Open in Colab

Q1: One-vs-All (OVA) Logistic Regression for Handwritten Digits

preprocess colab and data files.

```
import numpy as np
import scipy.io as sio
import matplotlib.pyplot as plt
from tqdm import tqdm
```

```


from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
%cd /content/drive/MyDrive/24787/hw4
```

```
/content/drive/MyDrive/24787/hw4
```

```
%ls
```

```
digits.mat  MLAI24787_hw04_2022Spring.pdf  train.txt  w.npy
hw4.ipynb   test.txt                       w_all.npy  w_sk.npy
```

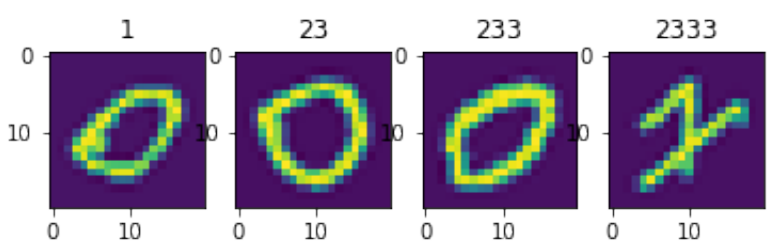
(a) Load data

```
data = sio.loadmat("digits.mat")

x = data['X']
y = np.squeeze(data['y']).reshape((-1,1))

np.place(y,y==10,0) #replace 10 with 0 in labels
numExamples = x.shape[0]
numFeaturs = x.shape[1]
numLabels = 10 #10 class
```

```
print(f"numExamples {numExamples} numFeaturs {numFeaturs} numLabels {numL
abels} y.shape {y.shape}")
```

```
numExamples 5000 numFeaturs 400 numLabels 10 y.shape (5000, 1)
```

```
range1 = [0,22,232,2332]
fig, axs = plt.subplots(1, 4)
# axs[0, 0].plot(x, y)
# axs[1, 1].scatter(x, y)

for idx, i in enumerate(range1):
  pic = x[i,:].reshape((20,20))
  axs[idx].imshow(pic)
  axs[idx].set_title(f"{i+1}")
plt.show()
```



(b)(Training the OVA classifier with gradient descent)

```
def sigmoid(z):

  return 1 / (1 + np.exp(-z))

def cost(theta, X, y):

  predictions = sigmoid(X @ theta)
  predictions[predictions == 1] = 0.999 #log(1)=0 causes error in division
  error = -y * np.log(predictions) - (1 - y) * np.log(1 - predictions)
  return sum(error) / len(y);

def costGradient(theta, X, y):

  predictions = sigmoid(X @ theta)
  # print(f"predictions.shape {predictions.shape} X.shape {X.shape} X.T @ p
  return X.transpose() @ (predictions - y) / len(y)
```

define splitdata function: split data to train and test

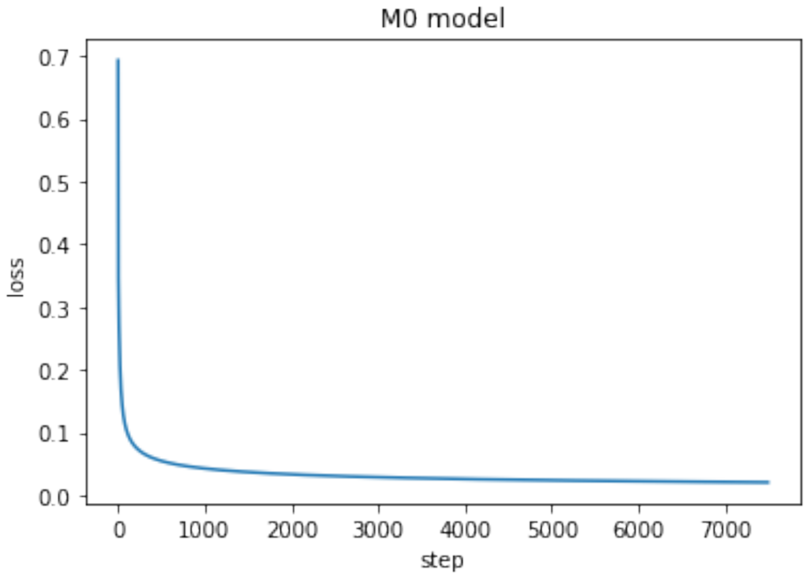
```
def splitdata(x,y):
  train_x = np.empty((0,numFeaturs))
  train_y = np.empty((0,1))
  val_x = np.empty((0,numFeaturs))
  val_y = np.empty((0,1))
  for i in range(10):
    train_x = np.vstack((train_x,x[i*500:(i+1)*500-100,:]))
    train_y = np.vstack((train_y,y[i*500:(i+1)*500-100,:]))
    val_x = np.vstack((val_x,x[i*500+400:(i+1)*500,:]))
    # print(val_x.shape)
    val_y = np.vstack((val_y,y[i*500+400:(i+1)*500,:]))
  return train_x,train_y,val_x,val_y

train_x,train_y,val_x,val_y = splitdata(x,y)
print(f"train_x {train_x.shape} train_y {train_y.shape} val_x {val_x.shap
e} val_y {val_y.shape}")
```

```
train_x (4000, 400) train_y (4000, 1) val_x (1000, 400) val_y (1000, 1)
```

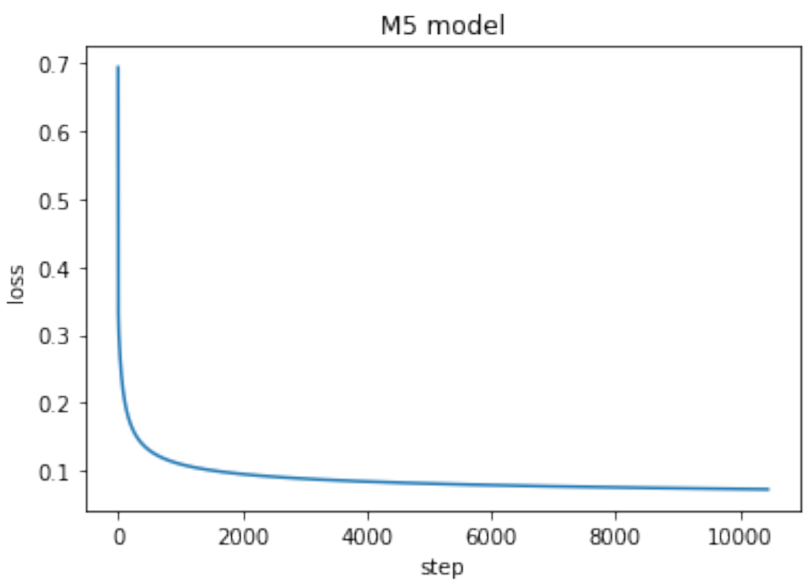
```
def plotloss(losslst,id=None):
    plt.plot(np.arange(len(losslst)), losslst)
    plt.title(f"M{id} model")
    plt.xlabel("step")
    plt.ylabel("loss")
```

```
    plt.show()

def train(X,Y,class_idx,step=0.1,iter=1e6,threshold=1e-6,e=1e-8,ifplot =
False,id=None):
    X = np.hstack((X,np.ones((X.shape[0],1))))
    Z = np.zeros((Y.shape))
    Z[Y==class_idx] = 1
    Z[Y!=class_idx] = 0
    error  = 1e5
    count = 0
    prev = 0
    J_lst = []
    # w = np.array([[-65],[0],[0]],dtype=float)
    theta = np.zeros((numFeaturs+1,1))
    while error > threshold and count<iter:

        J = cost(theta,X,Z)
        G = costGradient(theta,X,Z)
        # print(f"G.shape {G.shape}. theta.shape {theta.shape}")
        theta -= step*G
        error = np.abs(J-prev)
        prev = J
        J_lst.append(J)
        count+=1
        # print(f"loss {J}")
    if ifplot:
      plotloss(J_lst,id=id)

    return theta

# W = np.zeros((numFeaturs,1))
w_all = np.empty((numFeaturs+1,0))

for i in tqdm(range(10)):
  if i==0 or i==5:
    # w_all[str(i)] = train(train_x,train_y,i,ifplot=True)
    w_all = np.hstack((w_all,train(train_x,train_y,i,ifplot=True,id=i)))
  else:
    w_all = np.hstack((w_all,train(train_x,train_y,i,ifplot=False)))
  # w_sk =
np.save('w_all.npy',w_all)
```

```
  0%|          | 0/10 [00:00<?, ?it/s]
```




```
 50%|█████     | 5/10 [05:26<05:37, 67.54s/it]
```




```
100%|██████████| 10/10 [12:01<00:00, 72.11s/it]
```

prediction

```
# w_mat = np.empty((numFeaturs+1,0))
# for i in range(10):
#   w_mat = np.hstack((w_mat,w_all[str(i)]))
# print(w_mat.shape)
```

```
(401, 10)
```

generate prediction for the $[1,23,233,2333]$ images.

```


def predict(x,w_mat):
  predictions = np.empty((numExamples,0))
  x = np.hstack((x,np.ones((x.shape[0],1))))
  for i in range(10):
    # theta = w_all[str(i)]
    prob = sigmoid(x @ w_mat[:,i]).reshape((-1,1))

    predictions = np.hstack((predictions,prob))

  pred = np.argmax(predictions,axis=1).reshape((-1,1))
  return pred

pred = predict(x,w_all)

for i in range1:
  print(f" {i+1}: {pred[i]}\n ")

```

```
 1: [0]

 23: [0]

 233: [0]

 2333: [4]
```

calculate accuracy for my trained model.

```
def pred_acc(x,y,w_mat):
  x = np.hstack((x,np.ones((x.shape[0],1))))
  predictions = sigmoid(x @ w_mat)
  pred = np.argmax(predictions,axis=1).reshape((-1,1))
  # print(pred)
  acc = sum(pred==y)/len(y)
  # print(acc)
  # print("acc: {:.2f}%".format(acc))
  return acc[0]
train_acc = pred_acc(train_x,train_y,w_all)
test_acc = pred_acc(val_x,val_y,w_all)
whole_acc = pred_acc(x,y,w_all)
print("train acc: {:.2f}%".format(train_acc*100))
print("test acc: {:.2f}%".format(test_acc*100))
# print("whole acc: {:.2f}%".format(whole_acc*100))
```

```
train acc: 93.23%
test acc: 90.10%
```

calculate accuracy using sklearn.

```
from sklearn import linear_model

# pad_train_x =
def sk_train(X,Y,class_idx):
    X = np.hstack((X,np.ones((X.shape[0],1))))
    Z = Y.copy()
    Z[Y==class_idx] = 1
    Z[Y!=class_idx] = 0
    # print(Z)
    clf = linear_model.LogisticRegression(penalty="l2", solver="liblinear",
    clf.fit(X,Z.ravel())
    w = clf.coef_.reshape(-1,1)
    # score = clf.score(X,Z)
    # print(score)
    return w

w_sk = np.empty((numFeaturs+1,0))
for i in range(10):
  # print(i)
  w_sk = np.hstack((w_sk,sk_train(x,y,i)))
np.save('w_sk.npy',w_sk)
```

```
# w_sk = w_sk.reshape((-1,1))
train_acc = pred_acc(train_x,train_y,w_sk)
test_acc = pred_acc(val_x,val_y,w_sk)
whole_acc = pred_acc(x,y,w_sk)
print("train acc: {:.2f}%".format(train_acc*100))
print("test acc: {:.2f}%".format(test_acc*100))
# print("whole acc: {:.2f}%".format(whole_acc*100))
```

```
train acc: 91.90%
test acc: 91.90%
```

Here we could notice that the accuracy of my model is very similar to the accuracy of using sklearn linear model. Thus, my model could be verified useful.

Q2 : Data Normalization and Error

```
test_data = np.loadtxt("test.txt")
train_data = np.loadtxt("train.txt")
print(f"test_data {test_data.shape} train_data {train_data.shape}")
```

```
test_data (500, 3) train_data (1000, 3)
```

```
# test_data = np.hstack((test_data, np.ones((test_data.shape[0],1))) )
# train_data = np.hstack((train_data, np.ones((train_data.shape[0],1))) )
```

```
def splitdata(data):
  x = data[:,:2]
  y = data[:,-1]
  return x,y
```

```
train_x, train_y = splitdata(train_data)
test_x, test_y = splitdata(test_data)
```

use map_feature method to map input to a cubic function.

```
def map_feature(X,feature_num=3):
  x1 = X[:,0].reshape(-1,1)
  x2 = X[:,1].reshape(-1,1)
  # feature_lst = []
  count = 0
  feature_lst = None
  for j in range(feature_num+1):
    for k in range(j+1):
      if count ==0:
        feature_lst = (x1**(k))*(x2**(j-k))
      else:
        feature_lst= np.hstack((feature_lst,(x1**(k))*(x2**(j-k))))
      count +=1
  print(feature_lst.shape)
  return feature_lst
```

```
train_X_cubic = map_feature(train_x)
test_X_cubic = map_feature(test_x)
# print(train_X_cubic)
```

```
(1000, 10)
(500, 10)
```

(a) no-standardization, output the vector of

eigenvalues of the A^TA matrix

```
eig, _ = np.linalg.eig(train_X_cubic.T @ train_X_cubic)
ratio = max(eig) / min(eig)
print(f"eig {eig}")
print(f"ratio {ratio}")
```

```
eig [0.00000000e+00 3.69838267e+27 2.55647952e+20 1.84299854e+17
 2.39035796e+13 7.29695889e+09 1.18665615e+07 2.25891639e+06
 4.41363216e+04 4.45802681e+02]
ratio inf


/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: RuntimeWa
rning: divide by zero encountered in double_scalars
```

Here we could notice that since the min eigenvalue is 0, thus the ratio between the largest and the smallest is infinity. Since the min eigenvalue is 0, thus the ratio is inf and connot be computed.

(b) no-standardization, output your model's prediction

```
w = np.linalg.inv(train_X_cubic.T @ train_X_cubic) @ train_X_cubic.T @ tr
ain_y.reshape((-1,1))
print(f"w {w.shape}")
```

```
w (10, 1)
```

```
pred = test_X_cubic @ w
print(f"prediction: \n{pred.ravel()}")
```

```
prediction:
[5.19119605e+13 4.44431016e+13 6.43861773e+13 6.62406132e+13
 5.70370896e+13 4.92107632e+13 3.81858722e+13 2.69819279e+13
 3.57122013e+13 7.09070992e+13 2.80846127e+13 3.27958990e+13
 3.61554203e+13 5.22253082e+13 2.40993940e+13 3.08835113e+13
 5.25381909e+13 2.26480638e+13 4.28626202e+13 4.31175501e+13
 4.74734167e+13 4.74765899e+13 2.76043985e+13 2.32318219e+13
```

```
2.47151402e+13 2.96180614e+13 7.01054900e+13 5.94107521e+13
2.49779271e+13 6.81651503e+13 3.61472227e+13 5.83745257e+13
5.57094520e+13 5.37825273e+13 3.84248885e+13 4.13350594e+13
5.16120201e+13 6.70176447e+13 5.77020342e+13 5.19096790e+13
2.29693468e+13 6.32855007e+13 4.66388426e+13 6.85340933e+13
3.86630957e+13 5.63651765e+13 2.89219289e+13 6.14869570e+13
4.13280565e+13 3.57204196e+13 2.87535130e+13 6.66303930e+13
4.74878560e+13 5.16062827e+13 6.66273128e+13 6.93061759e+13
3.12594829e+13 4.13268432e+13 3.72710889e+13 2.57866999e+13
4.36444146e+13 2.66675882e+13 4.44463357e+13 6.29166779e+13
2.28812584e+13 2.28057599e+13 5.19176115e+13 5.73578917e+13
5.73563827e+13 6.22025371e+13 6.07853162e+13 6.85283423e+13
3.24016970e+13 4.33882547e+13 3.65901195e+13 6.18638042e+13
2.94447383e+13 2.72824819e+13 3.44355883e+13 4.39128492e+13
4.23516215e+13 6.73796733e+13 6.29250741e+13 4.26001251e+13
3.34079370e+13 2.56589055e+13 4.71965383e+13 2.55177357e+13
6.04406572e+13 2.53713404e+13 4.03450317e+13 2.28849097e+13
3.05126316e+13 5.01049876e+13 2.77688754e+13 5.22319662e+13
4.83416320e+13 5.73742679e+13 3.96087531e+13 4.71968443e+13
4.86365987e+13 3.79586095e+13 4.08300982e+13 2.97903808e+13
2.59297499e+13 2.87543930e+13 2.44644687e+13 7.08885066e+13
2.47164001e+13 3.22124009e+13 2.77678718e+13 2.47169306e+13
5.73624196e+13 3.79579038e+13 4.08369523e+13 4.31189904e+13
7.00996494e+13 7.04927168e+13 4.13375814e+13 5.37804139e+13
5.16157019e+13 5.01072173e+13 3.10716740e+13 5.41107015e+13
4.18387426e+13 4.08266726e+13 4.77568724e+13 4.33872900e+13
3.77304059e+13 2.87596018e+13 5.10100987e+13 4.18266976e+13
3.98557938e+13 5.73678780e+13 3.55016900e+13 3.81879975e+13
7.09052258e+13 2.77607068e+13 3.59265767e+13 4.33744669e+13
2.32341747e+13 3.61608036e+13 2.39788104e+13 2.33316042e+13
6.77768258e+13 2.74434228e+13 2.28097796e+13 2.38701919e+13
3.18285694e+13 2.53799085e+13 5.37846410e+13 5.76966715e+13
5.77085652e+13 3.12521953e+13 5.83795784e+13 4.05863172e+13
2.92678630e+13 4.80431542e+13 4.36450923e+13 3.74922689e+13
3.68184045e+13 3.42241805e+13 5.77086818e+13 4.08274132e+13
5.19059859e+13 2.68156073e+13 6.29356988e+13 6.62599183e+13
2.92618546e+13 5.44313518e+13 3.93721064e+13 2.30464658e+13
4.23533305e+13 2.82573134e+13 5.25418074e+13 4.13383288e+13
3.40232343e+13 2.28170717e+13 2.33378575e+13 2.35375128e+13
4.69061843e+13 2.60861224e+13 2.37608954e+13 5.60437104e+13
3.35990909e+13 4.66406647e+13 4.60695540e+13 7.08846299e+13
6.62595346e+13 3.25962644e+13 3.50740065e+13 4.55360697e+13
6.51238628e+13 2.99703533e+13 4.63580644e+13 4.44502571e+13
6.54862314e+13 3.31976998e+13 6.14953476e+13 6.40265878e+13
2.68245651e+13 3.72657739e+13 3.34087476e+13 4.13353396e+13
```

```
2.63769809e+13 2.49693156e+13 3.70510536e+13 2.68195250e+13
3.96061218e+13 3.38072346e+13 4.44347750e+13 2.71234935e+13
4.20886902e+13 7.04856582e+13 3.28050233e+13 7.08840952e+13
4.15889556e+13 2.57926306e+13 2.38632624e+13 3.96158335e+13
4.71941927e+13 2.47083807e+13 2.71233526e+13 4.00854411e+13
6.04479937e+13 2.71346373e+13 4.72043943e+13 5.09948461e+13
2.44603205e+13 4.36398655e+13 2.38624210e+13 3.20102163e+13
4.26028876e+13 6.77712436e+13 4.05912076e+13 2.69767251e+13
5.83848679e+13 3.81832163e+13 5.25304130e+13 4.74742355e+13
4.36402526e+13 5.63760824e+13 4.15798608e+13 2.92649695e+13
6.93198746e+13 2.69723689e+13 4.49815219e+13 2.96169394e+13
4.13252566e+13 4.15743325e+13 2.85861663e+13 3.01565366e+13
4.44462377e+13 2.37534682e+13 5.31484055e+13 2.29620877e+13
5.37754098e+13 4.13236703e+13 2.26527567e+13 3.29931136e+13
3.46500316e+13 4.66371220e+13 5.90562838e+13 6.15039858e+13
4.39047817e+13 4.66270052e+13 6.22106264e+13 3.38087859e+13
2.34330318e+13 2.40991984e+13 2.85908433e+13 5.60511460e+13
2.63746263e+13 2.26834793e+13 3.20165937e+13 5.66998327e+13
4.80450111e+13 5.47545104e+13 3.23991579e+13 2.52335377e+13
6.25660803e+13 3.68261006e+13 3.29967279e+13 2.80867058e+13
6.93111800e+13 4.05891774e+13 3.61565309e+13 6.00850541e+13
5.97427433e+13 2.80897382e+13 3.52875992e+13 4.83301320e+13
2.28908318e+13 3.98475925e+13 3.74990925e+13 5.83833396e+13
4.80452174e+13 6.77661822e+13 3.36143842e+13 2.69811543e+13
3.81820656e+13 4.18430744e+13 4.23433649e+13 3.52749966e+13
2.36488259e+13 5.97360639e+13 5.70278343e+13 2.57981568e+13
4.92070936e+13 3.18318666e+13 6.81647592e+13 7.01068177e+13
2.60799358e+13 3.10621780e+13 3.25994514e+13 3.03301979e+13
4.00882754e+13 3.81941100e+13 4.05805992e+13 4.60825013e+13
3.54982302e+13 3.08822037e+13 5.76934082e+13 6.47512040e+13
2.39774467e+13 6.54905425e+13 5.80445937e+13 4.13239503e+13
2.49749219e+13 3.34043710e+13 2.43374286e+13 7.04952478e+13
2.59337864e+13 4.05816135e+13 5.57151464e+13 6.73880763e+13
2.96137988e+13 5.13131597e+13 4.74708582e+13 5.77014512e+13
4.20820773e+13 5.70397516e+13 5.12988170e+13 6.14987537e+13
3.40220035e+13 4.31276356e+13 3.98464084e+13 5.28512257e+13
5.06937218e+13 2.32351925e+13 5.10078420e+13 6.89400044e+13
4.41838393e+13 5.50682077e+13 3.68157253e+13 4.28628116e+13
2.33388791e+13 6.54926985e+13 2.30545012e+13 6.66348859e+13
3.70521828e+13 3.75066212e+13 6.07903850e+13 2.96206051e+13
6.18525658e+13 6.04512421e+13 2.60759518e+13 2.82486152e+13
7.08893088e+13 2.34347591e+13 2.43358543e+13 6.55032283e+13
6.62461086e+13 2.43440581e+13 6.40140920e+13 6.18467052e+13
5.16168933e+13 2.51101071e+13 2.69732118e+13 2.29724424e+13
5.44196869e+13 4.18441105e+13 2.28888151e+13 2.38672120e+13
```

```
 4.89273783e+13 6.70137783e+13 2.99688460e+13 2.80822316e+13
 2.29653690e+13 2.48370029e+13 3.26019221e+13 5.77041331e+13
 5.53909376e+13 6.81435240e+13 2.26475009e+13 4.52547666e+13
 3.03273114e+13 3.08843574e+13 3.28001399e+13 2.77662949e+13
 4.71952125e+13 2.74450579e+13 4.28585073e+13 2.72838976e+13
 4.77705448e+13 6.04524453e+13 3.48614328e+13 5.44212567e+13
 4.83440161e+13 2.96084176e+13 4.66432972e+13 4.92151680e+13
 5.57061502e+13 4.08331542e+13 4.00858067e+13 3.77205674e+13
 3.22082098e+13 4.05834578e+13 6.07839889e+13 3.50620395e+13
 3.32003620e+13 2.82460076e+13 2.74481869e+13 4.98063109e+13
 3.84209754e+13 2.62188967e+13 4.44512377e+13 2.51073594e+13
 2.79286128e+13 6.62600463e+13 6.36604602e+13 5.07036704e+13
 3.52833965e+13 3.03308817e+13 4.66310509e+13 5.13127281e+13
 4.74777161e+13 2.30519693e+13 6.81582427e+13 4.55162548e+13
 6.11433638e+13 4.18438279e+13 5.28503455e+13 3.98430389e+13
 5.97401189e+13 2.60729309e+13 5.83883952e+13 2.76058976e+13
 3.96025841e+13 2.45810886e+13 2.35353317e+13 6.22123428e+13
 5.10103137e+13 6.81599368e+13 4.69191857e+13 5.01014843e+13
 2.38731733e+13 3.96171957e+13 4.13392631e+13 3.86591654e+13
 3.18240968e+13 3.30029961e+13 2.80843240e+13 2.31471361e+13
 4.13320710e+13 3.06887471e+13 4.74862167e+13 3.79594036e+13
 3.42206408e+13 3.84200862e+13 4.66295336e+13 4.20814162e+13
 6.97124477e+13 2.26429369e+13 3.06992371e+13 4.83459860e+13
 2.76062545e+13 2.72903423e+13 5.73683426e+13 3.14427833e+13
 2.84220742e+13 2.60785618e+13 4.44466298e+13 2.82528181e+13
 4.03366705e+13 2.31375545e+13 5.67019071e+13 3.33987818e+13
 6.62436802e+13 2.53712730e+13 2.59336496e+13 3.48544297e+13
 6.73940254e+13 4.26049838e+13 2.56540099e+13 5.50575855e+13
 6.73851025e+13 2.92656372e+13 2.62222737e+13 3.34112611e+13]
```

(c) no-standardization, compute and output the root mean squared error

```
rms = np.sqrt(np.mean((pred-test_y.reshape(-1,1))**2,axis=0))
print(f"rms {rms}")
```

```
rms [18.91742004]
```

(d) standardization, output the vector of eigenvalues

of the A^TA matrix

```
def standardization(train_x,test_x):
  u = np.mean(train_x,axis=0)
  std = np.std(train_x,axis=0)
  print(f"u.shape {u.shape} std.shape {std.shape}")
  train_x_std = (train_x-u)/std
  test_x_std = (test_x-u)/std
  return train_x_std, test_x_std
```

```
train_x_std, test_x_std = standardization(train_x, test_x)
# test_x_std =  standardization(test_x)
print(f"train_x_std {train_x_std.shape}")
```

```
u.shape (2,) std.shape (2,)
train_x_std (1000, 2)
```

```
train_x_std_cubic = map_feature(train_x_std,feature_num=3)
test_x_std_cubic = map_feature(test_x_std,feature_num=3)
print(f"train_x_std_cubic {train_x_std_cubic.shape}")
```

```
(1000, 10)
(500, 10)
train_x_std_cubic (1000, 10)
```

```
# print(train_x_std_cubic)
```

```
eig, _ = np.linalg.eig(train_x_std_cubic.T @ train_x_std_cubic)
ratio = max(eig) / min(eig)
print(f"eig {eig}")
print(f"ratio {ratio}")
```

```
eig [6278.01972693 5438.37641409 3424.40744854 1080.47322434  210.3182749
2
  112.52071394  100.41022042  839.68902588  662.31826487  705.42569236]
ratio 62.52371223172815
```

This ratio is 62 and valid since the min eigenvalue is not zero.

(e) standardization, output your new model's prediction

```
w = np.linalg.inv(train_x_std_cubic.T @ train_x_std_cubic) @ train_x_std_
cubic.T @ train_y.reshape((-1,1))
print(f"w {w.shape}")
```

```
w (10, 1)
```

```
# print(w)
```

```
pred = test_x_std_cubic @ w
print(f"prediction:\n {pred.ravel()}")
```

```
prediction:
 [  45.84719945   50.55510691    9.47781833   23.62176654  -20.19232475
   49.60034084  119.99144611   17.3949887   124.87883488   87.62907344
  120.21445619  116.57534948   53.99790827   -2.5993565    61.82501524
   59.98939605  -23.86384406   45.7616979    41.46209503   94.03043283
   72.35192551   69.09409651   61.37915413    5.68083592   60.15787887
   66.68030402   13.10122093  -19.62814538   12.21877153   59.93585175
  118.84117668   23.84811001   40.52951898   43.79447986   63.30032367
   18.62917892  -10.69191041   68.98636334    5.65599243   54.66243249
   10.83823226   26.31228682   -7.72989686   30.71087425   19.33590504
   38.49654737  111.24757192  -14.12426599  100.44747041   24.77493105
   89.61506807   -5.17743807  -12.75180679   41.69450743    2.92541624
   -1.56793049   30.8315755   105.84753942   87.54482527  -38.46377949
   69.72541612  102.58021781   12.38501052  -31.9298015   -23.60325314
 -106.83697701   -4.53013077   20.54344647    0.21794713   23.26450198
   24.50419762   -2.76266289  123.96168744   -0.90564184  114.95527154
    8.77520573   37.44088992  109.19894137   76.41143613   18.92618488
    0.65211953   28.33741189   23.21946589   88.77625686   38.70173165
    8.97923098   38.00142376   19.79791101   17.48482776   56.41886516
    9.88641787   58.59661407  104.16148056  -18.81158425   21.61085589
  -14.92045705   11.23970311  -25.13723878   98.76730946   34.52078407
  -17.06268793   82.02615525   97.93854326  125.89045842    4.70518454
   74.98877834   34.12648313   36.31705163   38.97991587   85.26567399
   22.01692031   30.15902277   33.44295296   91.70267134   15.60887702
   83.46498595   30.59183166   35.50046244    3.58693484   46.86403319
  -23.58665934  -19.89117813   33.80896737  -26.70594531   23.25656136
  105.18103053   57.41622359    4.84094705   50.02434109   26.07534806
```

-4.27450526	59.23833728	27.15117686	-1.91285054	43.67632733
110.51017992	55.35505942	115.55837159	102.583091	69.3986121
62.08402948	25.71849883	63.17897699	61.53646862	9.90841574
96.43145727	53.64282575	26.21314099	41.23026778	22.13235995
30.90155581	34.59167326	-26.70921694	127.3217418	-11.3373351
57.95023843	81.07661398	53.38553008	61.81934631	105.6057538
106.39076356	114.95238429	-26.59955518	108.21382877	32.40913231
-17.28312213	-16.80033626	45.01609834	112.70561418	-21.96971445
53.74508354	-63.81765494	3.44161053	22.64169077	-12.83113638
4.05281096	28.70391718	-20.22087875	0.56885245	70.37811881
58.53328523	12.0411059	-8.40096156	-2.44109865	53.20097986
-13.25989396	67.73501042	16.84904319	38.30805791	77.20374151
25.60393731	6.9170603	18.28417519	105.78280531	7.05966955
-6.06145611	-27.85074167	131.91971455	19.82451484	-13.35315282
63.55221799	119.69819692	31.14378443	15.95058724	22.61518897
62.00015506	17.4535133	108.2138313	114.16179523	128.64449262
66.53398826	14.91906792	76.85001072	-8.85810725	47.70217235
11.3776959	2.82854149	96.82778127	31.76175299	13.74884299
62.13626211	0.61650782	7.40057764	27.50958084	-22.82735079
26.67423278	-11.66880357	58.73772313	84.79487819	93.76143201
-3.62451616	78.05759624	57.65065135	3.57872857	7.36016304
78.61673025	-23.82170739	96.36591774	37.16815632	75.09218374
94.64978054	-25.18688063	96.43303959	120.68208948	10.39530144
108.47563668	53.69561512	85.3774955	102.98283104	70.93507976
69.6071153	27.24749854	13.38408174	72.72995338	39.89423891
-0.72333557	-11.88463647	85.81868732	-20.67242938	58.09305319
25.01404801	5.89400457	23.26740469	-21.60293775	93.49974195
73.92253399	0.6215597	130.76480043	67.09440387	64.51025923
24.51909482	-18.38654931	57.24929193	-57.42408915	111.63619573
19.07006364	70.09177358	5.70568	62.45766062	-39.8295846
18.22889274	18.01758452	131.1750926	102.51549559	31.99088959
21.94213999	39.28279777	11.8080506	29.57134857	53.72933722
29.10332813	34.88909728	-15.64210033	113.11281977	82.82906507
-26.274222	70.94161791	23.21513323	27.47311828	18.53288891
69.68201616	3.87241342	86.66227772	75.36107652	19.1632941
-22.67575064	36.46063219	17.39743365	66.91334147	31.24329088
52.80866241	11.89350424	89.08575408	123.28288365	131.7717351
109.38929139	97.3840804	31.32657815	109.78816568	-7.44793639
94.52230033	81.1763418	23.00477652	25.12214871	24.50488894
21.54378901	-26.33009525	90.03287241	58.50889229	90.71388177
82.72536244	30.87284895	102.91913515	107.71841695	10.53066652
3.42438669	124.95779085	-8.22861742	42.54626136	10.1022813
81.27326104	-27.12492798	56.6516185	-0.37412805	40.00207202
-2.38177781	109.30211897	-24.20605123	59.33786549	64.85503025
-21.8019691	74.44527861	-1.30705544	-7.79530855	124.82184406

```
  39.01038595  -10.56603349   26.75076887   19.48467878   12.37977068
  19.48408192   19.98055529   19.14245913   32.43090446    8.06118496
 -16.2327391    87.16334373  121.47471091   36.83186483   62.96035636
  74.56813832   -9.5593513    18.73653099   -1.00870359   26.35349857
  26.71130406  -18.84693706   16.45330758  112.57370975  -19.47110248
  38.32520303   11.75286334   11.711716     70.15142168  -18.44651252
  11.19157413  123.17089068  100.58227331   60.05744538   -7.42110192
 121.45972401  -10.00387317    0.75218491   12.89308623   50.832143
  33.21880671  129.78792851   47.67000287  119.52117287   35.88700313
  52.43071184   70.35433126   89.01025419  114.8242553   -13.95469724
  -5.91429895   24.86414789   27.93571127   -9.75123552   30.15874684
   2.7388623     4.63170189   21.567991     61.76735094   40.37330533
  90.43931296  132.15385704   93.92077208   15.38928197  105.10230107
 118.0318855    88.29626165   25.37571256   -9.33801831  111.69638986
  -5.84118797   -1.25063452   59.81183713   28.50728426   47.37696786
 -15.91015283  -18.93726818   85.27036579   99.45772821   69.94083087
 -12.75010745   60.71003861   56.04855091    1.65423982   13.47232808
  11.96580084    8.97909074  -25.62380924   46.96138386   26.31569137
 -37.99114442   11.48792303   38.09199028   77.54131477  -19.94831396
  41.23651199   -9.22737091   -1.37563025    4.83836548   -7.48966001
   7.53710313   -7.70881306    9.21456785    8.70912282   67.34845969
 108.56968627   68.52615912  120.5194538   -13.60256169   55.45089057
  51.52404538  -15.13575507   70.57900071  127.74472422  117.1490812
  78.2208531    69.03452276   10.47828987  -36.5159516    53.08333186
 -17.55183793   33.50079221   29.67718211   -5.42980283  106.55344262
  27.60208044  102.37372414    9.4953915    75.98953311  104.10224393
  -4.94302813    2.80768989  126.17291339   26.31826865   54.21986641
 102.69946256  114.20625831    7.59201996   30.71605378   67.00025063
  30.95319874   15.59815517  114.01471585  100.92694202   29.87571418]
```

```
# print(pred)
```

(f) standardization, compute and output the root mean squared

```
rms = np.sqrt(np.mean((pred-test_y.reshape(-1,1))**2,axis=0))
print(f"rms {rms}")
```

```
rms [18.91742004]
```

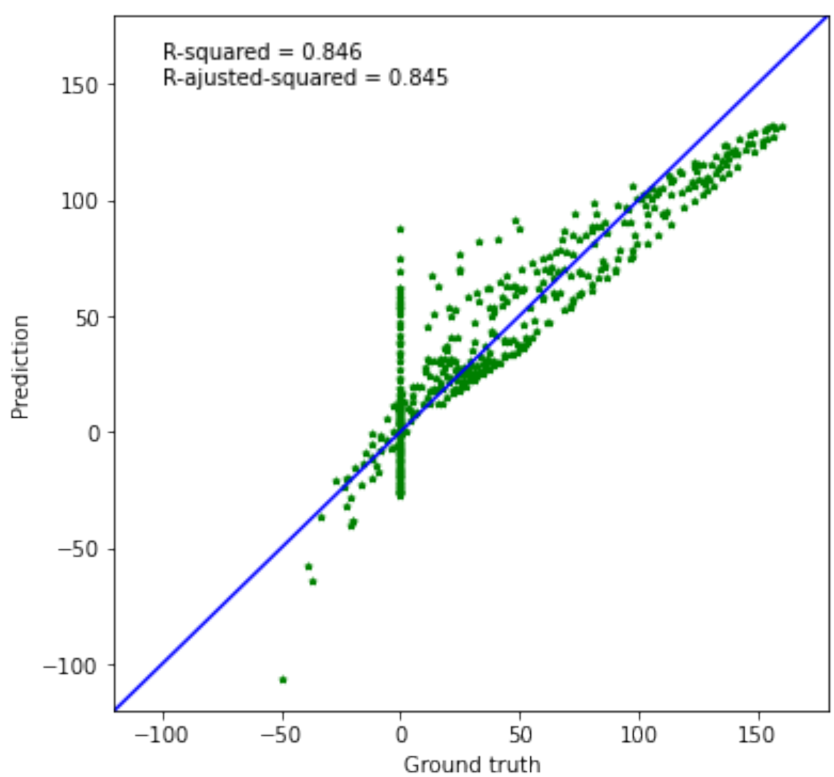
(g) visualize the ground-truth vs. your model's prediction on a square-axis R-squared plot s

```
def R_cal(pred,test_y):
  ssres = np.sum((pred-test_y.reshape(-1,1))**2)
  y_mean = np.mean(test_y)

  sstot = np.sum((test_y.reshape(-1,1)-y_mean)**2)
  R_2 = 1- ssres/sstot
  return R_2
def R_adj_cal(pred,test_y):
  ssres = np.sum((pred-test_y.reshape(-1,1))**2)
  y_mean = np.mean(test_y)

  sstot = np.sum((test_y.reshape(-1,1)-y_mean)**2)
  R_2 = 1- ssres/sstot
  R_adj = 1-(1-R_2)*(len(pred)-1)/(len(pred)-1-2)
  return R_adj
```

```
R_2 = R_cal(pred,test_y)
print(f"R_2 {R_2}")
R_adj = R_adj_cal(pred,test_y)
print(f"R_adj {R_adj}")
```

```
R_2 0.8459503878948689
R_adj 0.8453304699387114
```

```
def r_plot(r_value, pred,test_y):
  # ax.set_aspect('equal', adjustable='box')
  # plt.figure(figsize=(6,6))
  fig, ax = plt.subplots(figsize=(6, 6))
  # ax.plot(test_y,test_y)
  ax.plot([0,1],[0,1],transform=ax.transAxes,c='blue')
  ax.scatter(test_y,pred,c="green",s=10,marker='*')
  ax.annotate("R-squared = {:.3f}\nR-ajusted-squared = {:.3f}".format(R_2,R

  plt.xlim([-120, 180])
  plt.ylim([-120, 180])
  plt.xlabel('Ground truth')
  plt.ylabel('Prediction')

  plt.show()
r_plot(R_2, pred,test_y)
```



(h) Plot your model’s predictions as a surface.

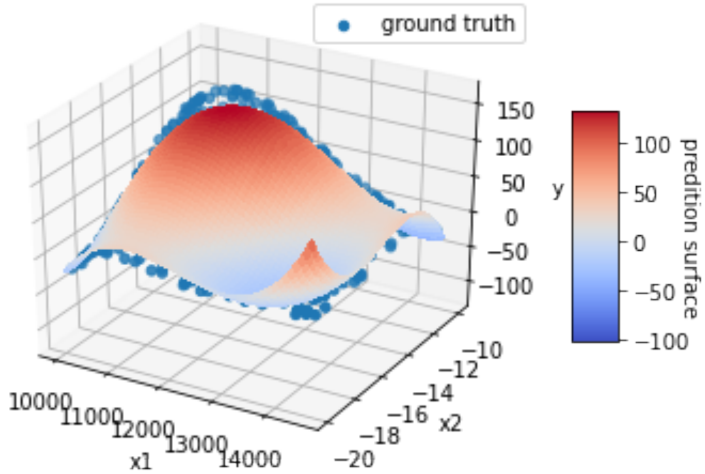
```
max_x =np.max(test_x,axis=0)
min_x =np.min(test_x,axis=0)
x1_before_mesh = np.linspace(min_x[0],max_x[0],100)
x2_before_mesh = np.linspace(min_x[1],max_x[1],100)
xx1,xx2 = np.meshgrid(x1_before_mesh,x2_before_mesh)
print(f"xx1.shape {xx1.shape}")
xx1_1 = xx1.reshape(-1,1)
xx2_1 = xx2.reshape(-1,1)
mesh_x = np.hstack((xx1_1,xx2_1))
_,mesh_std_x =standardization(train_x,test_x=mesh_x)
mesh_std_cubic_x = map_feature(mesh_std_x)
pred = mesh_std_cubic_x @ w
# plt.scatter(xx1_1,xx2_1,c=pred)

print(f"pred.shape {pred.shape}")
```

```
xx1.shape (100, 100)
u.shape (2,) std.shape (2,)
(10000, 10)
pred.shape (10000, 1)
```

```
print(f"xx1_1,xx2_1,pred {xx1_1.shape} {xx2_1.shape} {pred.shape}")
```

```
xx1_1,xx2_1,pred (10000, 1) (10000, 1) (10000, 1)
```

```
from matplotlib import cm

fig, ax = plt.subplots(subplot_kw={"projection": "3d"})

# Plot the surface.
surf = ax.plot_surface(xx1,xx2,pred.reshape(xx1.shape), cmap=cm.coolwarm,
                       linewidth=0, antialiased=False)
ax.scatter(test_x[:,0].reshape(-1,1),test_x[:,1].reshape(-1,1),test_y.res
hape(-1,1),label="ground truth")
ax.legend()
cbar =fig.colorbar(surf, shrink=0.5, aspect=5)
cbar.ax.set_ylabel('predition surface', rotation=270)
ax.set(xlabel='x1',ylabel="x2",zlabel="y")
plt.show()
```



Q3: K-Nearest Neighbors

(a). k=1, output your model’s prediction

```
def distance(point,train_x,k=1):
  distance_lst = (np.sum((train_x-point)**2,axis=1))**(0.5)
  # print(((train_x-point)**2).shape)
  # print(distance_lst.shape)
  # lst = np.argpartition(distance_lst,k)[-k:]
  lst = (distance_lst).argsort()[:k]
  # print(lst)
  # print(np.argpartition(distance_lst, -k))
  # print()
  # print(lst)
  # print(np.argmin(distance_lst))
  # print(lst)
  # return np.argmin(distance_lst)
  return lst
def knn(test_x,train_x,train_y,k=1):

  n = len(test_x)
  ans_y =[]
  for i in range(n):
    # print(train_y[distance(test_x[i],train_x,k=k)])
    # print(np.mean(train_y[distance(test_x[i],train_x,k=k)]))
    ans_y.append(np.mean(train_y[distance(test_x[i],train_x,k=k)]))
    # break
  ans_y = np.array(ans_y).reshape(-1,1)
  return ans_y
# print()
pred = knn(test_x_std,train_x_std,train_y)
# distance([1,1],train_x)
# print(pred)
```

```
print(f"prediction:\n {pred.ravel()}")
```

```
prediction:
 [  45.84719945   50.55510691    9.47781833   23.62176654  -20.19232475
   49.60034084  119.99144611   17.3949887   124.87883488   87.62907344
  120.21445619  116.57534948   53.99790827   -2.5993565    61.82501524
   59.98939605  -23.86384406   45.7616979    41.46209503   94.03043283
   72.35192551   69.09409651   61.37915413    5.68083592   60.15787887
   66.68030402   13.10122093  -19.62814538   12.21877153   59.93585175
  118.84117668   23.84811001   40.52951898   43.79447986   63.30032367
   18.62917892  -10.69191041   68.98636334    5.65599243   54.66243249
   10.83823226   26.31228682   -7.72989686   30.71087425   19.33590504
   38.49654737  111.24757192  -14.12426599  100.44747041   24.77493105
   89.61506807   -5.17743807  -12.75180679   41.69450743    2.92541624
```

-1.56793049	30.8315755	105.84753942	87.54482527	-38.46377949
69.72541612	102.58021781	12.38501052	-31.9298015	-23.60325314
-106.83697701	-4.53013077	20.54344647	0.21794713	23.26450198
24.50419762	-2.76266289	123.96168744	-0.90564184	114.95527154
8.77520573	37.44088992	109.19894137	76.41143613	18.92618488
0.65211953	28.33741189	23.21946589	88.77625686	38.70173165
8.97923098	38.00142376	19.79791101	17.48482776	56.41886516
9.88641787	58.59661407	104.16148056	-18.81158425	21.61085589
-14.92045705	11.23970311	-25.13723878	98.76730946	34.52078407
-17.06268793	82.02615525	97.93854326	125.89045842	4.70518454
74.98877834	34.12648313	36.31705163	38.97991587	85.26567399
22.01692031	30.15902277	33.44295296	91.70267134	15.60887702
83.46498595	30.59183166	35.50046244	3.58693484	46.86403319
-23.58665934	-19.89117813	33.80896737	-26.70594531	23.25656136
105.18103053	57.41622359	4.84094705	50.02434109	26.07534806
-4.27450526	59.23833728	27.15117686	-1.91285054	43.67632733
110.51017992	55.35505942	115.55837159	102.583091	69.3986121
62.08402948	25.71849883	63.17897699	61.53646862	9.90841574
96.43145727	53.64282575	26.21314099	41.23026778	22.13235995
30.90155581	34.59167326	-26.70921694	127.3217418	-11.3373351
57.95023843	81.07661398	53.38553008	61.81934631	105.6057538
106.39076356	114.95238429	-26.59955518	108.21382877	32.40913231
-17.28312213	-16.80033626	45.01609834	112.70561418	-21.96971445
53.74508354	-63.81765494	3.44161053	22.64169077	-12.83113638
4.05281096	28.70391718	-20.22087875	0.56885245	70.37811881
58.53328523	12.0411059	-8.40096156	-2.44109865	53.20097986
-13.25989396	67.73501042	16.84904319	38.30805791	77.20374151
25.60393731	6.9170603	18.28417519	105.78280531	7.05966955
-6.06145611	-27.85074167	131.91971455	19.82451484	-13.35315282
63.55221799	119.69819692	31.14378443	15.95058724	22.61518897
62.00015506	17.4535133	108.2138313	114.16179523	128.64449262
66.53398826	14.91906792	76.85001072	-8.85810725	47.70217235
11.3776959	2.82854149	96.82778127	31.76175299	13.74884299
62.13626211	0.61650782	7.40057764	27.50958084	-22.82735079
26.67423278	-11.66880357	58.73772313	84.79487819	93.76143201
-3.62451616	78.05759624	57.65065135	3.57872857	7.36016304
78.61673025	-23.82170739	96.36591774	37.16815632	75.09218374
94.64978054	-25.18688063	96.43303959	120.68208948	10.39530144
108.47563668	53.69561512	85.3774955	102.98283104	70.93507976
69.6071153	27.24749854	13.38408174	72.72995338	39.89423891
-0.72333557	-11.88463647	85.81868732	-20.67242938	58.09305319
25.01404801	5.89400457	23.26740469	-21.60293775	93.49974195
73.92253399	0.6215597	130.76480043	67.09440387	64.51025923
24.51909482	-18.38654931	57.24929193	-57.42408915	111.63619573
19.07006364	70.09177358	5.70568	62.45766062	-39.8295846

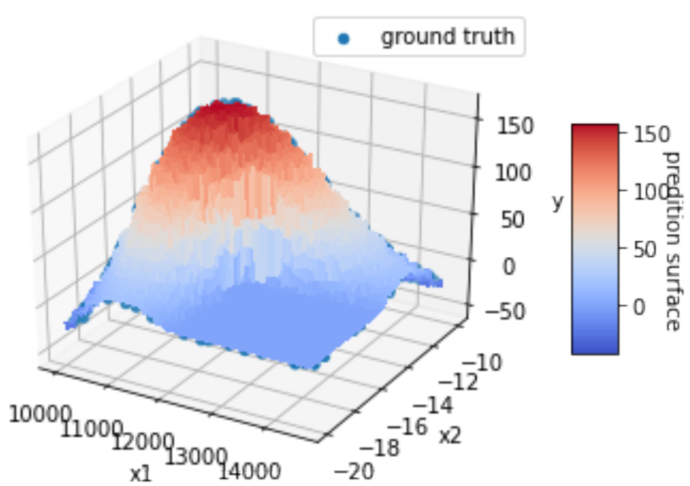
```
  18.22889274   18.01758452  131.1750926   102.51549559   31.99088959
  21.94213999   39.28279777   11.8080506    29.57134857   53.72933722
  29.10332813   34.88909728  -15.64210033  113.11281977   82.82906507
 -26.274222     70.94161791   23.21513323   27.47311828   18.53288891
  69.68201616    3.87241342   86.66227772   75.36107652   19.1632941
 -22.67575064   36.46063219   17.39743365   66.91334147   31.24329088
  52.80866241   11.89350424   89.08575408  123.28288365  131.7717351
 109.38929139   97.3840804    31.32657815  109.78816568   -7.44793639
  94.52230033   81.1763418    23.00477652   25.12214871   24.50488894
  21.54378901  -26.33009525   90.03287241   58.50889229   90.71388177
  82.72536244   30.87284895  102.91913515  107.71841695   10.53066652
   3.42438669  124.95779085   -8.22861742   42.54626136   10.1022813
  81.27326104  -27.12492798   56.6516185    -0.37412805   40.00207202
  -2.38177781  109.30211897  -24.20605123   59.33786549   64.85503025
 -21.8019691    74.44527861   -1.30705544   -7.79530855  124.82184406
  39.01038595  -10.56603349   26.75076887   19.48467878   12.37977068
  19.48408192   19.98055529   19.14245913   32.43090446    8.06118496
 -16.2327391    87.16334373  121.47471091   36.83186483   62.96035636
  74.56813832   -9.5593513    18.73653099   -1.00870359   26.35349857
  26.71130406  -18.84693706   16.45330758  112.57370975  -19.47110248
  38.32520303   11.75286334   11.711716     70.15142168  -18.44651252
  11.19157413  123.17089068  100.58227331   60.05744538   -7.42110192
 121.45972401  -10.00387317    0.75218491   12.89308623   50.832143
  33.21880671  129.78792851   47.67000287  119.52117287   35.88700313
  52.43071184   70.35433126   89.01025419  114.8242553   -13.95469724
  -5.91429895   24.86414789   27.93571127   -9.75123552   30.15874684
   2.7388623     4.63170189   21.567991     61.76735094   40.37330533
  90.43931296  132.15385704   93.92077208   15.38928197  105.10230107
 118.0318855    88.29626165   25.37571256   -9.33801831  111.69638986
  -5.84118797   -1.25063452   59.81183713   28.50728426   47.37696786
 -15.91015283  -18.93726818   85.27036579   99.45772821   69.94083087
 -12.75010745   60.71003861   56.04855091    1.65423982   13.47232808
  11.96580084    8.97909074  -25.62380924   46.96138386   26.31569137
 -37.99114442   11.48792303   38.09199028   77.54131477  -19.94831396
  41.23651199   -9.22737091   -1.37563025    4.83836548   -7.48966001
   7.53710313   -7.70881306    9.21456785    8.70912282   67.34845969
 108.56968627   68.52615912  120.5194538   -13.60256169   55.45089057
  51.52404538  -15.13575507   70.57900071  127.74472422  117.1490812
  78.2208531    69.03452276   10.47828987  -36.5159516    53.08333186
 -17.55183793   33.50079221   29.67718211   -5.42980283  106.55344262
  27.60208044  102.37372414    9.4953915    75.98953311  104.10224393
  -4.94302813    2.80768989  126.17291339   26.31826865   54.21986641
 102.69946256  114.20625831    7.59201996   30.71605378   67.00025063
  30.95319874   15.59815517  114.01471585  100.92694202   29.87571418]
```

```
# rms = np.sqrt(np.mean((pred-test_y.reshape(-1,1))**2,axis=0))
# print(f"rms {rms}")
```

(b). k=1, plot prediction and ground truth

```
pred = knn(mesh_std_x,train_x_std,train_y)
# print(f"prediction")
```

```
from matplotlib import cm
fig, ax = plt.subplots(subplot_kw={"projection": "3d"})

# Plot the surface.
surf = ax.plot_surface(xx1,xx2,pred.reshape(xx1.shape), cmap=cm.coolwarm,
                       linewidth=0, antialiased=False)
ax.scatter(test_x[:,0].reshape(-1,1),test_x[:,1].reshape(-1,1),test_y.res
hape(-1,1),label="ground truth")
ax.legend()
cbar =fig.colorbar(surf, shrink=0.5, aspect=5)
cbar.ax.set_ylabel('predition surface', rotation=270)
ax.set(xlabel='x1',ylabel="x2",zlabel="y")
plt.show()
```



(c). k=4 , output your model's prediction

```
# print(test_x_std[:5])
# print(train_x_std[:5])
# print(train_y[:5])
```

```
pred = knn(test_x_std,train_x_std,train_y,k=4)
# distance([1,1],train_x)
print(f"prediction:\n {pred.ravel()}")
```

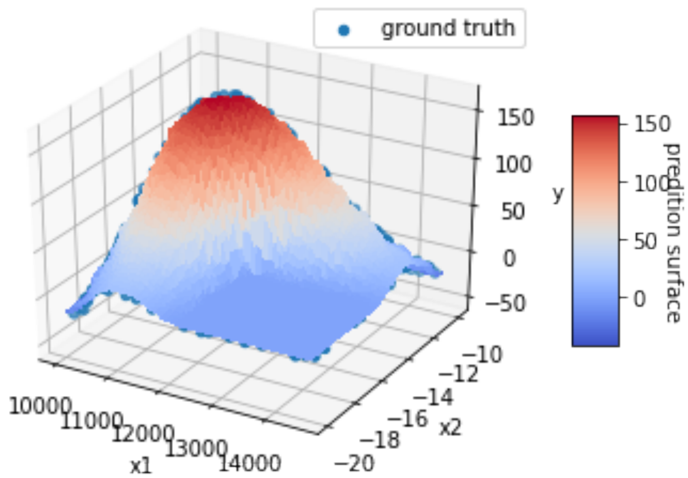
```
prediction:
 [ 47.82520075   1.94127595   3.53349     23.010314     0.
  33.16616875 143.6071275   17.25110925 144.553385     0.
 139.317855   122.1577575   32.56594375   0.          46.58921875
  76.93522625   0.          25.6603695    0.         100.38126775
  84.8478385   81.60874825  76.7378305  -10.31073568  65.5553395
  84.7474185    0.           0.          18.20339375   0.
 129.913825    34.7981695   57.53032975  56.48446725  18.506154
   0.           0.           0.           1.28746453  75.71514075
   5.1117646   32.5441315    0.          20.89536025   8.1929867
  57.53032975 137.56297    -10.05935588 108.4501775   22.79880075
 114.490785     0.           0.          39.166985     0.
 -10.3120267   37.338484   118.212585    46.2902115  -16.71533475
  32.04212145 104.35800725   0.         -18.039955   -26.779211
 -42.06734825   0.          25.04395275   2.17379664  20.61651525
  26.565107   -10.8642447  135.257495     0.         129.913825
   0.          48.557467   114.1255275   69.63465      0.
   0.          24.0293205   30.059337    71.79197225  45.65286675
   8.22519373   0.          23.29668625  16.28814475  46.269951
   1.34069895  28.26058575 127.42141      0.          16.52663043
   0.           0.           0.          71.27629925   0.
   0.          37.82122025  96.87074425 152.2188625  -13.50847975
  95.25886375  37.89297875  20.89536025  44.70490775  92.5671365
  23.757714    42.95161275  44.50529175  56.88306925   0.
  61.13782525  12.56068882  20.89536025   0.          60.1904595
   0.           0.          43.19043425   0.           0.
 117.3768375   65.519863     0.          24.2527955   20.23909225
   0.          52.88594975   1.72341835   0.          39.903931
 117.4642125    0.         136.9223475  104.622189    57.2597515
  32.347913    10.71355365  45.46953775  35.9352925    0.
 113.8856025   32.347913    19.60477275  49.1453005   28.2083865
  26.324164    49.6471965    0.         157.068135     0.
   0.         102.2695485   59.57128025   1.94127595 104.07936975
 106.0937195  127.8874075    0.         122.9378925   36.34371925
  -4.27379614   0.           0.         109.63539275   0.
   5.963572   -41.0650515    0.          21.50457025   0.
   0.          32.973254   -15.74450275   6.57874722  34.31953325
  59.57128025  10.36060443  -9.2768644    0.          34.839483
   0.          69.34187775   5.01421383   0.          67.77000475
  26.197627     0.          17.94014925 132.2512425    0.
```

```
   0.         -19.40981325 154.098655    24.93186975   0.
  73.42673725 131.622885    36.7270825    0.          32.72866425
  56.1742595    9.0789077  112.8063545  133.7904775  147.411415
  53.439019     2.17719111  27.86148275 -14.58170133  48.8969475
   0.72484325   0.          94.211027    10.48317058   3.600121
  41.40336675 -11.07359075   2.17719111  22.39827475   0.
  34.92788775   0.          75.71514075  89.567969   110.979485
  -8.7290735   75.6223725    0.           0.           0.
 101.6526295    0.          86.5317875   39.166985    88.6605315
 111.5049475    0.         106.3435175  146.1350275    0.
 116.7058675    4.5704862  102.2695485  117.3768375   64.19510825
  89.71383975  22.5152235    0.          34.97302875  44.83542625
 -10.31073568   0.46691312  79.0509185  -14.60804218  36.48324675
  25.1794875    0.          35.60614025   0.         111.5049475
  81.78022775   0.         155.13313     34.97302875  45.57684925
  21.77430425   0.          67.47957625 -34.79555325 123.26996
  10.14592203  78.9403135    0.          41.2032585  -16.197954
  23.282967    11.01389913 151.429325   126.2322625   20.89536025
   0.          24.93435225  12.69640505  43.33966475  69.82616875
  31.09968525  14.69953025 -13.69537318 131.601745    46.2902115
   0.          83.515547    14.01185432  19.4202495   21.491069
  54.18786975   0.          61.13782525  64.492366     5.1117646
 -13.22029817  53.04044275  23.9227055   80.58941375  20.73910775
   0.           0.         101.8595845  132.4953075  151.429325
 136.1928775   92.939668    12.16613735 127.2691475    0.
  81.85451875  96.5276635   28.151358    27.94416975   8.35364008
  16.39819425   0.          89.74429675  65.5553395  105.79641625
  76.17377625  12.56068882 110.5049025  115.2187225    2.29638192
   0.         150.34947      0.          51.86458375   4.3189102
  79.0509185    0.          74.64539125   0.          45.65286675
   0.         125.79888      0.          77.91828325  35.13448625
   0.           0.           0.           0.         147.447925
   0.         -14.60804217  27.1111825   16.9740265    0.
   9.01278103   7.2548613   23.9376815   43.19043425   5.553989
   0.          72.545109   142.518895    20.89536025  36.94298225
  58.16879325   0.          11.95084065  -3.3382919   30.67802275
  37.93228025   0.          23.29668625 117.6353975  -17.836004
  51.980923     0.           8.8531945   57.90919025   0.
   0.         148.4401925  101.1519465   28.26058575  -7.5189555
 145.1551075    0.           0.           2.83363015  29.4244105
   0.         152.0255975   62.3673105  137.7909175   46.76988975
  21.5358705   84.30458175  88.0224945  126.0474675    0.
   0.          25.1794875   26.324164     0.          19.05861387
   0.           0.          25.4658125    0.          29.15845075
  87.3593605  157.2833875   59.0980785   20.00482475 105.44544025
```

```
 137.7909175   78.33384525  33.1321255    0.          121.8389
 -13.50847975   0.          65.5553395   35.72480325   0.
   0.           0.          62.483261   122.0742175   65.42895275
   0.          41.40336675  23.6494515    0.          10.88752325
  13.0381985    0.           0.          32.12378125  26.8740765
 -16.71533475   0.          52.34873275  74.3706035  -18.896867
  23.75227255   0.           0.           0.           0.
   0.          -8.7772454    2.30662682   0.          18.5170805
 123.26996     80.208951   142.518895   -14.60804218   0.
  33.86010025   0.          42.80022725 147.411415   134.1176125
  87.02657625  64.19510825   0.         -26.779211    72.6429525
   0.          44.330738    39.096541     0.         125.082345
  32.50757025 105.61616425   0.         100.7470235  109.3984125
 -10.31073568   0.         144.2620425   26.1677685   46.269951
 102.0033515  120.6546825    0.           0.          65.5553395
  36.609176     7.34841787 140.04343    106.6951245   19.518925  ]
```

(b). k=4, plot prediction and ground truth

```
pred = knn(mesh_std_x,train_x_std,train_y,k=4)

fig, ax = plt.subplots(subplot_kw={"projection": "3d"})

# Plot the surface.
surf = ax.plot_surface(xx1,xx2,pred.reshape(xx1.shape), cmap=cm.coolwarm,
                       linewidth=0, antialiased=False)
ax.scatter(test_x[:,0].reshape(-1,1),test_x[:,1].reshape(-1,1),test_y.res
hape(-1,1),label="ground truth")
ax.legend()
cbar =fig.colorbar(surf, shrink=0.5, aspect=5)
cbar.ax.set_ylabel('predition surface', rotation=270)
ax.set(xlabel='x1',ylabel="x2",zlabel="y")
plt.show()
```



(e) Note and report your qualitative observations about the differences between $K=1$ vs. $K=4$ regression.

From the two graph, we could notice that when $K=4$, the 3D plot is more smooth and seems natural, this is reasonable since it is using nearest 4 points and weights them to get the results. However, for the $K=1$ condition, only use one point will lead the prediction only has the out put of the train set, thus, the graph seems not smooth and natural than the $K=4$.