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

numTP = lambda y,yhat: sum([1 for i in range(len(y)) if y[i] == True
and yhat[i] == True])
numFN = lambda y,yhat: sum([1 for i in range(len(y)) if y[i] == True
and yhat[i] == False])
numFP = lambda y,yhat: sum([1 for i in range(len(y)) if y[i] == False
and yhat[i] == True])
numTN = lambda y,yhat: sum([1 for i in range(len(y)) if y[i] == False
and yhat[i] == False])
confusion_matrix = lambda y,yhat:
np.vstack([[numTP(y,yhat),numFN(y,yhat)],
[numFP(y,yhat),numTN(y,yhat)]])
error_rate = lambda y,yhat: (numFN(y,yhat) + numFP(y,yhat)) / len(y)
error_rate2 = lambda y,yhat: np.average(y != yhat)
```

Least Squares for Classification

Source: CS189, Spring 2018, UC Berkeley

Classification is an important problem in applied machine learning and is used in many different applications like image classification, object detection, speech recognition, machine translation and others. In classification, we assign each datapoint a class from a finite set (for example the image of a digit could be assigned the value 0,1,...,9 of that digit). This is different from regression, where each datapoint is assigned a value from a continuous domain like R (for example features of a house like location, number of bedrooms, age of the house, etc. could be assigned the price of the house).

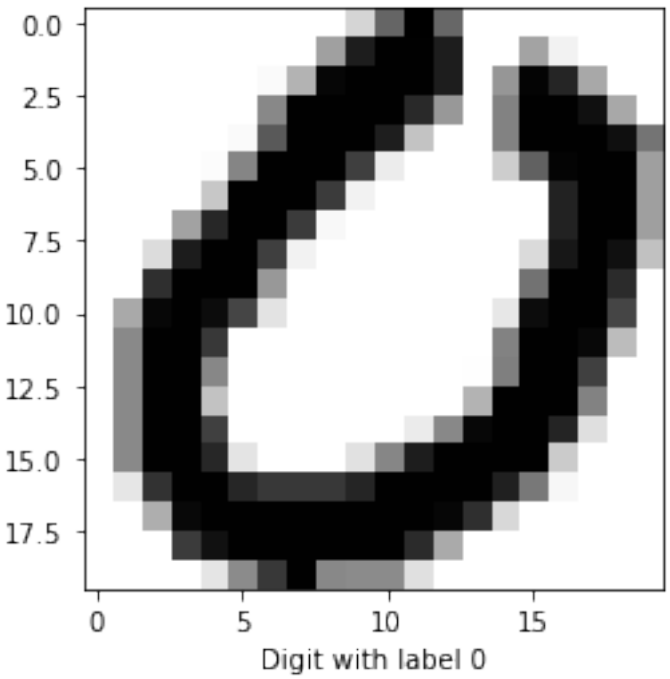
In this problem we consider the simplified setting of classification where we want to classify data points from R^d into two classes. For a linear classifier, the space R^d is split into two parts by a hyperplane: All points on one side of the hyperplane are classified as one class and all points on the other side of the hyperplane are classified as the other class.

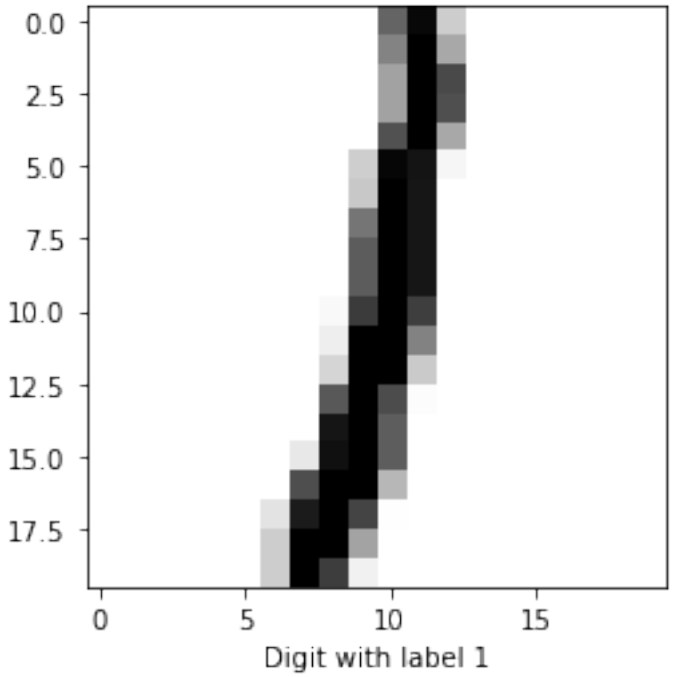
The goal of this problem is to show that even a regression technique like linear regression can be used to solve a classification problem. This can be achieved by regressing the data points in the training set against −1 or 1 depending on their class and then using the level set of 0 of the regression function as the classification hyperplane (i.e. we use 0 as a threshold on the output to decide between the classes).

(a) **Visualizing data:** The dataset used in this exercise is a subset of the MNIST dataset. The MNIST dataset assigns each image of a handwritten digit their value from 0 to 9 as a class. For this problem we only keep digits that are assigned a 0 or 1, so we simplify the problem to a two-class classification problem. Visualize three images that are labeled as 0 and three images that are labeled as 1.

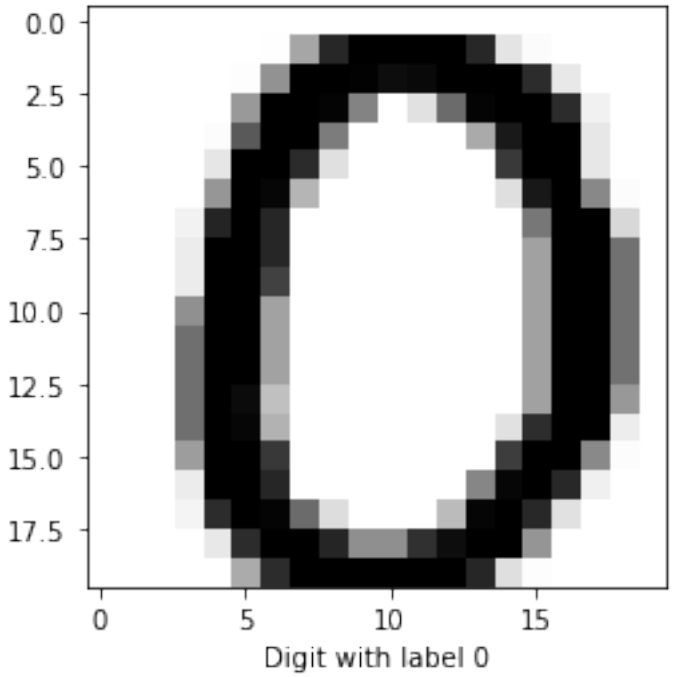
```
# Load the training and testing dataset
train_features = np.load("train_features.npy")
```

```
train_labels = np.load("train_labels.npy").astype("int8")

n_train = train_labels.shape[0]

def visualize_digit(features, label):
    # Digits are stored as a vector of 400 pixel values. Here we
    # reshape it to a 20x20 image so we can display it.
    plt.imshow(features.reshape(20, 20), cmap="binary")
    plt.xlabel("Digit with label " + str(label))
    plt.show()

# TODO: Plot three images with label 0 and three images with label 1
zerosf = []
onesf = []
for i in range(len(train_features)):
    f = train_features[i]
    l = train_labels[i]
    if(l == 1):
        onesf.append(f)
    elif(l == 0):
        zerosf.append(f)
for i in range(3):
    visualize_digit(zerosf[i], 0)
    visualize_digit(onesf[i], 1)
```

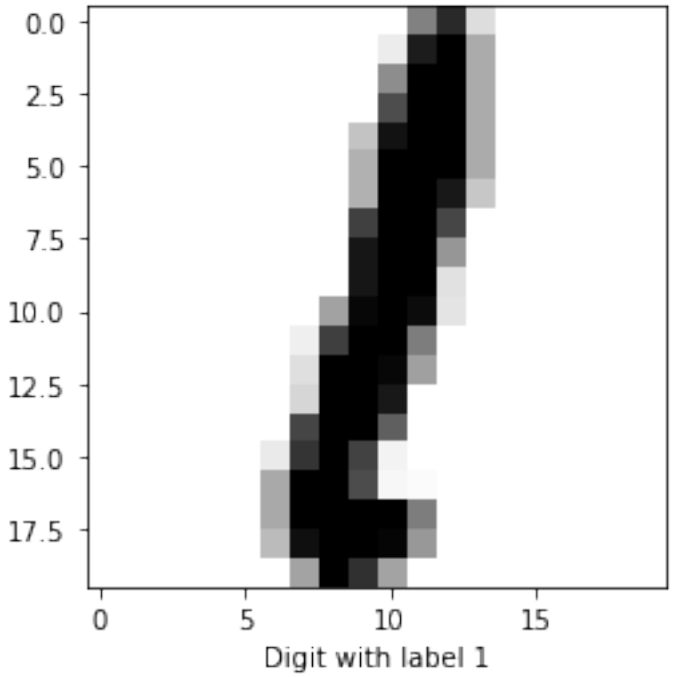




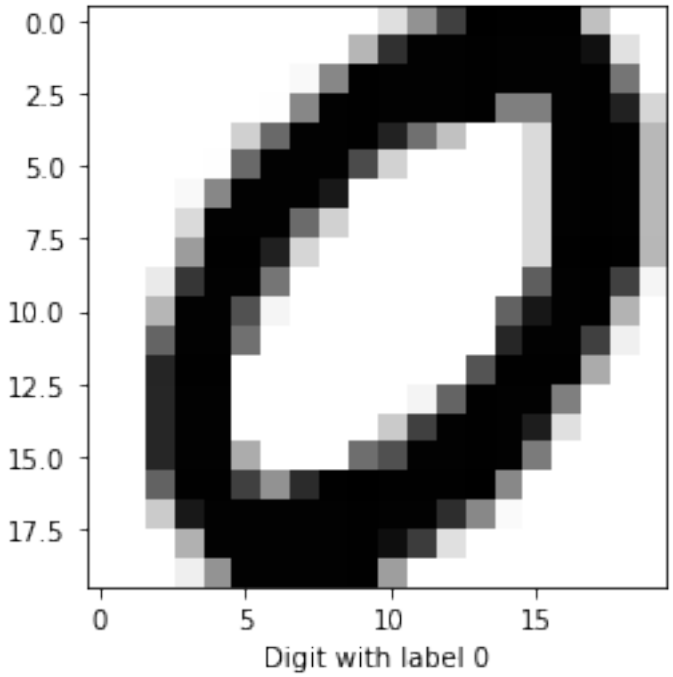
Digit with label 1



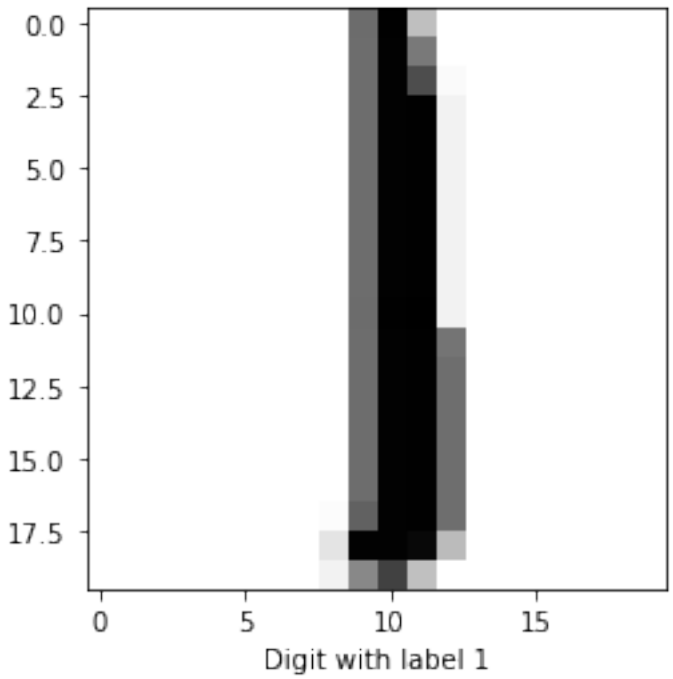
Digit with label 0



Digit with label 1



Digit with label 0



(b) **Solving linear regression problem**. We now want to use linear regression for the problem, treating class labels as real values y = −1 for class "zero" and y = 1 for class "one". In the dataset we provide, the images have already been flattened into one dimensional vectors (by concatenating all pixel values of the two dimensional image into a vector) and stacked as rows into a feature matrix X. We want to set up the regression problem $\min_w \| Xw - y \|^2$ where the entry y_i is the value of the class (−1 or 1) corresponding to the image in row x_i^T of the feature matrix. Solve this regression problem for the training set and report the value of $\| Xw - y \|^2$ as well as the weights w. For this problem you may only use pure Python and numpy (no machine learning libraries!).

```
# TODO: Solve the linear regression problem
ys = train_labels
for i in range(len(ys)):
    if(ys[i]==0):
        ys[i] = -1
ws = np.linalg.lstsq(train_features, ys, rcond=None)[0]

# TODO: Report the residual error and the weight vector
res = np.matmul(train_features, ws)
new_res = np.linalg.norm(res-ys)**2
print("Error: " + str(new_res))

print("Weight: " + np.array2string(ws))
```

```
Error: 422.75080869213656
Weight: [-3.30801139e-01  3.91726636e-01  1.48153634e-01 -1.60602318e-
01
  1.03277284e-01 -1.96941103e-02 -1.27705114e-01  9.45889276e-03
```

-1.71493893e-02 -5.67518079e-03 -4.69069786e-03 -1.12641776e-02
-5.71029120e-03 4.59008401e-03 1.78762857e-02 -3.00976494e-02
 1.00373536e-02 -6.45811914e-02 -2.30413879e-02 -2.63012485e-02
-1.63458898e-01 4.67076721e-01 -2.82694076e-03 -1.05642710e-01
-1.80703185e-01 1.34413187e-01 -5.09450682e-03 -3.07269166e-02
-6.59225173e-02 1.76021545e-02 -3.06512637e-02 -6.22918718e-03
 1.54059878e-02 -3.13653554e-02 -2.52826819e-03 -6.17331537e-03
-1.02623406e-03 5.19405643e-02 3.95632027e-02 6.29594620e-02
-3.88408603e-01 -2.16922478e-01 -9.80431113e-02 1.68330198e-01
-5.70891161e-02 1.50769252e-02 2.43554088e-03 2.72974893e-02
 6.30415690e-02 -2.69231967e-02 7.73327627e-03 -1.71158562e-02
-5.02592486e-02 8.06240492e-03 -6.38849770e-03 -1.26750594e-02
 1.55050383e-02 -1.06068843e-02 -7.69542917e-03 -4.18694971e-02
 5.26520951e-01 2.31919518e-01 -8.41796994e-02 9.90934564e-02
-3.96223757e-02 -2.31646771e-02 6.78329096e-03 -5.28093879e-02
 2.26991410e-02 -1.84079900e-02 2.87996076e-03 -1.06359104e-02
 2.26940906e-02 -3.03847988e-02 -1.73679042e-02 2.12025185e-02
 3.33768909e-02 -3.48220335e-02 -2.71485106e-02 -6.14572381e-02
-2.08387560e-01 -7.03671783e-02 -6.45086500e-02 -5.42103612e-02
 2.14511581e-02 -2.83405066e-02 -2.76361408e-02 1.08086806e-02
-3.84239993e-02 2.86567441e-02 -2.90154300e-02 5.93136853e-03
-7.47531064e-03 -9.10032760e-03 -1.19273841e-02 -1.55744011e-02
-2.03919741e-02 -6.55518087e-03 4.85632938e-02 -3.95774272e-02
-3.30413742e-02 1.84931206e-02 8.31143893e-02 9.48879907e-03
 1.04639995e-02 2.05530104e-02 -2.89796769e-02 1.20898556e-02
 1.13589550e-02 -1.36114834e-02 1.48536638e-02 4.62730992e-03
 3.68167942e-03 7.35567039e-03 2.30061097e-02 -1.49417000e-02
-5.77568488e-02 2.55881175e-02 -1.23854511e-02 -3.55654726e-02
 5.60248488e-02 -7.86040799e-02 -5.50626530e-02 -3.80891487e-03
 2.05816648e-02 -8.04723684e-03 7.83590240e-04 5.85386507e-03
-3.78688890e-02 2.50237081e-04 -4.85700060e-03 7.12657738e-03
-6.08274231e-03 -1.06570756e-02 -4.51331887e-02 7.35493612e-03
-5.78687059e-04 -3.72889917e-02 2.24066792e-02 -3.55815656e-02
 1.25211773e-01 3.23686421e-02 2.73657229e-02 -3.78581831e-02
-2.84901406e-02 -2.55289913e-03 -3.19249774e-02 3.13738975e-02
 4.19972229e-02 7.49288537e-03 2.51217726e-02 6.45677307e-03
 6.20250983e-03 -2.07480714e-03 5.90324803e-03 -8.57605436e-03
 2.90282976e-03 2.02997888e-02 -8.59505835e-02 -8.97488032e-03
-5.60652860e-02 5.42657208e-02 -1.96536505e-02 5.73922649e-02
-2.16765556e-02 7.90975732e-03 -5.28758320e-03 -7.76409307e-02
-5.07018915e-02 -2.74831121e-02 6.07874106e-02 4.88354241e-02
-2.40432308e-02 -4.98371695e-02 -4.93468612e-03 -4.80952810e-02
-4.14496906e-02 -6.00526515e-02 -1.63114576e-02 -5.78013508e-02
-1.04833520e-01 -5.49018706e-02 1.14533924e-02 -4.62900803e-02
 9.94957314e-03 -2.29688096e-03 -5.04550302e-02 6.33387080e-02
 8.83330138e-03 1.28803089e-02 5.98684110e-02 5.08413487e-02
 5.55651710e-02 3.24115536e-02 1.85714205e-03 -6.51329564e-03
 2.07802344e-02 4.16188717e-02 5.11904721e-02 8.70418106e-03
 3.71278576e-03 -3.43094081e-02 -3.77520298e-02 2.70951558e-02
-5.88876207e-03 -1.95807115e-02 9.65403176e-03 -7.81864039e-02

```
 2.13194931e-02 -1.35682550e-01  4.39434232e-01 -8.18695552e-02
 5.43932286e-02 -3.66796531e-02 -5.72935866e-02  4.75733136e-02
-8.33187932e-03 -7.52157625e-02 -2.67535161e-02 -4.92983061e-02
 3.49869470e-02  5.02427662e-02  6.05641718e-02 -3.81384136e-02
 2.50290061e-03 -1.11001809e-02 -6.17957606e-02 -2.07922646e-02
 7.77941531e-03  9.29618071e-02  3.28980264e-01 -5.16301737e-02
-1.47197085e-02 -8.32066068e-02 -7.43272430e-02 -7.29191078e-02
-4.11997692e-02  3.09923664e-02  1.29082324e-02 -2.84667028e-02
-3.92752223e-03 -1.00523373e-01 -3.35037819e-02  1.55451901e-03
-7.21693812e-04 -2.80621774e-02  5.95076351e-03  3.56638788e-02
 1.08803683e-02  1.07424371e-01  4.42845123e-02 -1.00376145e-03
 1.39013616e-03 -8.00528598e-02 -3.36564930e-02 -7.74051769e-03
-8.94936296e-03 -2.00021638e-02  2.30867327e-02 -7.37627685e-02
 8.12704123e-03  4.37768234e-02 -3.59035407e-02 -3.68318177e-02
-6.17194519e-02 -3.72650641e-02 -5.80592050e-02 -1.79030156e-02
 5.28741089e-02  7.57543808e-04  4.68768911e-02  3.64770540e-02
 3.71717313e-02  3.71803807e-02  3.30825393e-02 -3.52975601e-02
-8.89291519e-02 -2.42449208e-02 -7.89638026e-02  5.99963913e-02
-8.14004893e-02 -3.61159932e-02 -1.25758026e-02 -2.42134562e-02
 5.14482082e-03  1.78411742e-02  2.22024038e-02  2.82196190e-03
 3.35291036e-02 -4.93128634e-02  2.01923654e-02  3.94619986e-02
-2.18392008e-02 -9.06847488e-02 -4.53201942e-03 -2.45190680e-02
 1.84695541e-02 -1.53101240e-02 -1.20567957e-02 -1.18375088e-01
 2.87997967e-02 -1.90534418e-02 -5.65406808e-02  1.42896673e-02
-2.23252171e-02 -8.45870495e-03 -2.25759114e-02  2.53210363e-02
-3.29712991e-02  2.33254148e-02  1.05684370e-02  1.18440323e-02
 3.81859163e-02  2.81853583e-02 -2.45859748e-03  5.40199647e-03
-4.95017511e-04  1.93879100e-02  5.34837926e-02 -5.50344906e-02
-1.27019215e-01 -1.13498793e-01  7.43925649e-02 -6.41129177e-02
-2.47320306e-02  1.66997781e-02  2.09302835e-02  1.21599431e-02
 4.45490968e-02 -1.40928016e-02  1.58415125e-02  2.01893090e-02
-1.64967191e-02  1.85763552e-02  2.12733554e-02  7.97269613e-03
-3.30506313e-02 -1.31742761e-01 -5.20436949e-02 -6.18674093e-03
 4.09978294e-02  6.37197178e-02  1.58207202e-02  8.63952766e-02
 4.42899685e-02 -2.31190507e-02 -8.82104047e-03 -3.39890869e-02
 1.49178533e-02  4.80800804e-02  1.25205322e-02 -2.58488228e-03
 3.72429550e-02 -2.24007228e-03  5.62991911e-02  9.65398717e-03
 1.01850966e-01  8.83478617e-02 -4.26438979e-02  2.76072504e-01
 5.16144280e-02  5.58907411e-02  1.50355680e-02 -7.85846737e-03
-1.79312925e-02  5.78407044e-03 -1.39456749e-02 -2.94643055e-02
-7.18814483e-02 -6.41082120e-02  2.71522711e-03 -2.76250116e-02
-3.69620130e-02  7.89809590e-03  7.59364737e-02 -8.44839593e-03
 1.20683508e-02  1.71199664e-01 -3.02479371e-01  2.80118222e-03
-2.34505772e-03 -3.16537319e-02  3.73912235e-03 -3.21984594e-02
 1.20262298e-02 -4.40909214e-03  1.04539715e-02  3.74839855e-03
 1.62244016e-02 -3.36821028e-03 -4.06860536e-02 -1.61140971e-02
-6.13231934e-03 -3.09129487e-02 -5.01960458e-02  1.58634491e-02
-1.27863763e-01  1.68131755e-01 -2.77894244e-01  4.19035954e-02]
```

(c) **Evaluating Learned Model.** Given a new flattened image x, one natural rule to classify it is the following one: It is a zero if $x^\top w \leq 0$ and a one if $x^\top w > 0$. Report what percentage of the digits in the training set are correctly classified by this rule. Report what percentage of the digits in the test set are correctly classified by this rule.

```
# Load the test dataset
# It is good practice to do this after the training has been
# completed to make sure that no training happens on the test
# set!
test_features = np.load("test_features.npy")
test_labels = np.load("test_labels.npy").astype("int8")

n_test = test_labels.shape[0]
```

```
(11623,)
(2115,)
(11623,)
(2115,)
% of correct training digits: 53.00696894089306
% of correct test digits: 99.81087470449172
```

```
# TODO: Implement the classification rule and evaluate it
# on the training and test set
res_test = np.matmul(test_features, ws)
for i in range(len(res)):
    if(res[i]<=0):
        res[i] = 0
    else:
        res[i] = 1
for i in range(len(res_test)):
    if(res_test[i]<=0):
        res_test[i] = 0
    else:
        res_test[i] = 1

train_err = (res == train_labels).sum()/(train_labels.shape[0])*100
test_err = (res_test == test_labels).sum()/(test_labels.shape[0])*100

print("% of correct training digits: " + str(train_err))
print("% of correct test digits: " + str(test_err))
```

```
% of correct training digits: 99.75909833949926
% of correct test digits: 99.81087470449172
```

Compare with Logistic Regression

```
# Logistic Regression

# You can also compare against how well logistic regression is doing.

import sklearn.linear_model
```

```
X = train_features
lr = sklearn.linear_model.LogisticRegression()
lr.fit(X, train_labels)

test_error_lr = 1.0 * sum(lr.predict(test_features) != test_labels) /
n_test
print("Logistic Test Accuracy," + str(1 - test_error_lr))
```

```
Logistic Test Accuracy,0.9995271867612293
```