MLA - Uebung 2

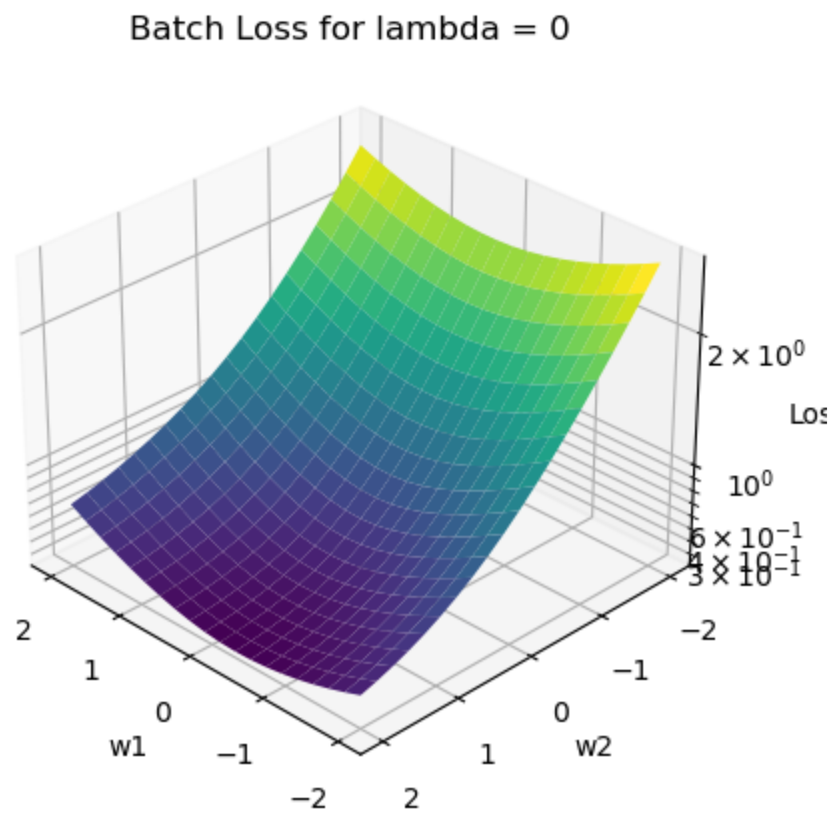
Gerald Ebmer, e01325683

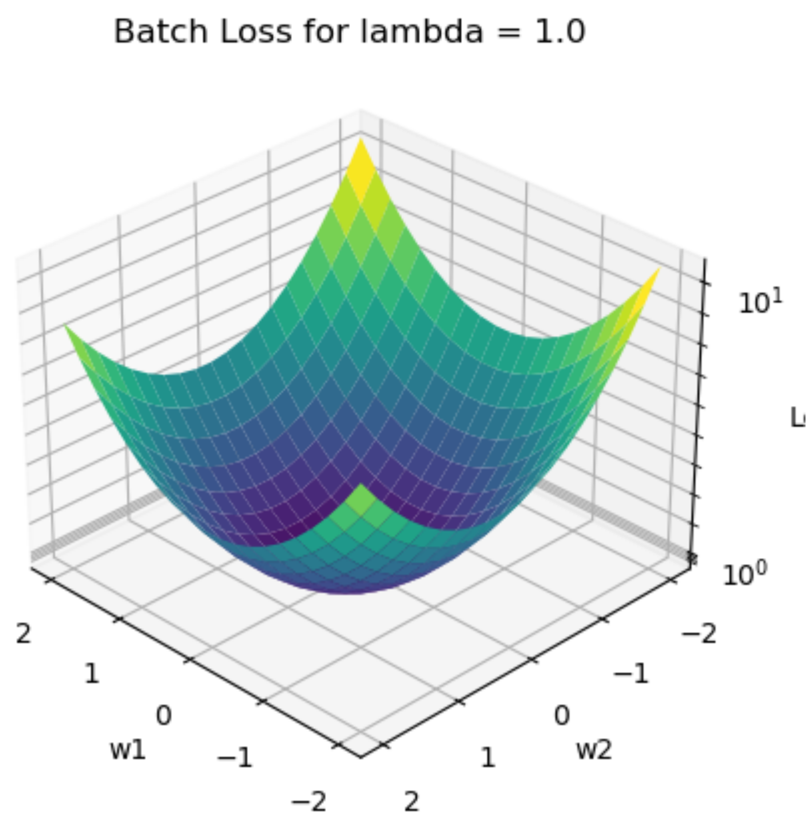
Problem 2.1

2.1.1

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

data = pd.read_csv('data.csv')

def sigmoid(w, x):
    return 1/(1+np.exp(-np.dot(w.flatten(),x.flatten())))

def sample_loss(w,x,y, l=0):
    y_hat = sigmoid(w,x)
    return -(y*np.log(y_hat) + (1-y)*np.log(1-y_hat)) + l*np.sum(w**2)

def batch_loss(w, X, y, l=0):
    return np.mean([sample_loss(w, x, y, l) for x,y in zip(X,y)])
```

In [ ]:
```
# plot loss over data
def plot_loss(x,y, w_lim, l=0):

    w1_values = np.linspace(-w_lim, w_lim, 20)
    w2_values = np.linspace(-w_lim, w_lim, 20)
    w1_mesh, w2_mesh = np.meshgrid(w1_values, w2_values)
    loss_values = np.zeros_like(w1_mesh)

    # Calculate loss for each combination of w1 and w2
    for i in range(len(w1_values)):
        for j in range(len(w2_values)):
            w = np.array([w1_values[i], w2_values[j]])
            loss_values[i, j] = batch_loss(w, x, y, l)

    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')

    # Plot the loss as a surface
    surface = ax.plot_surface(w1_mesh, w2_mesh, loss_values, cmap='viridi
    ax.view_init(elev=30, azim=135)
    ax.set_zscale('log')
    ax.set_xlabel('w1')
    ax.set_ylabel('w2')
    ax.set_zlabel('Loss')
    ax.set_title('Batch Loss for lambda = {}'.format(l))

x = np.stack([np.array(data['x0']), np.array(data['x1'])], axis=1)
y = np.array(data['y'])

plot_loss(x, y, w_lim=2, l=0)
plot_loss(x, y, w_lim=2, l=1e0)
```





2.1.2 Analytic Gradient

$\nabla_w L = (\hat{y}_i - y_i)\vec{x} - \lambda\vec{w}$

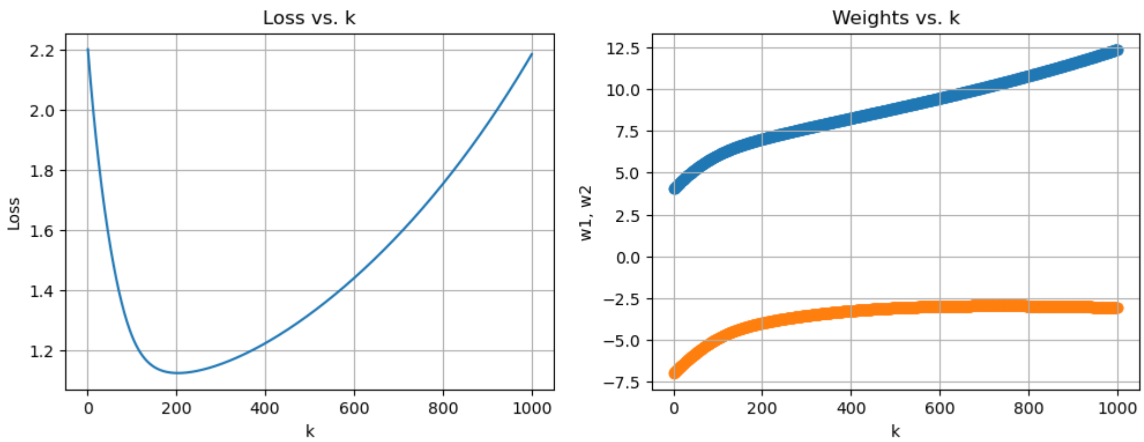
In [ ]:
```
def grad_w(w, x, y, l=0):
    y_hat = sigmoid(w,x)
    return (y_hat - y)*x - l*w

# test gradient
w_ = np.array([1,2])
x_ = np.array([1,2])
y_ = 0
print("grad: {}".format(grad_w(w_,x_,y_, l=0)))
```

```
grad: [0.99330715 1.9866143 ]
```

2.1.3 Batch Gradient Descent

In [ ]:
```
def batch_gradient_descent(w, x, y, alpha=0.1, l=0):
    loss = batch_loss(w, x, y, l)
    w = w - alpha * np.mean([grad_w(w, x, y, l) for x,y in zip(x,y)], axi
    return w, loss
```

In [ ]:
```
w_list = []
loss_list = []
w = np.array([4,-7])
k = np.arange(1, 1001)
for i in k:
    w, loss = batch_gradient_descent(w, x, y, alpha=0.1, l=1e-2)
    w_list.append(w)
    loss_list.append(loss)

# first subplot loss vs k, second subplot scatter plot of weights over k
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12,4))
ax1.plot(k,loss_list)
ax1.set_xlabel('k')
ax1.set_ylabel('Loss')
ax1.set_title('Loss vs. k')
ax1.grid()
ax2.scatter(k, [w[0] for w in w_list])
ax2.scatter(k, [ w[1] for w in w_list])
ax2.set_xlabel('k')
ax2.set_ylabel('w1, w2')
ax2.set_title('Weights vs. k')
ax2.grid()
plt.show()
```



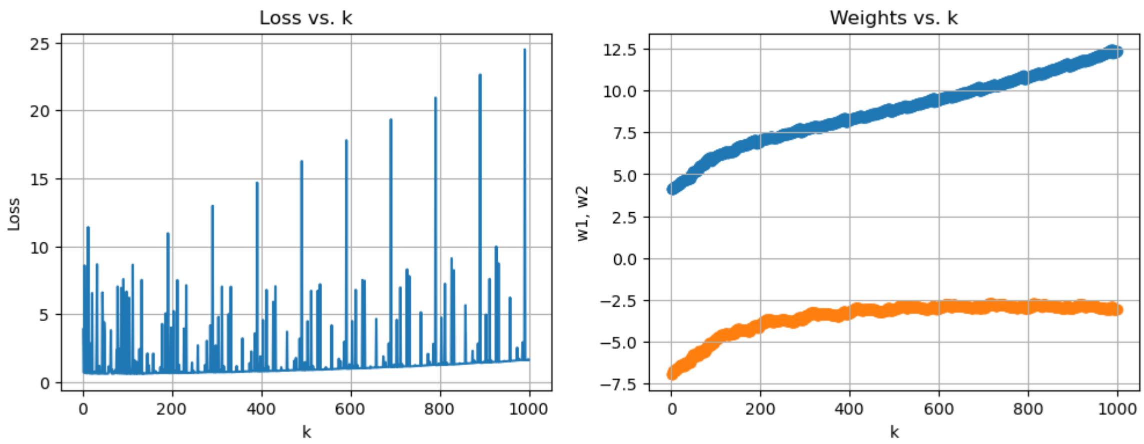
Optimizes but stop criteria is missing. Moves away from optimal w at some point.

2.1.4 Stochastic Gradient Descent

In [ ]:
```
def stochastic_gradient_descent(w, x, y, alpha=0.1, l=0):
    loss = sample_loss(w, x, y, l)
    w = w - alpha * grad_w(w, x, y, l)
    return w, loss
```

In [ ]:
```
# stochastic gradient descent
w_list = []
loss_list = []
w = np.array([4,-7])
k = np.arange(1, 1001)

data_len = len(x)
print("data length: {}".format(data_len))

for i in k:
    w, loss = stochastic_gradient_descent(w, x[i%len(x)], y[i%len(x)], al
    w_list.append(w)
    loss_list.append(loss)

# first subplot loss vs k, second subplot scatter plot of weights over k
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12,4))
ax1.plot(k,loss_list)
ax1.set_xlabel('k')
ax1.set_ylabel('Loss')
ax1.set_title('Loss vs. k')
ax1.grid()
ax2.scatter(k, [w[0] for w in w_list])
ax2.scatter(k, [ w[1] for w in w_list])
ax2.set_xlabel('k')
ax2.set_ylabel('w1, w2')
ax2.set_title('Weights vs. k')
ax2.grid()
plt.show()
```

```
data length: 100
```



SGD better for large data sets. Epochs > datpoints to traverse whole data at least once.
SGD is noisier than BGD. BGD has smoother convergence properties.

2.1.5 Tensorflow - Auto Diff / GradientTape API

In [ ]:

```
import tensorflow as tf
import numpy as np

def sigmoid_tf(w, x):
    return 1 / (1 + tf.exp(-tf.linalg.matmul(tf.transpose(w), x)))

def custom_loss(w, x, y, l=0):
    y_hat = sigmoid_tf(w, x)
    loss = -(y * tf.math.log(y_hat) + (1 - y) * tf.math.log(1 - y_hat)) +
    return tf.reduce_sum(loss)

def grad_w_tf(w, x, y, l=0):
    with tf.GradientTape() as tape:
        y_hat = sigmoid_tf(w, x)
        loss = custom_loss(w,x,y,l)
    gradients = tape.gradient(loss, w)
    return gradients

# Example data
x = np.random.randn(2, 1).astype(np.float32)  # Sample input as a 2x1 mat
y = np.random.randn(2)  # Sample label
w = tf.Variable(np.random.randn(2, 1).astype(np.float32), dtype=tf.float3

print("x.shape: {}".format(x.shape))
print("y.shape: {}".format(y.shape))
print("w.shape: {}".format(w.shape))
gradients = grad_w_tf(w, x, y, l=0)
print(gradients)
```

```
2023-11-22 00:16:51.956348: I tensorflow/core/util/port.cc:113] oneDNN cu
stom operations are on. You may see slightly different numerical results
due to floating-point round-off errors from different computation orders.
To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2023-11-22 00:16:51.959329: I external/local_tsl/tsl/cuda/cudart_stub.cc:
31] Could not find cuda drivers on your machine, GPU will not be used.
2023-11-22 00:16:52.002993: E external/local_xla/xla/stream_executor/cuda
/cuda_dnn.cc:9261] Unable to register cuDNN factory: Attempting to regist
er factory for plugin cuDNN when one has already been registered
2023-11-22 00:16:52.003029: E external/local_xla/xla/stream_executor/cuda
/cuda_fft.cc:607] Unable to register cuFFT factory: Attempting to registe
r factory for plugin cuFFT when one has already been registered
2023-11-22 00:16:52.004318: E external/local_xla/xla/stream_executor/cuda
/cuda_blas.cc:1515] Unable to register cuBLAS factory: Attempting to regi
ster factory for plugin cuBLAS when one has already been registered
2023-11-22 00:16:52.011953: I external/local_tsl/tsl/cuda/cudart_stub.cc:
31] Could not find cuda drivers on your machine, GPU will not be used.
2023-11-22 00:16:52.012605: I tensorflow/core/platform/cpu_feature_guard.
cc:182] This TensorFlow binary is optimized to use available CPU instruct
ions in performance-critical operations.
To enable the following instructions: AVX2 AVX_VNNI FMA, in other operati
ons, rebuild TensorFlow with the appropriate compiler flags.
2023-11-22 00:16:53.540139: W tensorflow/compiler/tf2tensorrt/utils/py_ut
ils.cc:38] TF-TRT Warning: Could not find TensorRT
x.shape: (2, 1)
y.shape: (2,)
w.shape: (2, 1)
tf.Tensor(
[[-2.31838   ]
 [-0.34333378]], shape=(2, 1), dtype=float32)
```

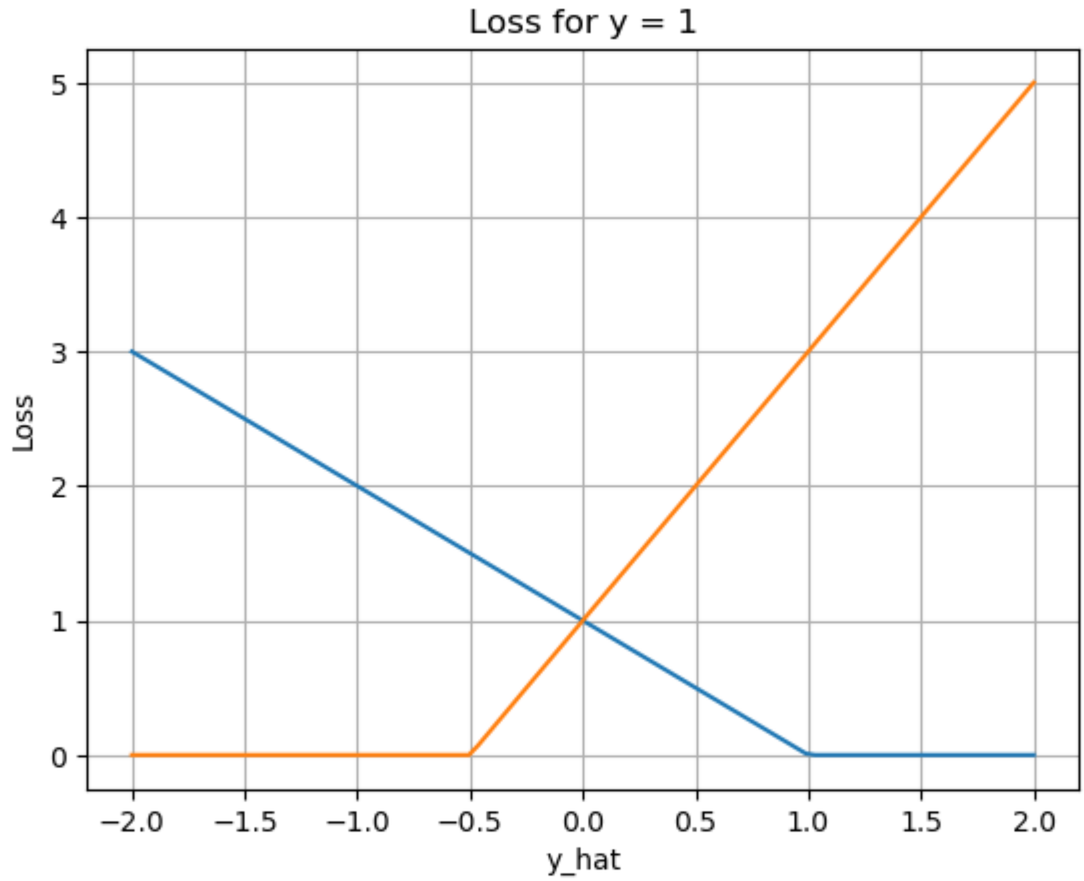
In [ ]:
```
# compare grad and grad_w
w_ = np.array([1,2]).reshape(2,1)
x_ = np.array([1,2]).reshape(2,1)
y_ = np.array([0])
my_grad = grad_w(w_,x_,y_, l=0)

print("grad: {}".format(my_grad))
w = tf.Variable(w_, dtype=tf.float32)  # Initializing weights as a 2x1 ma
tf_grad = grad_w_tf(w, x_, y_, l=0)
print("grad_w: {}".format(grad_w_tf(w,x_,y_, l=0)))
delta_grad = my_grad - tf_grad
print("delta_grad: {}".format(delta_grad))
```

```
grad: [[0.99330715]
 [1.9866143 ]]
grad_w: [[0.9933107]
 [1.9866214]]
delta_grad: [[-3.516674e-06]
 [-7.033348e-06]]
```

Both gradient computations match!

Problem 2.2

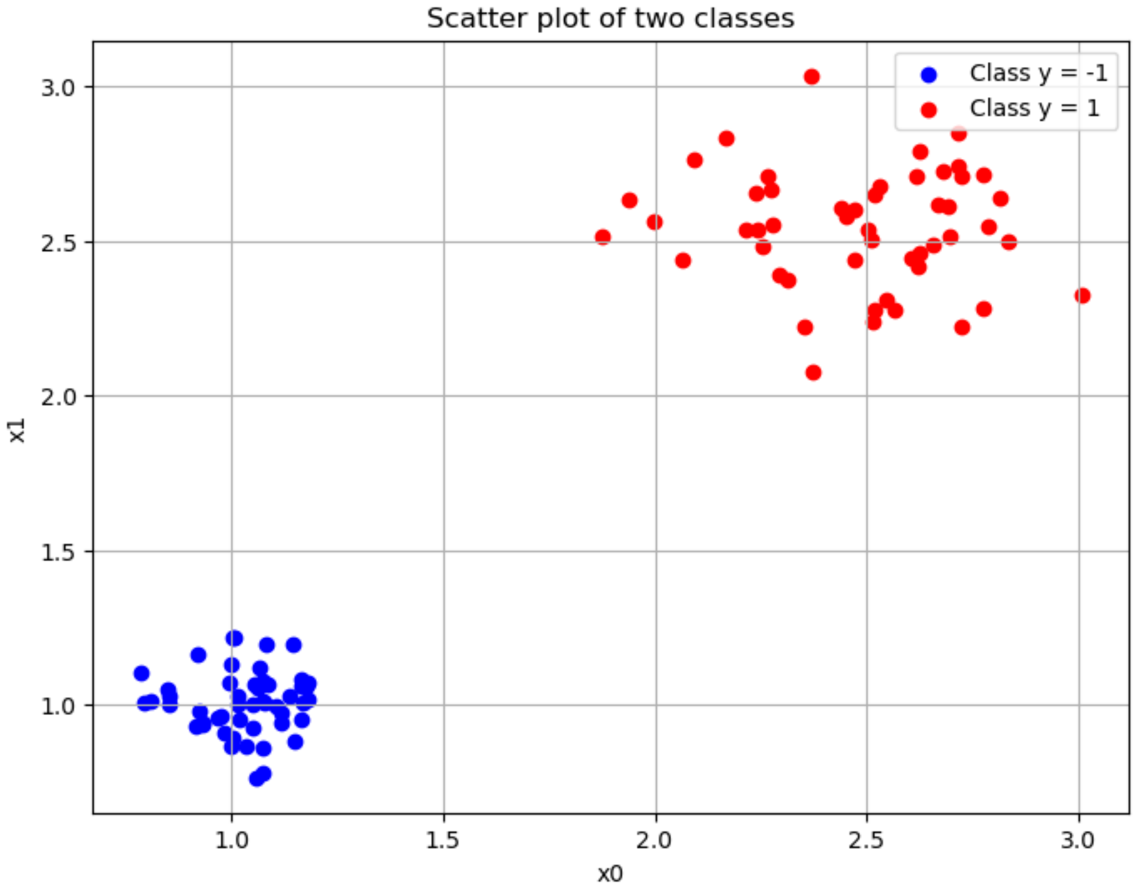
In [ ]:
```
def svc_loss(w, b, x, y, l=0):
    y_hat = x.transpose() @ w + b
    return np.mean(np.maximum(0, 1 - y * y_hat) + l * np.sum(w ** 2))

def svc_loss2(y_hat, y, l=0):
    w = 0
    return np.mean(np.maximum(0, 1 - y * y_hat) + l * np.sum(w ** 2))


# plot svc_loss2 with y = 1 over y_hat
y_hat = np.linspace(-2, 2, 100)
loss_values = np.zeros_like(y_hat)
loss_values2 = np.zeros_like(y_hat)
for i in range(len(y_hat)):
    loss_values[i] = svc_loss2(y_hat[i], y=1, l=0)
    loss_values2[i] = svc_loss2(y_hat[i], y=-2, l=0)

fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(y_hat, loss_values, label='y = 1')
ax.plot(y_hat, loss_values2, label='y = -1')
ax.set_xlabel('y_hat')
ax.set_ylabel('Loss')
ax.set_title('Loss for y = 1')
ax.grid()
plt.show()
```



Perceptron had smoother transition.

2.2.2 SVC with linear kernel and C = 1

In [ ]:
```
from sklearn.svm import SVC

data = pd.read_csv('blobs.csv')

x = np.stack([np.array(data['x_0']), np.array(data['x_1'])], axis=1)
y = np.array(data['y'])

# Scatter plot to visualize the classes
plt.figure(figsize=(8, 6))
plt.scatter(x[y == 0][:, 0], x[y == 0][:, 1], label='Class y = -1', c='bl
plt.scatter(x[y == 1][:, 0], x[y == 1][:, 1], label='Class y = 1', c='red
plt.xlabel('x0')
plt.ylabel('x1')
plt.title('Scatter plot of two classes')
plt.legend()
plt.grid()
plt.show()

# Fit SVC with linear kernel and C=1
clf = SVC(kernel='linear', C=1)
clf.fit(x, y)

# Extract weights and bias from the trained model
weights = clf.coef_[0]
bias = clf.intercept_[0]

print("Weights (w):", weights)
print("Bias (b):", bias)
```

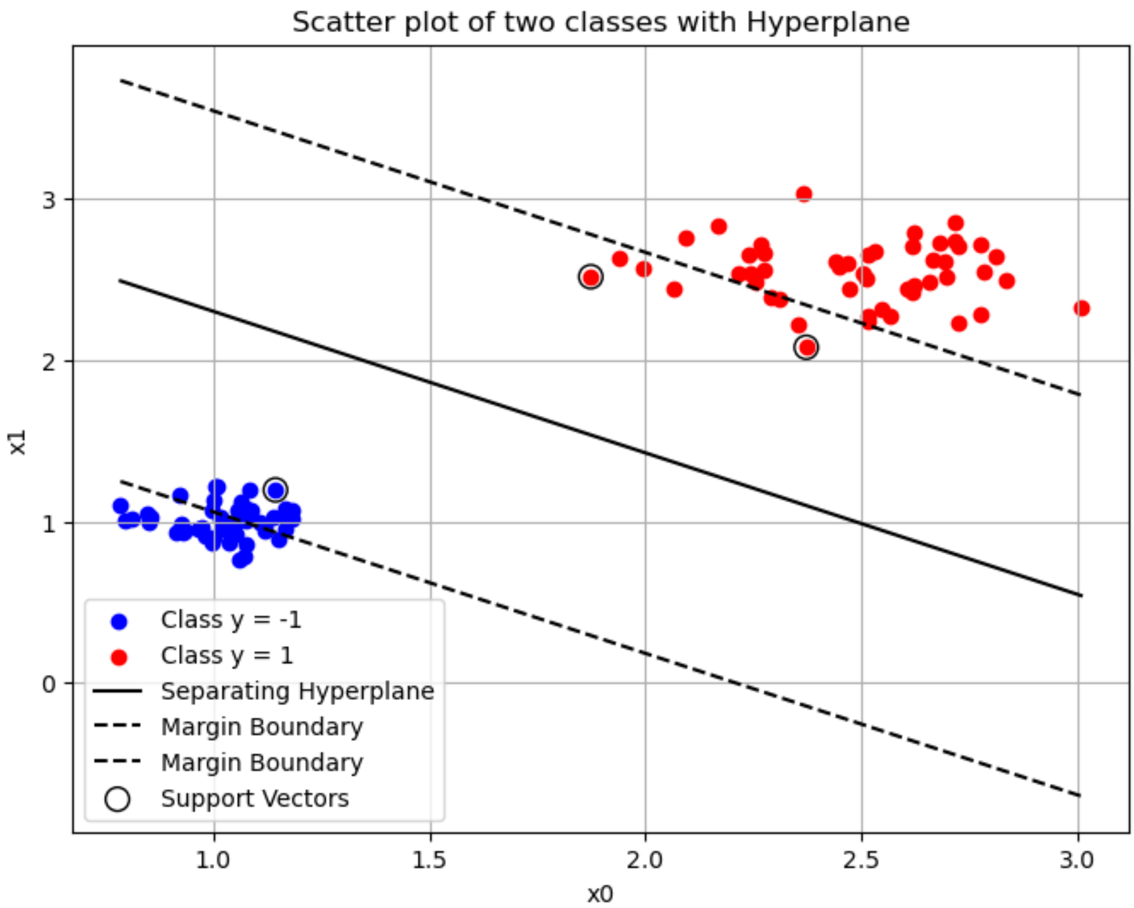



```
Weights (w): [0.89418064 1.01994712]
Bias (b): -3.241488064209292
```

2.2.3 Add Hyperplane

In [ ]:
```
# Scatter plot to visualize the classes
plt.figure(figsize=(8, 6))
plt.scatter(x[y == 0][:, 0], x[y == 0][:, 1], label='Class y = -1', c='bl
plt.scatter(x[y == 1][:, 0], x[y == 1][:, 1], label='Class y = 1', c='red
plt.xlabel('x0')
plt.ylabel('x1')
plt.title('Scatter plot of two classes with Hyperplane')


# Plotting the separating hyperplane
xx = np.linspace(np.min(x[:, 0]), np.max(x[:, 0]), 100)
yy = (-bias - weights[0] * xx) / weights[1]

plt.plot(xx, yy, 'k-', label='Separating Hyperplane')

# Calculate margin distance
support_vectors = clf.support_vectors_
distances = clf.decision_function(support_vectors)
margin_distance = 2 / np.linalg.norm(weights)

print("Margin distance:", margin_distance)

# Plotting margin boundaries
yy_down = yy - np.sqrt(1 + np.dot(weights, weights)) / np.linalg.norm(wei
yy_up = yy + np.sqrt(1 + np.dot(weights, weights)) / np.linalg.norm(weigh

plt.plot(xx, yy_down, 'k--', label='Margin Boundary')
plt.plot(xx, yy_up, 'k--', label='Margin Boundary')

# Plot support vectors
plt.scatter(support_vectors[:, 0], support_vectors[:, 1], s=100, facecolo
plt.legend()
plt.grid()
plt.show()
```

```
Margin distance: 1.4744792005954694
```

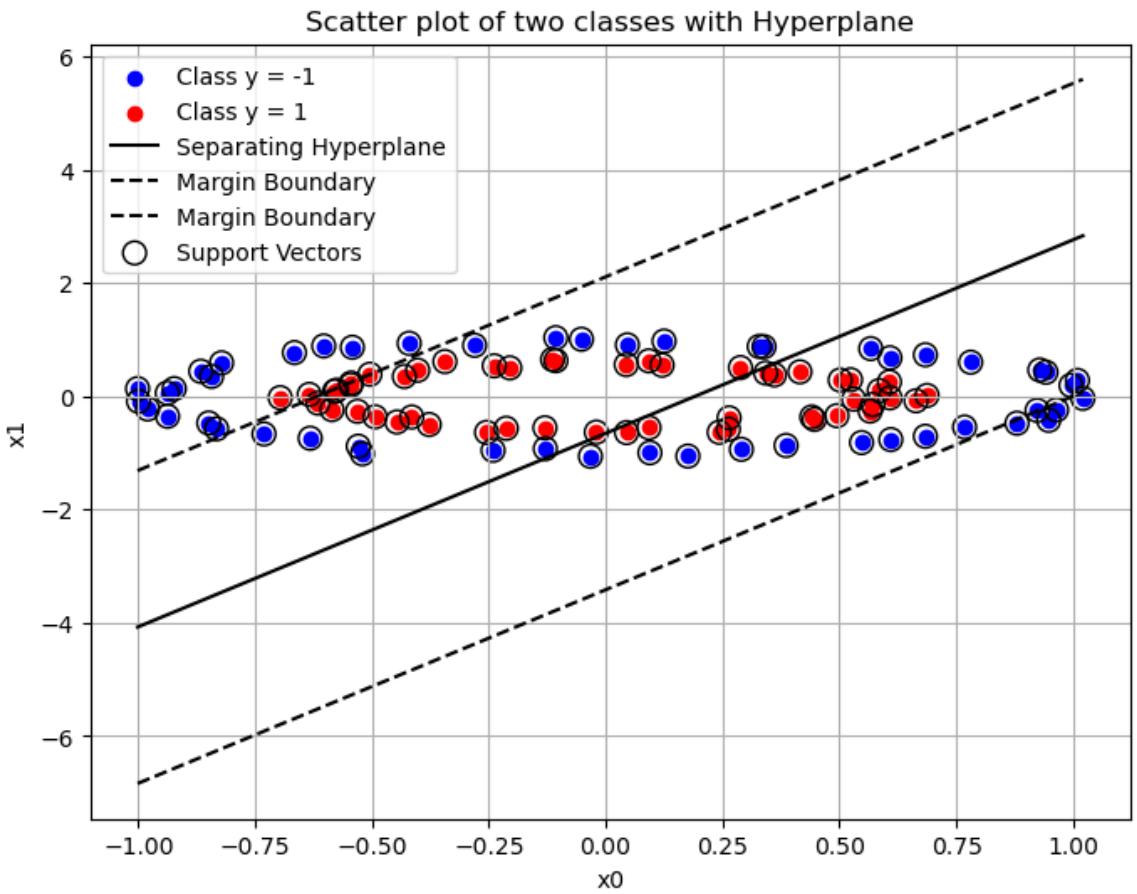


TODO: derive margin distance

2.2.4 Same but with Circles

In [ ]:
```
data = pd.read_csv('circles.csv')

x = np.stack([np.array(data['x_0']), np.array(data['x_1'])], axis=1)
y = np.array(data['y'])

# Scatter plot to visualize the classes
plt.figure(figsize=(8, 6))
plt.scatter(x[y == 0][:, 0], x[y == 0][:, 1], label='Class y = -1', c='bl
plt.scatter(x[y == 1][:, 0], x[y == 1][:, 1], label='Class y = 1', c='red
plt.xlabel('x0')
plt.ylabel('x1')
plt.title('Scatter plot of two classes with Hyperplane')

# Fit SVC with linear kernel and C=1
clf = SVC(kernel='linear', C=1)
clf.fit(x, y)

# Extract weights and bias from the trained model
weights = clf.coef_[0]
bias = clf.intercept_[0]

# Plotting the separating hyperplane
xx = np.linspace(np.min(x[:, 0]), np.max(x[:, 0]), 100)
yy = (-bias - weights[0] * xx) / weights[1]

plt.plot(xx, yy, 'k-', label='Separating Hyperplane')

# Calculate margin distance
support_vectors = clf.support_vectors_
distances = clf.decision_function(support_vectors)
margin_distance = 2 / np.linalg.norm(weights)

print("Margin distance:", margin_distance)

# Plotting margin boundaries
yy_down = yy - np.sqrt(1 + np.dot(weights, weights)) / np.linalg.norm(wei
yy_up = yy + np.sqrt(1 + np.dot(weights, weights)) / np.linalg.norm(weigh

plt.plot(xx, yy_down, 'k--', label='Margin Boundary')
plt.plot(xx, yy_up, 'k--', label='Margin Boundary')

# Plot support vectors
plt.scatter(support_vectors[:, 0], support_vectors[:, 1], s=100, facecolo
plt.legend()
plt.grid()
plt.show()
```

```
Margin distance: 5.1504212723688685
```



Dataset is not separable with a linear classifier.

2.2.5 Transformation

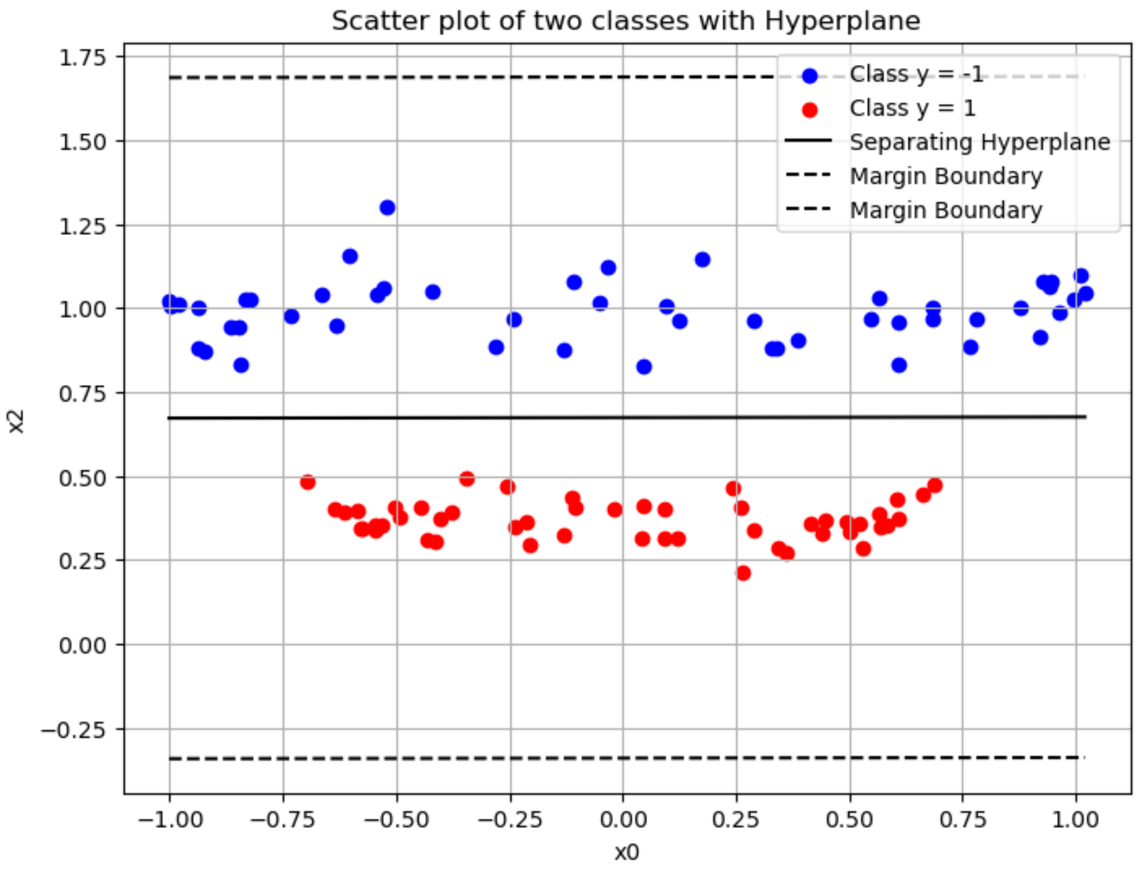
In [ ]:

```
x = np.stack([np.array(data['x_0']), np.array(data['x_1']), np.array(data
y = np.array(data['y'])

print("x.shape: {}".format(x.shape))
print("y.shape: {}".format(y.shape))
```

```
x.shape: (100, 3)
y.shape: (100,)
```

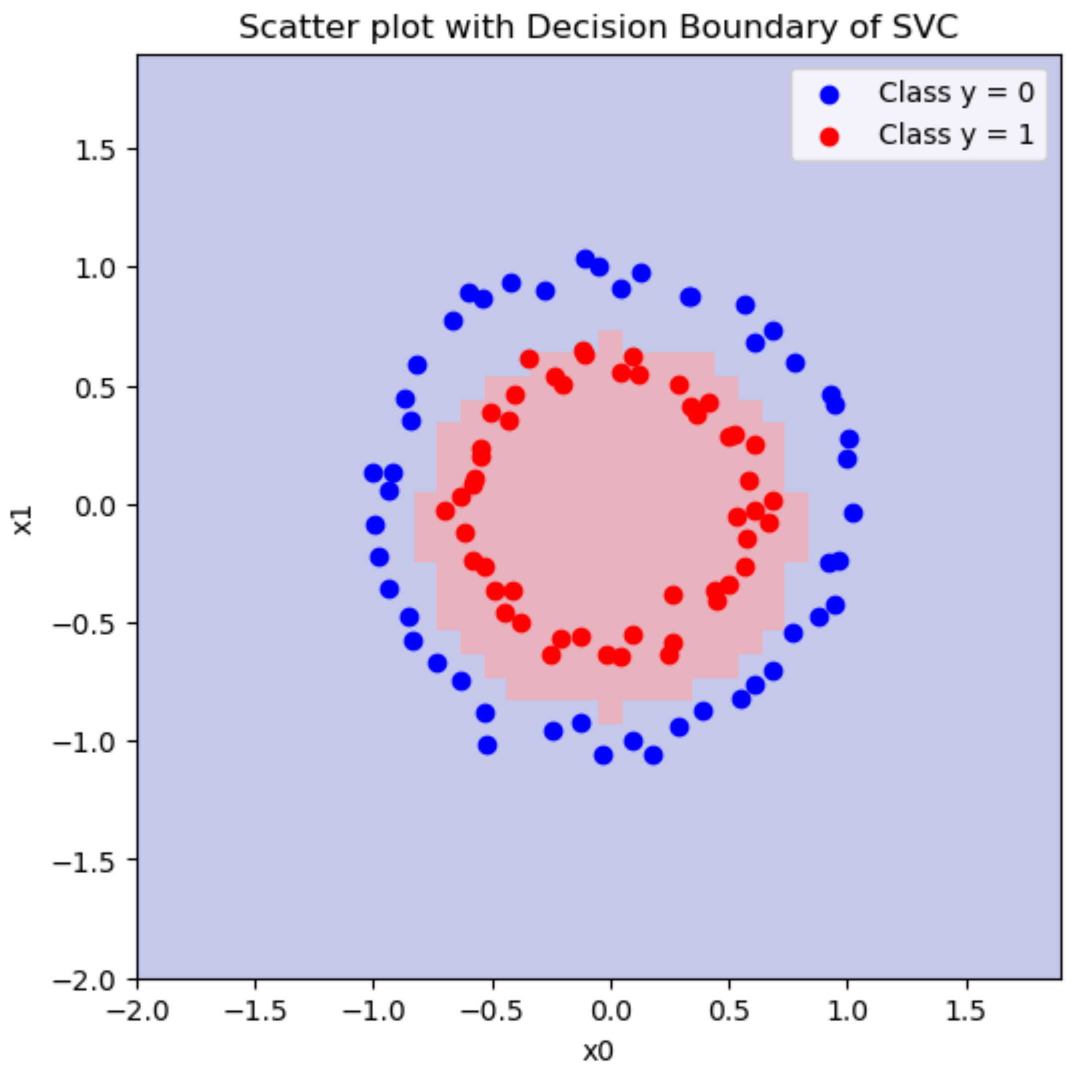
In [ ]:
```
# Scatter plot to visualize the classes
plt.figure(figsize=(8, 6))
plt.scatter(x[y == 0][:, 0], x[y == 0][:, 2], label='Class y = -1', c='bl
plt.scatter(x[y == 1][:, 0], x[y == 1][:, 2], label='Class y = 1', c='red
plt.xlabel('x0')
plt.ylabel('x2')
plt.title('Scatter plot of two classes with Hyperplane')

clf = SVC(kernel='linear', C=100)
clf.fit(x, y)

# Extract weights and bias from the trained model
weights = clf.coef_[0]
bias = clf.intercept_[0]

# Plotting the separating hyperplane
xx = np.linspace(np.min(x[:, 0]), np.max(x[:, 0]), 100)
yy = (-bias - weights[0] * xx) / weights[2]

plt.plot(xx, yy, 'k-', label='Separating Hyperplane')

# Calculate margin distance
support_vectors = clf.support_vectors_
distances = clf.decision_function(support_vectors)
margin_distance = 2 / np.linalg.norm(weights)

print("Margin distance:", margin_distance)

# Plotting margin boundaries
yy_down = yy - np.sqrt(1 + np.dot(weights, weights)) / np.linalg.norm(wei
yy_up = yy + np.sqrt(1 + np.dot(weights, weights)) / np.linalg.norm(weigh

plt.plot(xx, yy_down, 'k--', label='Margin Boundary')
plt.plot(xx, yy_up, 'k--', label='Margin Boundary')

plt.legend()
plt.grid()
plt.show()
```

```
Margin distance: 0.3360823132713146
```



2.2.6 Polynomial Kernel

In [ ]:
```
data = pd.read_csv('circles.csv')

x = np.stack([np.array(data['x_0']), np.array(data['x_1'])], axis=1)
y = np.array(data['y'])

# Fit SVC with linear kernel and C=1
clf = SVC(C=1, kernel="poly", degree=2)
clf.fit(x, y)

# Create a meshgrid of points spanning the range of x_0 and x_1
x_min = -2
x_max = 2
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
                     np.arange(x_min, x_max, 0.1))

# Predict the class labels for each point in the meshgrid
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Plotting the heatmap
plt.figure(figsize=(8, 6))
plt.imshow(Z, interpolation='nearest', extent=(xx.min(), xx.max(), yy.min

# Scatter plot to visualize the classes
plt.scatter(x[y == 0][:, 0], x[y == 0][:, 1], label='Class y = 0', c='blu
plt.scatter(x[y == 1][:, 0], x[y == 1][:, 1], label='Class y = 1', c='red
plt.xlabel('x0')
plt.ylabel('x1')
plt.title('Scatter plot with Decision Boundary of SVC')

plt.legend()
plt.show()
```



2.2.7 Transformation of quadratic polynomial kernel

$\phi(\vec{x}) = [x^2, 2x, 1]^T$ and $\dim(\phi(\vec{x})) = 2n + 1$ with $x \in \mathrm{R}^n$.

Problem 2.3

2.3.1 Prove Matrix Identity

$(\bm{X}^T\bm{X} + \lambda\bm{I}_{d\times d})^{-1}\bm{X}^T = \bm{X}^T(\bm{X}\bm{X}^T + \lambda\bm{I}_{n\times n})^{-1}$

$\bm{X}^T(\bm{X}\bm{X}^T + \lambda\bm{I}_{n\times n}) = (\bm{X}^T\bm{X} + \lambda\bm{I}_{d\times d})\bm{X}^T$

$\bm{X}^T\bm{X}\bm{X}^T + \lambda\bm{X}^T\bm{I}_{n\times n} = \bm{X}^T\bm{X}\bm{X}^T + \lambda\bm{I}_{d\times d}\backslash$

$\bm{X} \in \mathrm{R}^{n\times d}$, hence

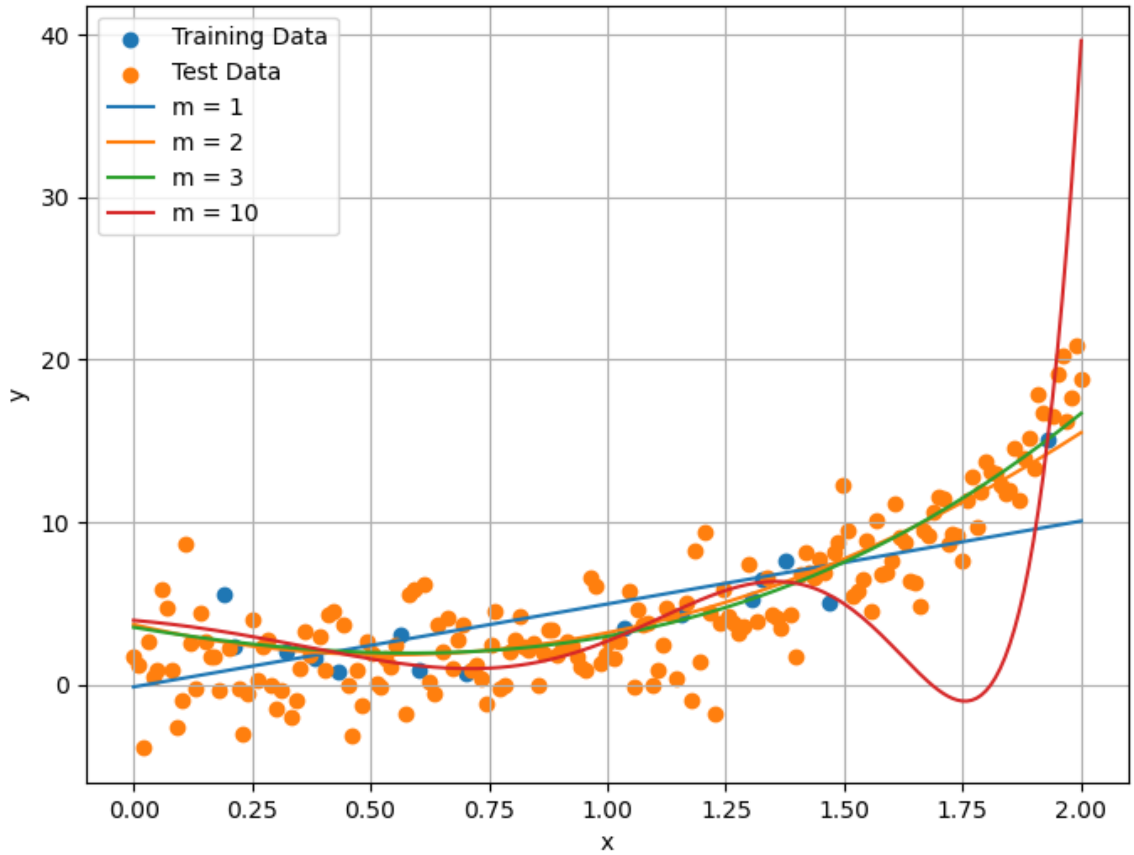
$\bm{X}^T\bm{X}\bm{X}^T + \lambda\bm{X}^T = \bm{X}^T\bm{X}\bm{X}^T + \lambda\bm{X}^T$

2.3.2 Custom Estimator with polynomial Kernel

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt

# Extract the data from the CSV files using pandas
train_data = pd.read_csv('../01_uebung/regression_train.csv')
x_train = train_data['x'].values.reshape(-1, 1)
y_train = train_data['y'].values.reshape(-1, 1)

test_data = pd.read_csv('../01_uebung/regression_test.csv')
x_test = test_data['x'].values.reshape(-1, 1)
y_test = test_data['y'].values.reshape(-1, 1)
```

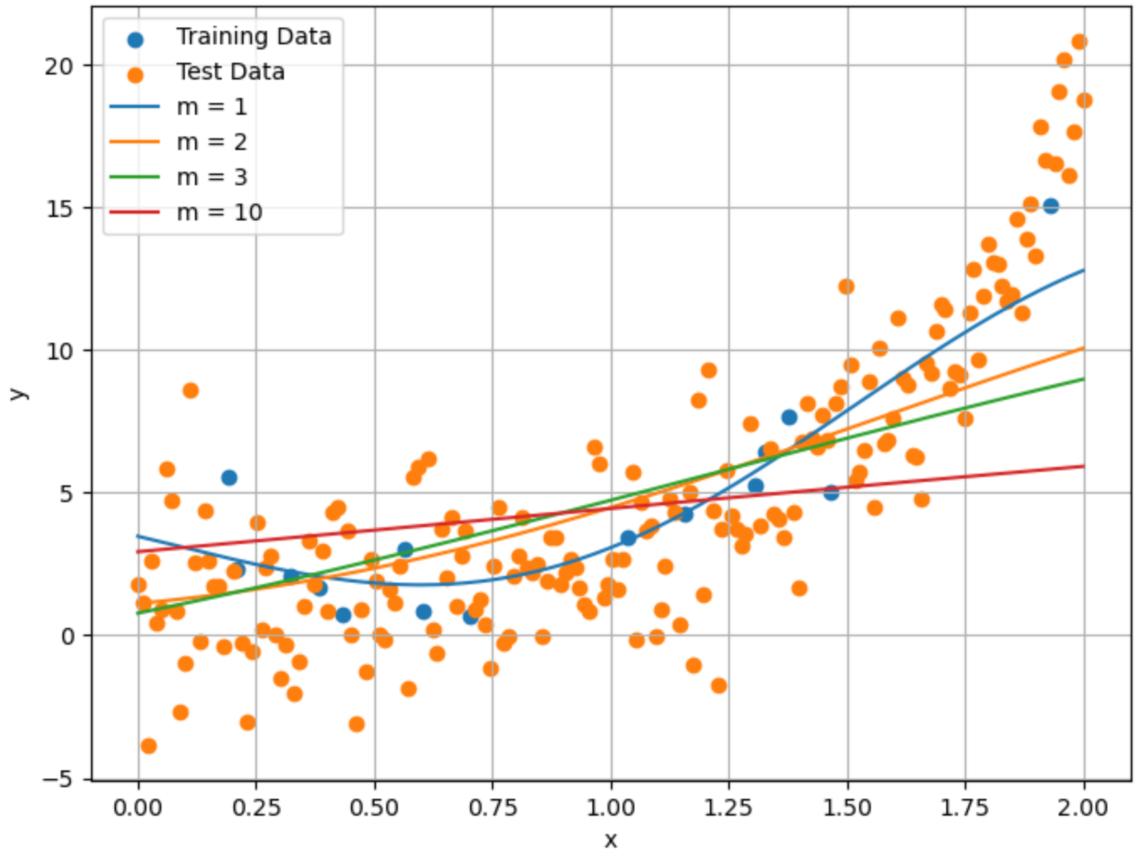
2.3.3 Custom Ridge Kernel

In [ ]:
```
import numpy as np

class CustomKernelRidge:
    def __init__(self, lambda_=1.0, m=1):
        self.lambda_ = lambda_
        self.m = m

    def fit(self, X, y):
        self.X_train = X
        n_samples = X.shape[0]

        self.kernel = lambda X, Y: (np.dot(X, Y.T) + 1) ** self.m

        K = self.kernel(X, X)

        # Add a small value to the diagonal for regularization
        K[np.diag_indices_from(K)] += self.lambda_

        # Compute alpha weights using the kernel matrix
        self.alpha_weights = np.linalg.solve(K, y)

    def predict(self, X):
        # Compute kernel matrix between test and train data
        K_x = self.kernel(X, self.X_train)

        # Make predictions
        y_pred = np.dot(K_x, self.alpha_weights)
        return y_pred


m_ls = [1,2,3,10]
pred_ls = []
for m in m_ls:

    model = CustomKernelRidge(lambda_=0.1, m=m)
    model.fit(x_train, y_train)

    # Assuming X_test is your test data
    predictions = model.predict(x_test)
    pred_ls.append(predictions)

# Plot the predictions
plt.figure(figsize=(8, 6))
plt.scatter(x_train, y_train, label='Training Data')
plt.scatter(x_test, y_test, label='Test Data')
plt.plot(x_test, pred_ls[0], label='m = 1')
plt.plot(x_test, pred_ls[1], label='m = 2')
plt.plot(x_test, pred_ls[2], label='m = 3')
plt.plot(x_test, pred_ls[3], label='m = 10')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.grid()
plt.show()
```



2.3.4 RBF Kernel

In [ ]:
```
import numpy as np

class RBFKernelRidge:
    def __init__(self, lambda_=1.0, l_s=1):
        self.lambda_ = lambda_
        self.l_s = l_s

    def fit(self, X, y):
        self.X_train = X
        n_samples = X.shape[0]

        self.kernel = lambda X, Y: (np.exp(-np.linalg.norm(X[:, None] - Y

        K = self.kernel(X, X)

        # Add a small value to the diagonal for regularization
        K[np.diag_indices_from(K)] += self.lambda_

        # Compute alpha weights using the kernel matrix
        self.alpha_weights = np.linalg.solve(K, y)

    def predict(self, X):
        # Compute kernel matrix between test and train data
        K_x = self.kernel(X, self.X_train)

        # Make predictions
        y_pred = np.dot(K_x, self.alpha_weights)
        return y_pred


l_s_ls = [1,2,3,10]
pred_ls = []
for l_s in l_s_ls:

    model = RBFKernelRidge(lambda_=0.1, l_s=l_s)
    model.fit(x_train, y_train)

    # Assuming X_test is your test data
    predictions = model.predict(x_test)
    pred_ls.append(predictions)

# Plot the predictions
plt.figure(figsize=(8, 6))
plt.scatter(x_train, y_train, label='Training Data')
plt.scatter(x_test, y_test, label='Test Data')
plt.plot(x_test, pred_ls[0], label='m = 1')
plt.plot(x_test, pred_ls[1], label='m = 2')
plt.plot(x_test, pred_ls[2], label='m = 3')
plt.plot(x_test, pred_ls[3], label='m = 10')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.grid()
plt.show()
```



higher values of l_s linearizes the estimation.

2.3.5 RBF mapping function

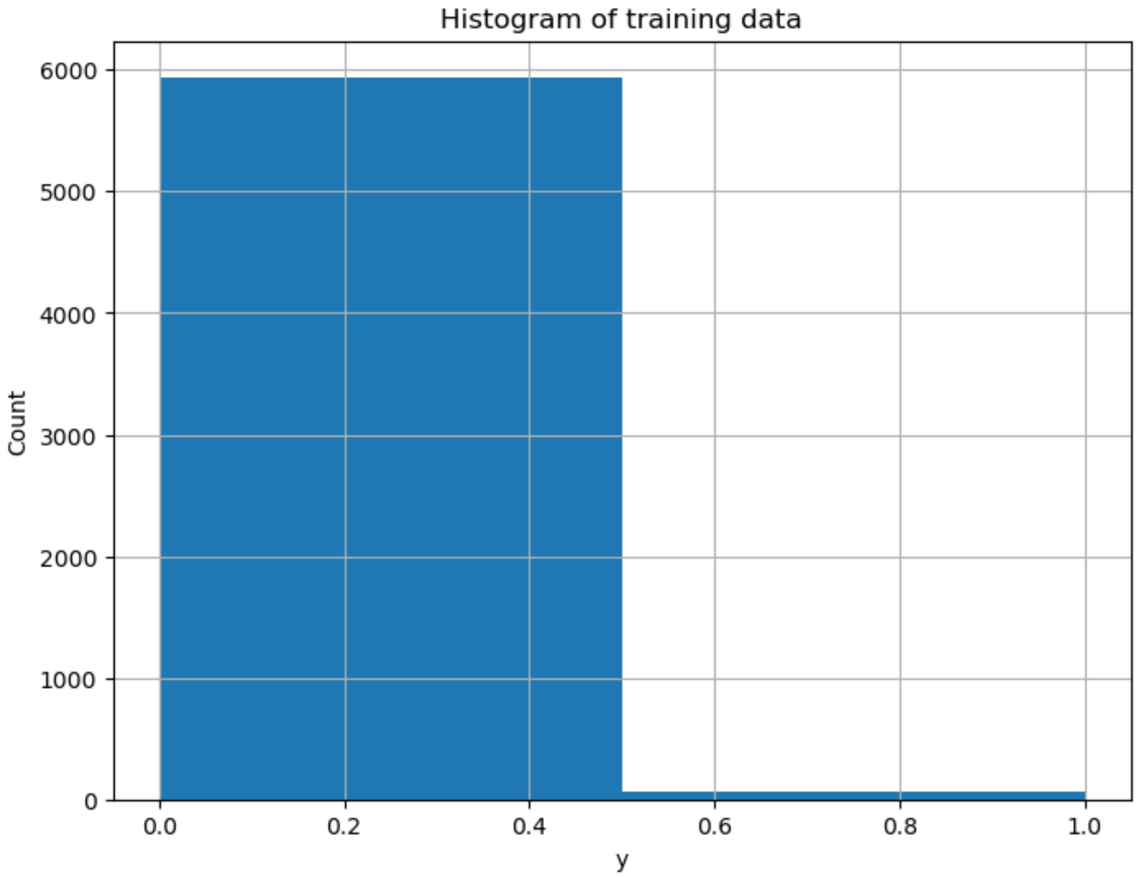
$$
\begin{aligned}
K_{\mathrm{RBF}}(\mathbf{x}, \mathbf{x}') &= \exp\left[-\frac{1}{2}\|\mathbf{x}-\mathbf{x}'\|^2\right] \\
&= \exp\left[-\frac{1}{2}\langle \mathbf{x}-\mathbf{x}', \mathbf{x}-\mathbf{x}'\rangle\right] \\
&= \exp\left[-\frac{1}{2}(\langle \mathbf{x}, \mathbf{x}-\mathbf{x}'\rangle - \langle \mathbf{x}', \mathbf{x}-\mathbf{x}'\rangle)\right] \\
&= \exp\left[-\frac{1}{2}(\langle \mathbf{x}, \mathbf{x}-\mathbf{x}'\rangle - \langle \mathbf{x}', \mathbf{x}-\mathbf{x}'\rangle)\right] \\
&= \exp\left[-\frac{1}{2}(\langle \mathbf{x}, \mathbf{x}\rangle - \langle \mathbf{x}, \mathbf{x}'\rangle - \langle \mathbf{x}', \mathbf{x}\rangle + \langle \mathbf{x}', \mathbf{x}'\rangle)\right] \\
&= \exp\left[-\frac{1}{2}(\|\mathbf{x}\|^2 + \|\mathbf{x}'\|^2 - 2\langle \mathbf{x}, \mathbf{x}'\rangle)\right] \\
&= \exp\left[-\frac{1}{2}\|\mathbf{x}\|^2 - \frac{1}{2}\|\mathbf{x}'\|^2\right]\exp\left[-\frac{1}{2} - 2\langle \mathbf{x}, \mathbf{x}'\rangle\right] \\
&= Ce^{\langle \mathbf{x}, \mathbf{x}'\rangle} && C := \exp\left[-\frac{1}{2}\|\mathbf{x}\|^2 - \frac{1}{2}\|\mathbf{x}'\|^2\right] \text{ is a constant} \\
&= C\sum_{n=0}^{\infty}\frac{\langle \mathbf{x}, \mathbf{x}'\rangle^n}{n!} && \text{Taylor expansion of } e^x \\
&= C\sum_{n=0}^{\infty}\frac{K_{\mathrm{poly(n)}}(\mathbf{x}, \mathbf{x}')}{n!}
\end{aligned}
$$

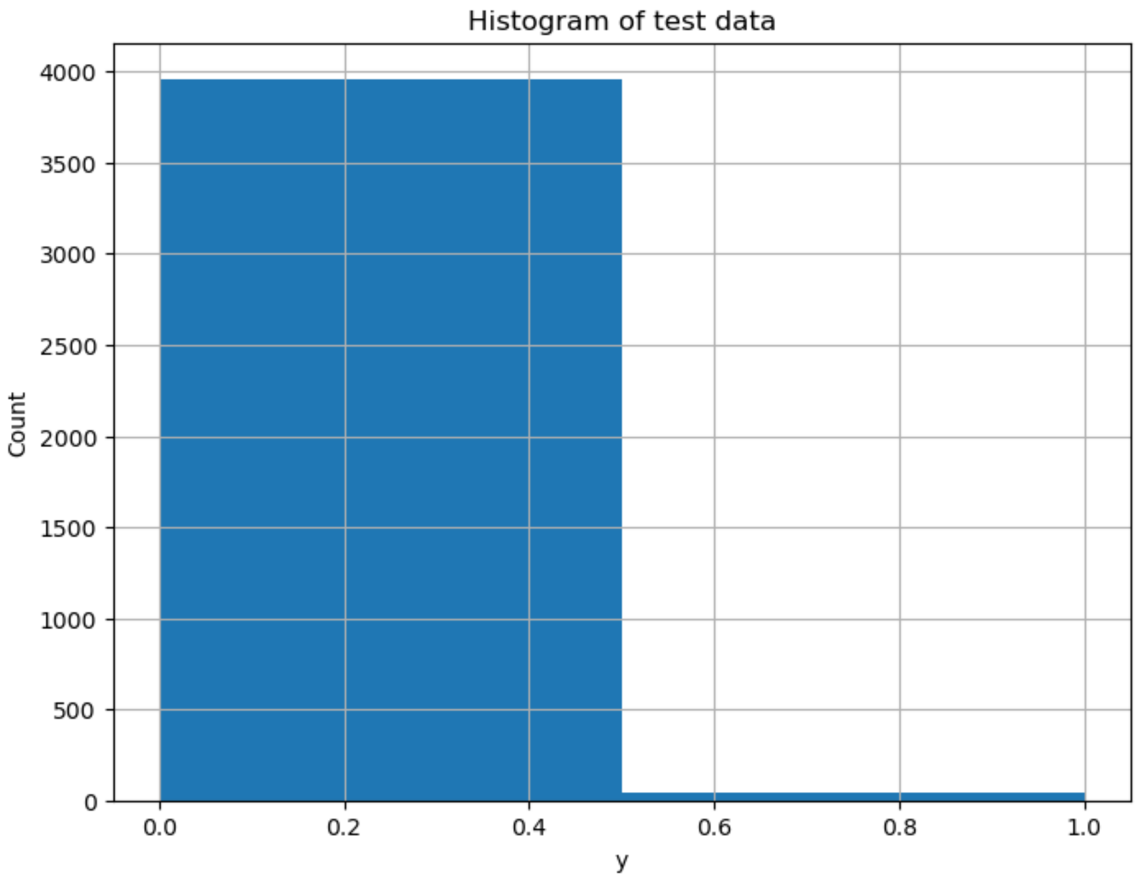
RBF kernel is formed by taking an infinite sum of polynomial kernels of all degrees.

Problem 2.4

2.4.1.

In [ ]:
```
train = pd.read_csv('imbalanced_train.csv')
test = pd.read_csv('imbalanced_test.csv')

x_train = np.stack([np.array(train['x_0']), np.array(train['x_1'])], axis
y_train = np.array(train['y'])

x_test = np.stack([np.array(test['x_0']), np.array(test['x_1'])], axis=1)
y_test = np.array(test['y'])


# histogram of training data
plt.figure(figsize=(8, 6))
plt.hist(y_train, bins=2)
plt.xlabel('y')
plt.ylabel('Count')
plt.title('Histogram of training data')
plt.grid()
plt.show()

# histogram of test data
plt.figure(figsize=(8, 6))
plt.hist(y_test, bins=2)
plt.xlabel('y')
plt.ylabel('Count')
plt.title('Histogram of test data')
plt.grid()
plt.show()
```





2.4.2 SVC with RBF Kernel

In [ ]:
```
# fit svc with rbf kernel and C=0.01 to training data
clf = SVC(C=0.01, kernel="rbf")
clf.fit(x_train, y_train)

# get accuracy for test data
accuracy = clf.score(x_test, y_test)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.99
```

2.4.3 Confusion Matrix

In [ ]:
```
import sklearn.metrics

y_pred = clf.predict(x_test)
confusion_matrix = sklearn.metrics.confusion_matrix(y_test, y_pred)
print("Confusion matrix:\n", confusion_matrix)

precision = sklearn.metrics.precision_score(y_test, y_pred)
recall = sklearn.metrics.recall_score(y_test, y_pred)
f1_score = sklearn.metrics.f1_score(y_test, y_pred)
print("Precision:", precision)
print("Recall:", recall)
print("F1 score:", f1_score)
```

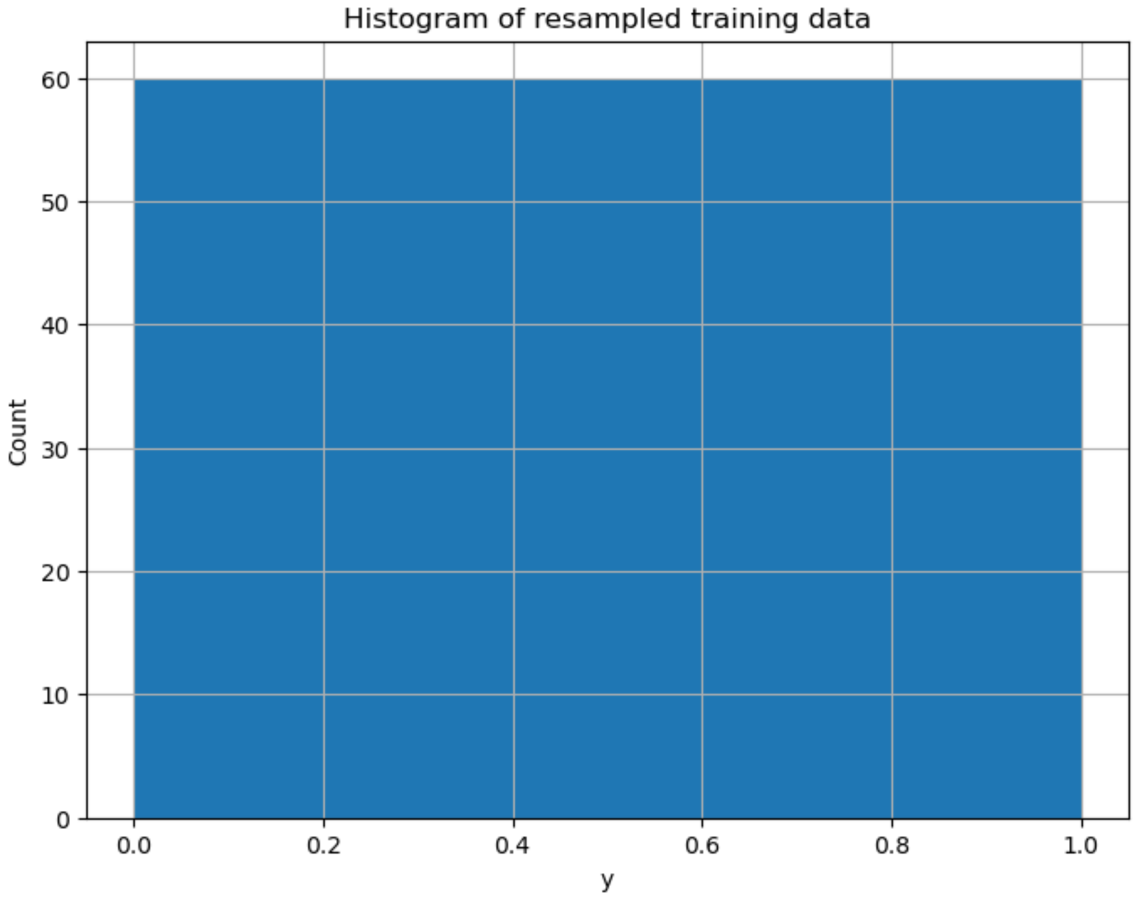
```
Confusion matrix:
 [[3960    0]
 [  40    0]]
Precision: 0.0
Recall: 0.0
F1 score: 0.0
```

```
/home/gebmer/anaconda3/lib/python3.11/site-packages/sklearn/metrics/_clas
sification.py:1344: UndefinedMetricWarning: Precision is ill-defined and
being set to 0.0 due to no predicted samples. Use `zero_division` paramet
er to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

2.4.4 Undersampling to balance data

In [ ]:

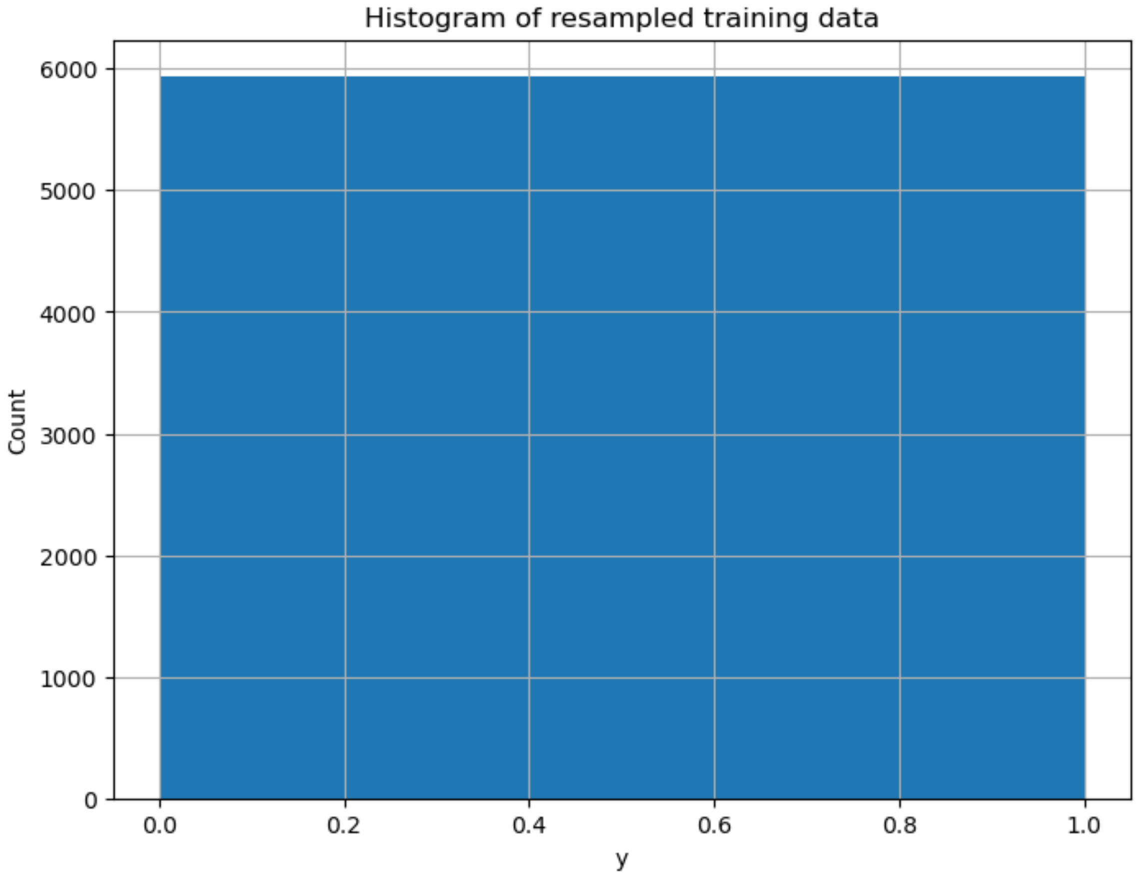
```
# undersample majority class in training data
from imblearn.under_sampling import RandomUnderSampler

rus = RandomUnderSampler(random_state=0)
x_train_resampled, y_train_resampled = rus.fit_resample(x_train, y_train)

# histogram of resampled training data
plt.figure(figsize=(8, 6))
plt.hist(y_train_resampled, bins=2)
plt.xlabel('y')
plt.ylabel('Count')
plt.title('Histogram of resampled training data')
plt.grid()
plt.show()

# fit svc with rbf kernel and C=0.01 to resampled training data
clf = SVC(C=0.01, kernel="rbf")
clf.fit(x_train_resampled, y_train_resampled)

# get accuracy for test data
accuracy = clf.score(x_test, y_test)
print("Accuracy:", accuracy)

y_pred = clf.predict(x_test)
confusion_matrix = sklearn.metrics.confusion_matrix(y_test, y_pred)
print("Confusion matrix:\n", confusion_matrix)

precision = sklearn.metrics.precision_score(y_test, y_pred)
recall = sklearn.metrics.recall_score(y_test, y_pred)
f1_score = sklearn.metrics.f1_score(y_test, y_pred)
print("Precision:", precision)
print("Recall:", recall)
print("F1 score:", f1_score)
```




```
Accuracy: 0.90775
Confusion matrix:
 [[3604  356]
 [  13   27]]
Precision: 0.07049608355091384
Recall: 0.675
F1 score: 0.12765957446808512
```

Oversampling to balance data

In [ ]:
```
# oversample minority class in training data
from imblearn.over_sampling import RandomOverSampler

ros = RandomOverSampler(random_state=0)
x_train_resampled, y_train_resampled = ros.fit_resample(x_train, y_train)

# histogram of resampled training data
plt.figure(figsize=(8, 6))
plt.hist(y_train_resampled, bins=2)
plt.xlabel('y')
plt.ylabel('Count')
plt.title('Histogram of resampled training data')
plt.grid()
plt.show()

# fit svc with rbf kernel and C=0.01 to resampled training data
clf = SVC(C=0.01, kernel="rbf")
clf.fit(x_train_resampled, y_train_resampled)

# get accuracy for test data
accuracy = clf.score(x_test, y_test)
print("Accuracy:", accuracy)

y_pred = clf.predict(x_test)
confusion_matrix = sklearn.metrics.confusion_matrix(y_test, y_pred)
print("Confusion matrix:\n", confusion_matrix)

precision = sklearn.metrics.precision_score(y_test, y_pred)
recall = sklearn.metrics.recall_score(y_test, y_pred)
f1_score = sklearn.metrics.f1_score(y_test, y_pred)

print("Precision:", precision)
print("Recall:", recall)
print("F1 score:", f1_score)
```




```
Accuracy: 0.947
Confusion matrix:
 [[3757  203]
 [   9   31]]
Precision: 0.13247863247863248
Recall: 0.775
F1 score: 0.22627737226277375
```

Oversampling is better.