
    all_a.append(a)
    all_b.append(b)
    all_c.append(c)
    all_i.append(i)
# 시작값 x0 부터 경사하강이 진행되는 값들 x1, x2, x3 의 a, b값을 출력하였습니다
```

```
x 0 is ( 1.0 , 0.9 )
x 1 is ( 0.73 , 0.040000000000000036 )
x 2 is ( 0.826 , -0.39499999999999996 )
x 3 is ( 1.0141 , -0.6848 )
```

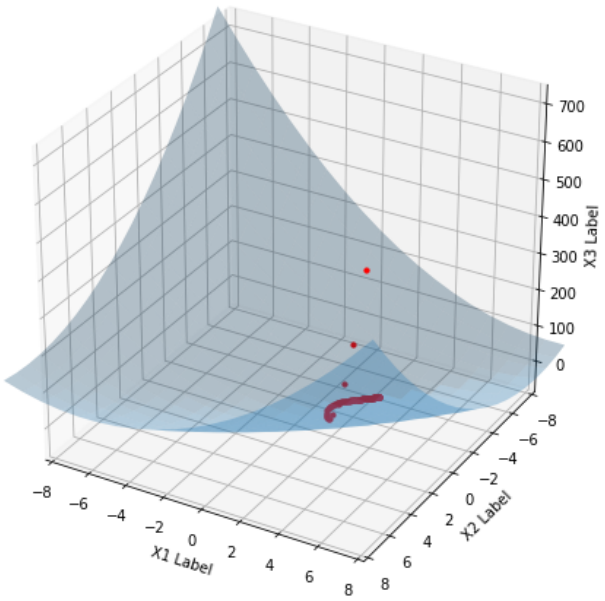
In [6]:
```
plt.plot(all_i, all_a)
plt.plot(all_i, all_b)
plt.plot(all_i, all_c)
plt.legend(['a', 'b', 'c'], loc='upper right')
plt.show()
print(all_a[-1],'/', all_b[-1],'/', all_c[-1])
# 경사하강이 진행됨에 따라 a, b, c 값이 어떻게 변화하는지 그래프로 확인하였습니다
```




```
3.142778786730424 / -2.8570645010161386 / -33.14285713527729
```

In [7]:

```
a = 10
b = 10
p = 0.03
all_a2 = []
all_b2 = []
all_c2 = []
all_i2 = []
for i in range(100):
    c = 2*(a**2) + 3*a*b +2*(b**2) - 4*a + 2*b - 24
    a2 = p * (4*a + 3*b - 4)
    b2 = p * (4*b + 3*a + 2)
    a -= a2
    b -= b2
    all_a2.append(a)
    all_b2.append(b)
    all_c2.append(c)
    all_i2.append(i)
# 시작값 x0 부터 경사하강이 진행되는 값들 x1, x2, x3 의 a, b값을 출력하였습니다

x1, x2 = np.meshgrid(range(-10, 10), range(-10, 10))
x3 = 2*(x1**2) + 3*x1*x2 +2*(x2**2) - 4*x1 + 2*x2 - 24

plt3d = plt.figure(figsize=(8, 8)).gca(projection='3d')
plt3d.plot_surface(x1, x2, x3, alpha=0.3)

ax = plt.gca()

for i in range(100):
    ax.scatter(all_a2[i], all_b2[i], all_c2[i], color='red', s = 10)

ax.set_xlabel('X1 Label')
ax.set_ylabel('X2 Label')
ax.set_zlabel('X3 Label')

ax.set_xlim(-8, 8)
ax.set_ylim(8, -8)

plt.show()
# 조금 더 나아가, 초기값이 10, 10 일 때, 3차원 공간 내에서 a, b, c 값의 변화를 점의 이동으로 살펴보았습니다
# 학습률을 0.03 으로 주어 점의 이동을 조금 더 상세하게 보았습니다
# 점이 최적점으로 점차 수렴해가는 모습을 볼 수 있었습니다
```



Artificial Intelligence Assignment 1
11번 : 베이즈 규칙
20132651 Sungjae Lee

베이즈 규칙을 활용하여 해당 문제를 풀면 다음과 같습니다.

우선 도전자는 진행자가 3번 문을 연 다음, 그대로 1번 문을 선택하거나 2번 문으로 선택을 변경합니다.

이 때, 1, 2, 3번 문에 상품이 존재할 확률을 각각 P(A1), P(A2), P(A3) 라고 할 수 있습니다.

그렇다면 베이즈 규칙에 의해 우리가 구해야 하는 것은 다음과 같습니다.

(a) : 진행자가 3번 문을 열었을 때, 1번 문을 그대로 선택하였을 때 상품이 존재할 확률
(b) : 진행자가 3번 문을 열었을 때, 2번 문으로 변경하였을 때 상품이 존재할 확률
(c) : 위의 두 확률은 동일함 (즉, (a) 와 (b) 의 확률은 50% 임)

이를 구하기 위해서는 진행자가 3번 문을 연다는 사건 B 속에서의 P(A1), P(A2), P(A3) 의 조건부 확률을 구해야 합니다.

(a) 를 기준으로 계산을 진행하면 다음과 같습니다.

P(A1|B) = { P(B|A1) *P(A1)* } / { *P(B|A1)* P(A1) + P(B|A2) *P(A2)* + *P(B|A3)* P(A3) }

수식을 설명하면 이렇습니다. P(A1|B) 는 진행자가 문을 열었을 때, 1번째 문에 상품이 존재할 확률입니다. 이는 진행자가 문을 열었을 때, 각각의 문에 상품이 존재할 확률 중에서, 1번째 문에 상품이 존재할 확률과 동일합니다. n번 문에 상품이 존재할 확률은 P(B|An) * P(An) 이므로 위와 같은 수식이 나타나게 됩니다

수식을 계산하면 다음과 같습니다.

P(B|A1) *P(A1)* == *P(B|A2)* P(A2) == P(B|A3) * P(A3) == 1/3 이므로,
(1/3) / (1/3 + 1/3 + 1/3) == (1/3) / (3/3) = 1/3
즉, (a) 의 확률, 1번 문을 그대로 선택했을 때 그 문에 상품이 존재하여 상품을 얻을 확률은 1/3 입니다

당연히 (b) 의 확률, 2번 문으로 선택을 변경하였을 때의 확률은 1 - (a) 로 2/3 입니다.
그러므로 (b) 의 확률이 높음을 유도해 내어 선택을 변경하는 (b) 의 전략을 선택하는 것이 합리적임을 알 수 있습니다