[Lab 4]

Name: Nguyễn Thành Trung – MSSV: 19522431

Link github: <https://github.com/Shu2301/Data_Mining/tree/main/Week4>

#Import necessary modules

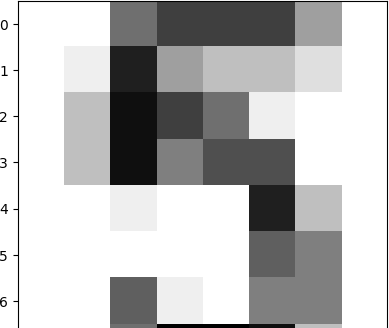
from sklearn import datasets

import matplotlib.pyplot as plt

#Load the digits dataset: digits digits = datasets.load\_digits()

#Display image 1010

plt.imshow(digits.images[1010], cmap = plt.cm.gray\_r, interpolation = 'nearest') plt.show()



from sklearn.model\_selection import train\_test\_split X = digits.data

Y= digits.target

#Split into training and test set

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size = 0.2, random\_state = 42, stratify = Y)

#Import necessary modules

from sklearn.neighbors import KNeighborsClassifier import numpy as np

#Create a k-NN classifier with 3 nightbors: knn knn = KNeighborsClassifier(n\_neighbors = 3)

#Fit the classifier to the training data knn.fit(X\_train, Y\_train)

#Print the accuracy

print("Accuracy: {0}".format(knn.score(X\_test, Y\_test)))

Accuracy: 0.9861111111111112

#Setup arrays to sore train nad test accuracies neighbors = np.arange(1, 9)

train\_accuracy = np.empty(len(neighbors)) test\_accuracy = np.empty(len(neighbors))

#Loop over different values of k for i, k in enumerate(neighbors):

#Setup a k-NN Classifier with k neighbors: knn knn = KNeighborsClassifier(n\_neighbors = k)

#FIt the classifier to the training data knn.fit(X\_train, Y\_train)

#Compute accuracy on the training set

train\_accuracy[i] = knn.score(X\_train, Y\_train)

#Compute accuracy on the training set

test\_accuracy[i] = knn.score(X\_test, Y\_test)

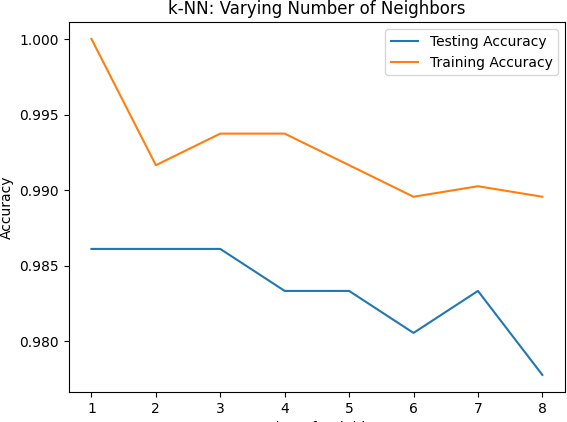
#Generate plot

plt.title('k-NN: Varying Number of Neighbors')

plt.plot(neighbors, test\_accuracy, label = 'Testing Accuracy') plt.plot(neighbors, train\_accuracy, label = 'Training Accuracy') plt.legend()

plt.xlabel('Number of Neighbors') plt.ylabel('Accuracy')

plt.show()



from future import print\_function import torch

import torch.nn as nn

import torch.nn.functional as P

from torch.autograd import Variable

from torchvision import datasets, transforms

mnist = datasets.MNIST(root = '.', train = True, download = True)

Downloading [h ttp://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz](http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz)

Downloading [h ttp://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz](http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz) to ./MNIST/raw/train-images-idx3-ubyte.gz

100%|██████████| 9912422/9912422 [00:00<00:00, 170547429.40it/s]Extracting ./MNIST/raw/train-images-idx3-ubyte.gz to ./MNIST/raw

Downloading [h ttp://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz](http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz)

Downloading [h ttp://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz](http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz) to ./MNIST/raw/train-labels-idx1-ubyte.gz 100%|██████████| 28881/28881 [00:00<00:00, 7437112.83it/s]

Extracting ./MNIST/raw/train-labels-idx1-ubyte.gz to ./MNIST/raw

Downloading [h ttp://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz](http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz)

Downloading [h ttp://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz](http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz) to ./MNIST/raw/t10k-images-idx3-ubyte.gz

100%|██████████| 1648877/1648877 [00:00<00:00, 65664261.95it/s]Extracting ./MNIST/raw/t10k-images-idx3-ubyte.gz to ./MNIST/raw

Downloading [h ttp://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz](http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz)

Downloading [h ttp://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz](http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz) to ./MNIST/raw/t10k-labels-idx1-ubyte.gz

100%|██████████| 4542/4542 [00:00<00:00, 13530205.09it/s]Extracting ./MNIST/raw/t10k-labels-idx1-ubyte.gz to ./MNIST/raw

print("Number of training examples", mnist.train\_data.shape) print("Image information", mnist[0])

/usr/local/lib/python3.9/dist-packages/torchvision/datasets/mnist.py:75: UserWarning: train\_data has been renamed data warnings.warn("train\_data has been renamed data")

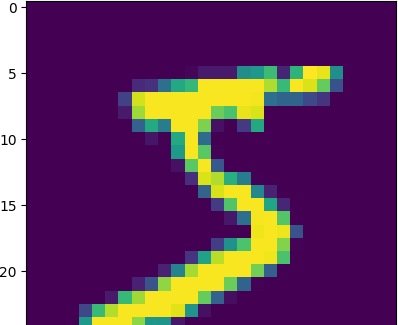
Number of training examples torch.Size([60000, 28, 28])

Image information (<PIL.Image.Image image mode=L size=28x28 at 0x7F334271F040>, 5)

import matplotlib.pyplot as plt

%matplotlib inline

plt.imshow(mnist[0][0])

<matplotlib.image.AxesImage at 0x7f333fee30d0>

class Net(nn.Module):

def init (self):

super(Net, self). init ()

self.fully = nn.Sequential( nn.Linear(28\*28, 10)

)

def forward(self, x):

x = x.view([-1, 28\*28]) x = self.fully(x)

x = F.log\_softmax(x, dim = 1) return x

train\_loader = torch.utils.data.DataLoader(datasets.MNIST(root = '.', train = True, transform = transforms.Compose([transforms.ToTensor()])), test\_loader = torch.utils.data.DataLoader(datasets.MNIST(root = '.', train = False, transform = transforms.Compose([transforms.ToTensor()])),

def train():

learning\_rate = 1e-3 num\_epochs = 3

net = Net()

optimizer = torch.optim.Adam(net.parameters(), lr = learning\_rate)

for epoch in range(num\_epochs):

for batch\_idx, (data, target) in enumerate(train\_loader): output = net(data)

loss = F.nll\_loss(output, target) optimizer.zero\_grad()

loss.backward() optimizer.step()

if batch\_idx % 100 == 0:

print('Epoch = %f. Batch = %s. Loss' % (epoch,batch\_idx, loss.item())) return net

net = train()

net.eval()

test\_loss = 0

correct = 0

total = 0

for data, target in test\_loader:

total += len(target) output = net(data)

pred = output.max(1, keepdim = True)[1]

correct += target.eq(pred.view\_as(target)).sum()

print("Correct out of %s" % total, correct.item())

print("Percentage accuracy", correct.item()\*100/10000.)

#Import numpy and pandas import numpy as np

import pandas as pd

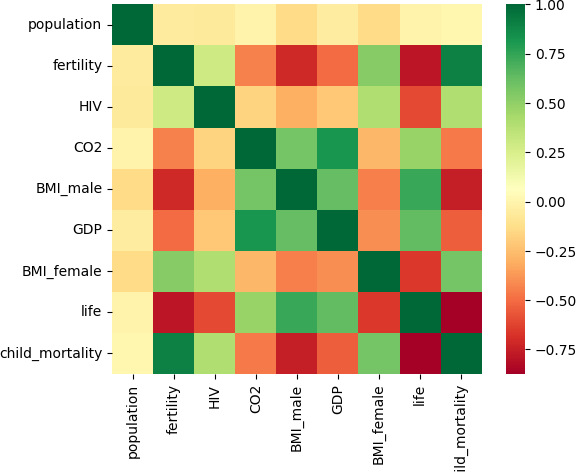
import seaborn as sns

import matplotlib.pyplot as plot

#Read the CSV file into a DataFrame: df df = pd.read\_csv('gapminder.csv')

ax = sns.heatmap(df.corr(), square = True, cmap = 'RdYlGn') plt.show()

<ipython-input-16-3cf57014a7aa>:1: FutureWarning: The default val ax = sns.heatmap(df.corr(), square = True, cmap = 'RdYlGn')



 

from sklearn.linear\_model import LinearRegression, LogisticRegression from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error

from sklearn.model\_selection import train\_test\_split

x\_fertility = df['fertility'].values.reshape(-1,1) y\_life = df['life'].values.reshape(-1,1)

prediction\_space = np.linspace(min(x\_fertility), max(x\_fertility)).reshape(-1,1)

#Create training and test sets

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x\_fertility, y\_life, test\_size = 0.3, random\_state = 42)

#Create the regression model: reg\_all reg = LinearRegression()

#Fit the regression to the training data reg.fit(x\_train, y\_train)

y\_predict = reg.predict(prediction\_space)

#Print accuracy

print(reg.score(x\_fertility, y\_life))

#Plot regression Line

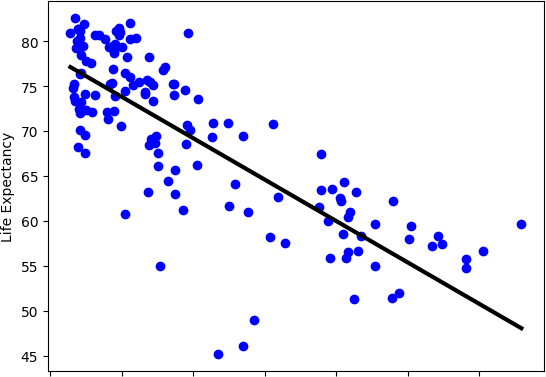
plt.scatter(x\_fertility, y\_life, color = 'blue')

plt.plot(prediction space, y predict, color = 'black', linewidth = 3)

plt.ylabel('Life Expectancy') plt.xlabel('Fertility ')

plt.show

 0.6162438752151917

<function matplotlib.pyplot.show(close=None, block=None)>

features = pd.read\_csv('gapminder.csv') df = pd.read\_csv('gapminder.csv')

del features['life'] del features['Region']

y\_life = df['life'].values.reshape(-1,1) #Create training and test sets

x\_train, x\_test, y\_train, y\_test = train\_test\_split(features, y\_life, test\_size = 0.3, random\_state = 42)

#Create the regression model: reg\_all reg\_all = LinearRegression()

#Fit the regression to the training data reg\_all.fit(x\_train, y\_train)

#Print accuracy

print(reg\_all.score(features, y\_life)) 0.8914651485793176

import matplotlib.pyplot as plt

%matplotlib inline import numpy as np

N = 10 #Number of data points m = .9

c = 1

x = np.linspace(0,2\*np.pi, N)

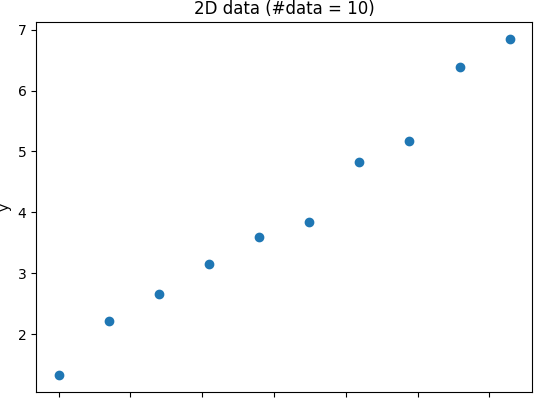
y = m\*x + c + np.random.normal(0, .3, x.shape) plt.figure()

plt.plot(x, y, 'o')

plt.xlabel('x')

plt.ylabel('y')

plt.title('2D data (#data = %d)' % N) plt.show()



import torch

from torch.utils.data import Dataset

class MyDataset(Dataset):

def init (self, x, y):

self.x = x self.y = y

def len (self):

return len(self.x)

def getitem (self, idx):

sample = {

'feature': torch.tensor([1, self.x[idx]]),

'label': torch.tensor([self.y[idx]])} return sample

dataset = MyDataset(x, y)

for i in range(len(dataset)):

sample = dataset[i]

print(i, sample['feature'], sample['label'])

1. tensor([1., 0.], dtype=torch.float64) tensor([1.3291], dtype=torch.float64)
2. tensor([1.0000, 0.6981], dtype=torch.float64) tensor([2.2222], dtype=torch.float64)
3. tensor([1.0000, 1.3963], dtype=torch.float64) tensor([2.6650], dtype=torch.float64)
4. tensor([1.0000, 2.0944], dtype=torch.float64) tensor([3.1505], dtype=torch.float64)
5. tensor([1.0000, 2.7925], dtype=torch.float64) tensor([3.5954], dtype=torch.float64)
6. tensor([1.0000, 3.4907], dtype=torch.float64) tensor([3.8335], dtype=torch.float64)
7. tensor([1.0000, 4.1888], dtype=torch.float64) tensor([4.8242], dtype=torch.float64)
8. tensor([1.0000, 4.8869], dtype=torch.float64) tensor([5.1775], dtype=torch.float64)
9. tensor([1.0000, 5.5851], dtype=torch.float64) tensor([6.3916], dtype=torch.float64)
10. tensor([1.0000, 6.2832], dtype=torch.float64) tensor([6.8490], dtype=torch.float64)

from torch.utils.data import DataLoader dataset = MyDataset(x, y)

batch\_size = 4 shuffle = True num\_workers = 4

dataloader = DataLoader(dataset, batch\_size = batch\_size, shuffle = shuffle, num\_workers = num\_workers)

/usr/local/lib/python3.9/dist-packages/torch/utils/data/dataloader.py:561: UserWarning: This DataLoader will create 4 worker processes warnings.warn(\_create\_warning\_msg(

 

import pprint as pp

for i\_batch, samples in enumerate(dataloader):

print('\nbatch# = %s' % i\_batch) print('samples: ')

pp.pprint(samples)

/usr/local/lib/python3.9/dist-packages/torch/utils/data/dataloader.py:561: UserWarning: This DataLoader will create 4 worker processes warnings.warn(\_create\_warning\_msg(

batch# = 0 samples:

{'feature': tensor([[1.0000, 2.0944],

[1.0000, 5.5851],

[1.0000, 6.2832],

[1.0000, 4.1888]], dtype=torch.float64), 'label': tensor([[3.1505],

[6.3916],

[6.8490],

[4.8242]], dtype=torch.float64)}

batch# = 1 samples:

{'feature': tensor([[1.0000, 3.4907],

[1.0000, 1.3963],

[1.0000, 0.6981],

[1.0000, 0.0000]], dtype=torch.float64), 'label': tensor([[3.8335],

[2.6650],

[2.2222],

[1.3291]], dtype=torch.float64)}

batch# = 2 samples:

{'feature': tensor([[1.0000, 4.8869],

[1.0000, 2.7925]], dtype=torch.float64), 'label': tensor([[5.1775],

[3.5954]], dtype=torch.float64)}

import torch.nn as nn

import torch.nn.functional as F class MyModel(nn.Module):

def init (self,input\_dim, output\_dim):

super(MyModel, sel). init ()

self.linear = nn.Linear(input\_dim, output\_dim)

def forward(self, x):

 

out = self.linear(x) return out

input\_dim = 2

output\_dim = 1

model = MyModel(input\_dim, output\_dim)

cost = nn.MSELoss()

num\_epochs = 10

l\_rate = 0.01

optimiser = torch.optim.SGD(model.parameters(), lr = l\_rate)

dataset = MyDataset(x, y) batch\_size = 4

shuffle = True num\_workers = 4

training\_sample\_generator = DataLoader(dateset, batch\_size=batch\_size, shuffle=suffle, num\_workers=num\_workers) for epoch in range(num\_epochs):

print ('Epoch = %s' % epoch)

for batch\_i, samples in enumerate(training\_sample\_generator): predictions = model(samples['feature'])

error = cost(predictions, samples['label'])

print('\tBatch = %s, Error = %s' % (batch\_i, error.item()))

optimiser.zero\_grad() error.backward()

optimiser.step()

x\_for\_plotting = np.linspace(0, 2\*np.pi, 1000)

design\_matrix = torch.tensor(np.vstack([np.ones(x\_for\_plotting.shape), x\_for\_plotting]).T, dtype=torch.float32) print('Design matrix shape:', design\_matrix.shape)

y\_for\_pltting = model.forward(design\_matrix)

print('y\_for\_plotting:', y\_for\_plotting.shape)

y\_for\_plotting = model.forward(design\_matrix)

print('y\_for\_plotting shape:', y\_for\_plotting.shape)

plt.figure()

plt.plot(x,y,'o')

plt.plot(x\_for\_plottingm y\_for\_plotting.data.numpy(), 'r-')

plt.xlabel('x')



plt.ylabel('y')

plt.title('2D data (#data = %d)' % N) plt.show()

def item\_cf(M, metric='cosine'):

pred = np.copy(m)

n\_users, n\_items = M.shape

avg\_ratings = np.nanmean(M, axis=0)

sim\_items = sim\_matrix(M, 'item', metric) for i in range(n\_users):

for j in range(n\_items):

if np.isnan(M[i,j]):

pred[i,j] = avg\_ratings[j] + np.nansum(sim\_items[j] \* (M[i.:] - avg\_ratings)) / sum(sim\_items[j]) return pred

evaluateRS(M, M\_result, 'user\_cf', 'cosine')

evaluateRS(M, M\_result, 'user\_cf', 'correlation') evaluateRS(M, M\_result, 'item\_cf', 'cosin')

evaluateRS(M, M\_result, 'item\_cf', 'correlation')