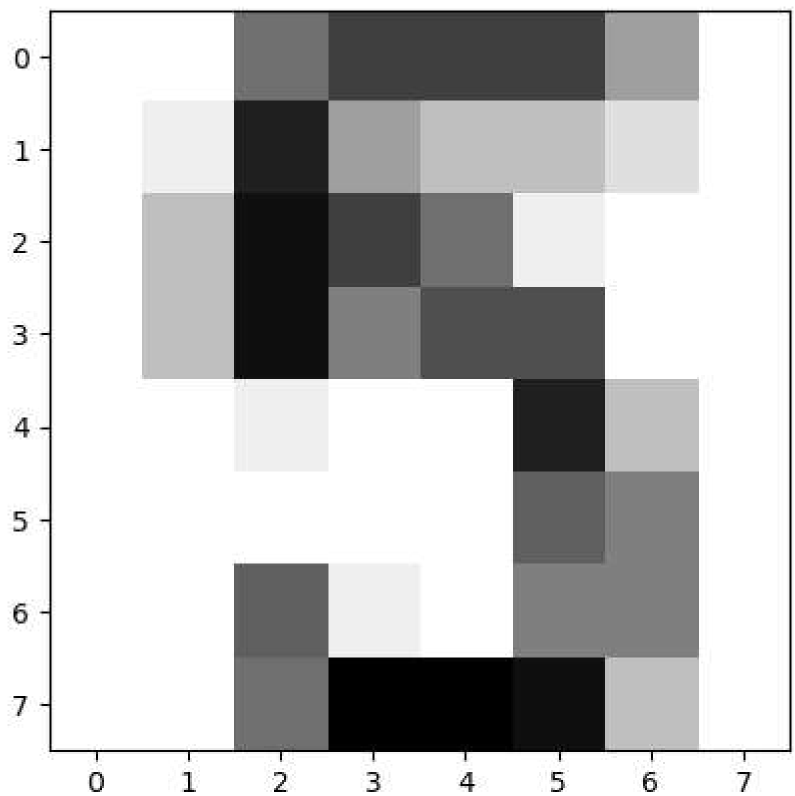
ID: 18521272

Name: Le Ngoc Thai Phuong

I. Classification

In [1]:
```
from sklearn import datasets
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

digits = datasets.load_digits()

# Display digit 1010
plt.imshow(digits.images[1010], cmap=plt.cm.gray_r, interpolation='neares
plt.show()
```



In [2]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split

# Create feature and target arrays
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)

# Create a k-NN classifier with 7 neighbors: knn
knn = KNeighborsClassifier(n_neighbors=7)

# Fit the classifier to the training data
knn.fit(X_train, y_train)

# Print the accuracy
print(knn.score(X_test, y_test))
```

```
0.9833333333333333

C:\Users\ACER\anaconda3\lib\site-packages\sklearn\neighbors\_classification.p
y:228: FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosi
s`), the default behavior of `mode` typically preserves the axis it acts alon
g. In SciPy 1.11.0, this behavior will change: the default value of `keepdims
` will become False, the `axis` over which the statistic is taken will be eli
minated, and the value None will no longer be accepted. Set `keepdims` to Tru
e or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

In [3]:

```
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 testing 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()
```

```
C:\Users\ACER\anaconda3\lib\site-packages\sklearn\neighbors\_classificatio
n.py:228: FutureWarning: Unlike other reduction functions (e.g. `skew`, `k
urtosis`), the default behavior of `mode` typically preserves the axis it
acts along. In SciPy 1.11.0, this behavior will change: the default value
of `keepdims` will become False, the `axis` over which the statistic is ta
ken will be eliminated, and the value None will no longer be accepted. Set
`keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\Users\ACER\anaconda3\lib\site-packages\sklearn\neighbors\_classificatio
n.py:228: FutureWarning: Unlike other reduction functions (e.g. `skew`, `k
urtosis`), the default behavior of `mode` typically preserves the axis it
acts along. In SciPy 1.11.0, this behavior will change: the default value
of `keepdims` will become False, the `axis` over which the statistic is ta
ken will be eliminated, and the value None will no longer be accepted. Set
`keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
C:\Users\ACER\anaconda3\lib\site-packages\sklearn\neighbors\_classificatio
n.py:228: FutureWarning: Unlike other reduction functions (e.g. `skew`, `k
urtosis`), the default behavior of `mode` typically preserves the axis it
acts along. In SciPy 1.11.0, this behavior will change: the default value
```

In [4]:

```
from __future__ import print_function
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.autograd import Variable
```

In [ ]:

```

```

In [5]:

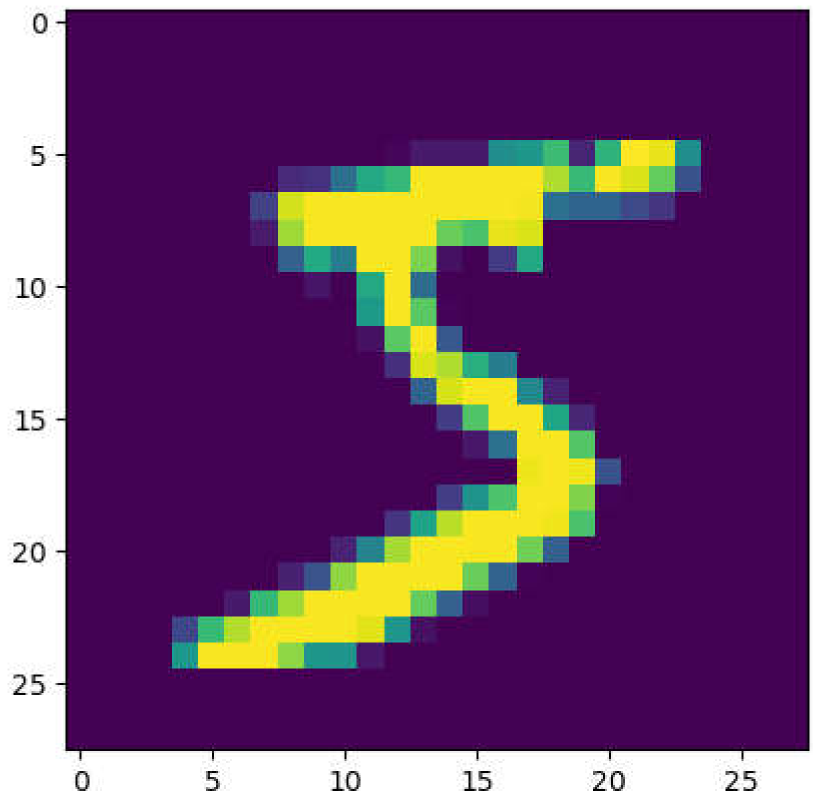
```
from torchvision import datasets, transforms
mnist = datasets.MNIST(root='', train=True, download=True)

print("Number of training example: ", mnist.train_data.shape)
print("Image information ", mnist[0])
```

```
Number of training example:  torch.Size([60000, 28, 28])
Image information  (<PIL.Image.Image image mode=L size=28x28 at 0x1D1426948B0
>, 5)

C:\Users\ACER\anaconda3\lib\site-packages\torchvision\datasets\mnist.py:75: U
serWarning: train_data has been renamed data
  warnings.warn("train_data has been renamed data")
```

In [6]:

```
import matplotlib.pyplot  as plt
%matplotlib inline
plt.imshow(mnist[0][0])
```

Out[6]: <matplotlib.image.AxesImage at 0x1d14a23f730>



In [7]:

```
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
```

In [8]:

```
train_loader = torch.utils.data.DataLoader(datasets.MNIST(root=".", train=
test_loader = torch.utils.data.DataLoader(datasets.MNIST(root=".", train=F

```

In [9]:

```
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 = %s' % (epoch, batch_

    return net
```

In [10]:
```
net = train()
```

```
Epoch = 0.000000. Batch = 0. Loss = 2.339271306991577
Epoch = 0.000000. Batch = 100. Loss = 0.80986487865448
Epoch = 0.000000. Batch = 200. Loss = 0.6029140949249268
Epoch = 0.000000. Batch = 300. Loss = 0.5262631177902222
Epoch = 0.000000. Batch = 400. Loss = 0.5180872082710266
Epoch = 0.000000. Batch = 500. Loss = 0.49103665351867676
Epoch = 0.000000. Batch = 600. Loss = 0.2785436511039734
Epoch = 0.000000. Batch = 700. Loss = 0.5144723653793335
Epoch = 0.000000. Batch = 800. Loss = 0.2813175618648529
Epoch = 0.000000. Batch = 900. Loss = 0.29325219988822937
Epoch = 1.000000. Batch = 0. Loss = 0.2884300947189331
Epoch = 1.000000. Batch = 100. Loss = 0.2784908413887024
Epoch = 1.000000. Batch = 200. Loss = 0.45401087403297424
Epoch = 1.000000. Batch = 300. Loss = 0.24643376469612122
Epoch = 1.000000. Batch = 400. Loss = 0.35517826676368713
Epoch = 1.000000. Batch = 500. Loss = 0.2768622040748596
Epoch = 1.000000. Batch = 600. Loss = 0.3907529413700104
Epoch = 1.000000. Batch = 700. Loss = 0.3144478499889374
Epoch = 1.000000. Batch = 800. Loss = 0.22636058926582336
Epoch = 1.000000. Batch = 900. Loss = 0.25695928931236267
Epoch = 2.000000. Batch = 0. Loss = 0.12280141562223434
Epoch = 2.000000. Batch = 100. Loss = 0.5356943607330322
Epoch = 2.000000. Batch = 200. Loss = 0.2567916214466095
Epoch = 2.000000. Batch = 300. Loss = 0.3874041736125946
Epoch = 2.000000. Batch = 400. Loss = 0.33864039182662964
Epoch = 2.000000. Batch = 500. Loss = 0.405193954706192
Epoch = 2.000000. Batch = 600. Loss = 0.1978750377893448
Epoch = 2.000000. Batch = 700. Loss = 0.2189231514930725
Epoch = 2.000000. Batch = 800. Loss = 0.5021193027496338
Epoch = 2.000000. Batch = 900. Loss = 0.3062554895877838
```

In [11]:
```
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.)
```

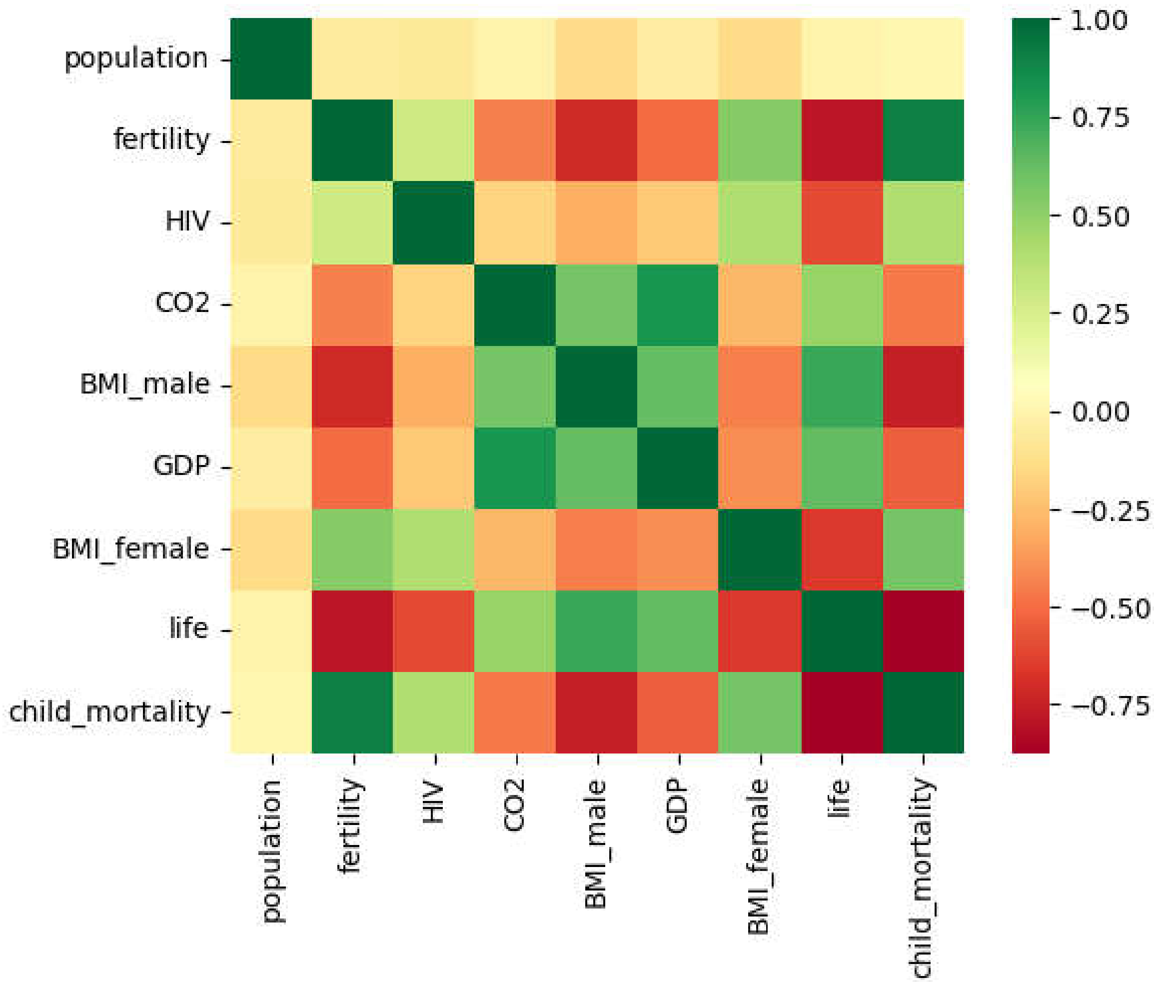
```
Correct out of 10000 9228
Percentage accuracy 92.28
```

II. Linear Regression

In [12]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv('gapminder.csv')
```

In [13]:
```
sns.heatmap(df.corr(), square=True, cmap='RdYlGn')
```

Out[13]: <AxesSubplot:>

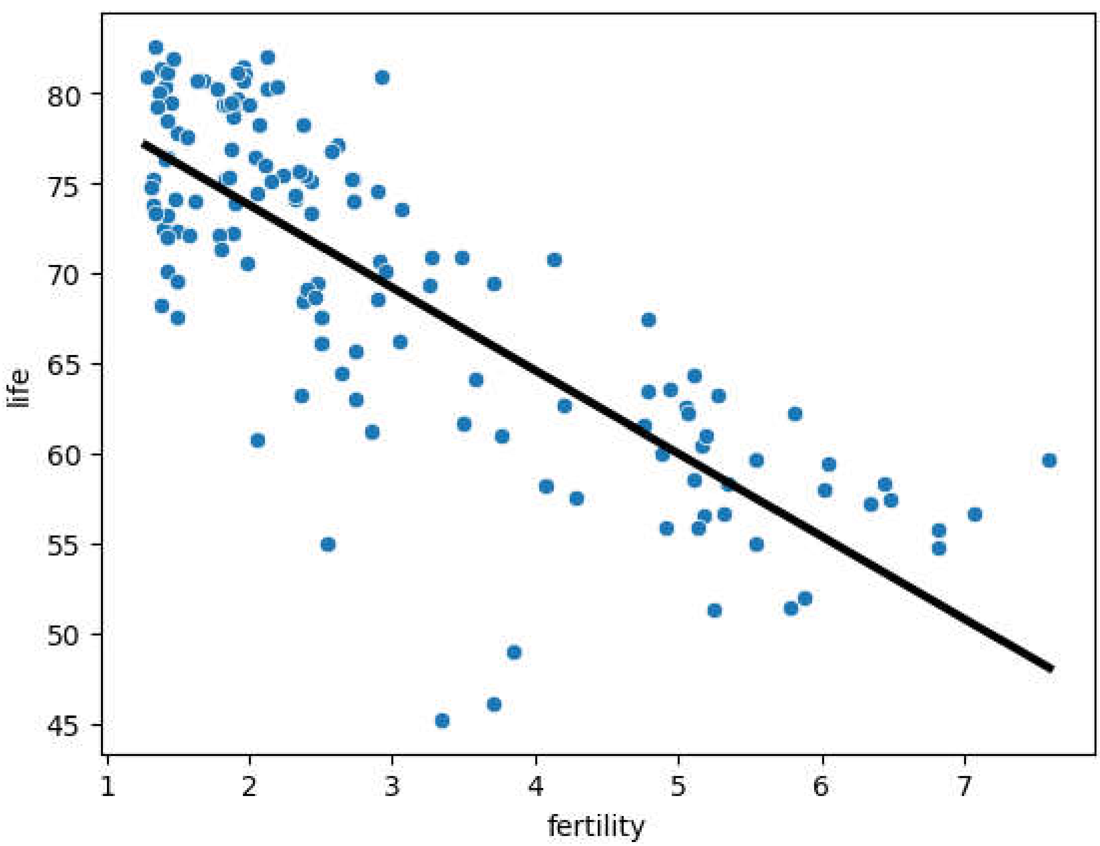


In [14]:

```
from sklearn.linear_model import LinearRegression

# Create the regressor: reg
reg = LinearRegression()

X_fertility = df['fertility'].values.reshape(-1, 1)
y = df['life'].values.reshape(-1, 1)

X_train, X_test, y_train, y_test = train_test_split(X_fertility, y, test_s

# Create th prediction space
prediction_space = np.linspace(min(X_fertility), max(X_fertility)).reshape

# Fit the model to the data
reg.fit(X_train, y_train)

# compute predictions over the prediction space: y_pred
y_pred = reg.predict(prediction_space)

# Print $R^2$
print(reg.score(X_fertility, y))

# Plot regression line on scatter plot
sns.scatterplot(x='fertility', y='life', data=df)
plt.plot(prediction_space, y_pred, color='black', linewidth=3)
```

```
0.6162438752151919
```

Out[14]: [<matplotlib.lines.Line2D at 0x1d14a2af730>]



In [15]:
```
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)
x_train, x_test, y_train, y_test = train_test_split(features, y_life, tes

reg_all = LinearRegression()
reg_all.fit(x_train, y_train)
print(reg_all.score(features, y_life))
```

```
0.8914651485793137
```

In [ ]:

In [ ]:

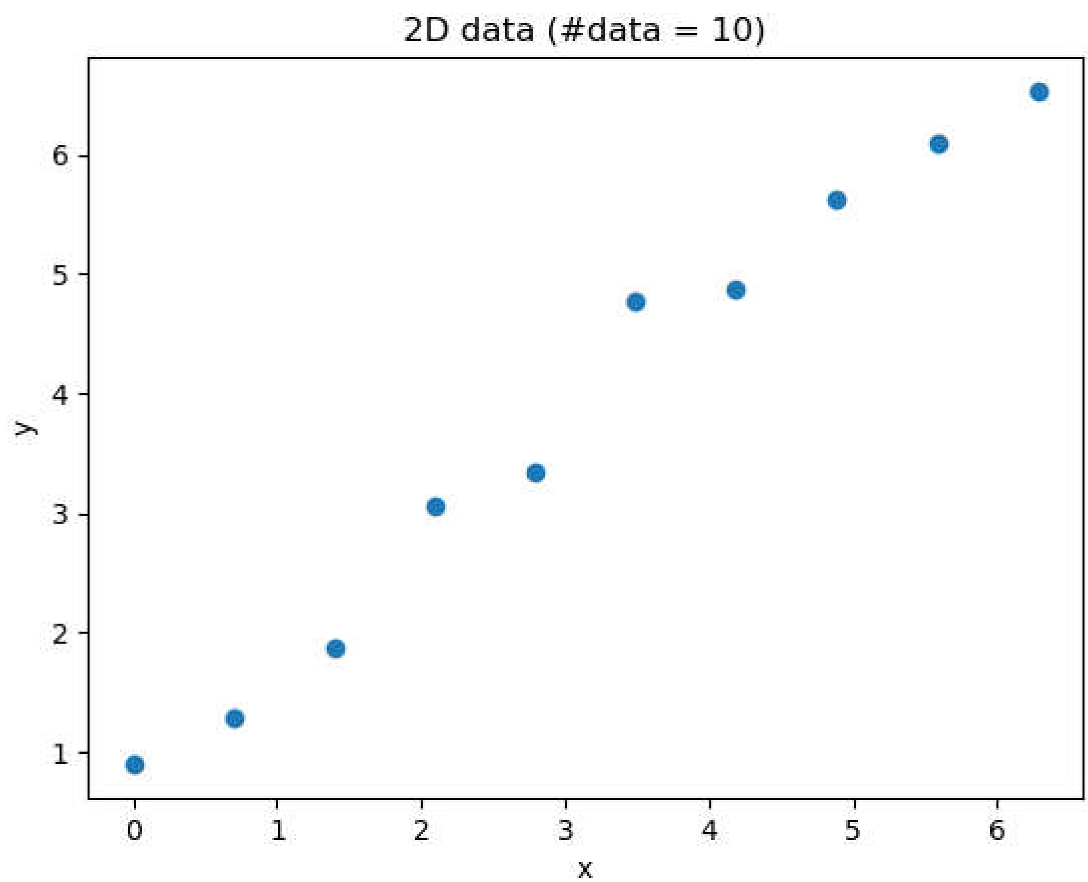
In [ ]:

In [ ]:

Linear Regression using PyTorch

In [16]:
```
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
```

In [17]:
```
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()
```



In [18]:
```
import torch
```

Dataset

In [19]:

```
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
```

In [20]:

```
dataset = MyDataset(x, y)
for i in range(len(dataset)):
    sample = dataset[i]
    print(i, sample['feature'], sample['label'])
```

```
0 tensor([1., 0.], dtype=torch.float64) tensor([0.9039], dtype=torch.float64)
1 tensor([1.0000, 0.6981], dtype=torch.float64) tensor([1.2929], dtype=torch.
float64)
2 tensor([1.0000, 1.3963], dtype=torch.float64) tensor([1.8805], dtype=torch.
float64)
3 tensor([1.0000, 2.0944], dtype=torch.float64) tensor([3.0595], dtype=torch.
float64)
4 tensor([1.0000, 2.7925], dtype=torch.float64) tensor([3.3478], dtype=torch.
float64)
5 tensor([1.0000, 3.4907], dtype=torch.float64) tensor([4.7624], dtype=torch.
float64)
6 tensor([1.0000, 4.1888], dtype=torch.float64) tensor([4.8760], dtype=torch.
float64)
7 tensor([1.0000, 4.8869], dtype=torch.float64) tensor([5.6290], dtype=torch.
float64)
8 tensor([1.0000, 5.5851], dtype=torch.float64) tensor([6.0992], dtype=torch.
float64)
9 tensor([1.0000, 6.2832], dtype=torch.float64) tensor([6.5297], dtype=torch.
float64)
```

Dataloader

In [29]:
```
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, 
```

In [ ]:
```
import pprint as pp
for i_batch, samples in enumerate(dataloader):
    print('\nbatch# = %s' % i_batch)
    print('samples: ')
    pp.pprint(samples)
```

Model

In [23]:
```
import torch.nn as nn
import torch.nn.functional as F
class MyModel(nn.Module):
    def __init__(self, input_dim, output_dim):
        super(MyModel, self).__init__()
        self.linear = nn.Linear(input_dim, output_dim)

    def forward(self, x):
        out = self.linear(x)
        return out
```

Setting a model for our problem

In [24]:
```
input_dim = 2
output_dim = 1

model = MyModel(input_dim, output_dim)
```

Cost function

In [25]:
```
cost = nn.MSELoss()
```

Minimizing the cost function

In [ ]:

```
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(dataset, batch_size=batch_size, sh

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()
```

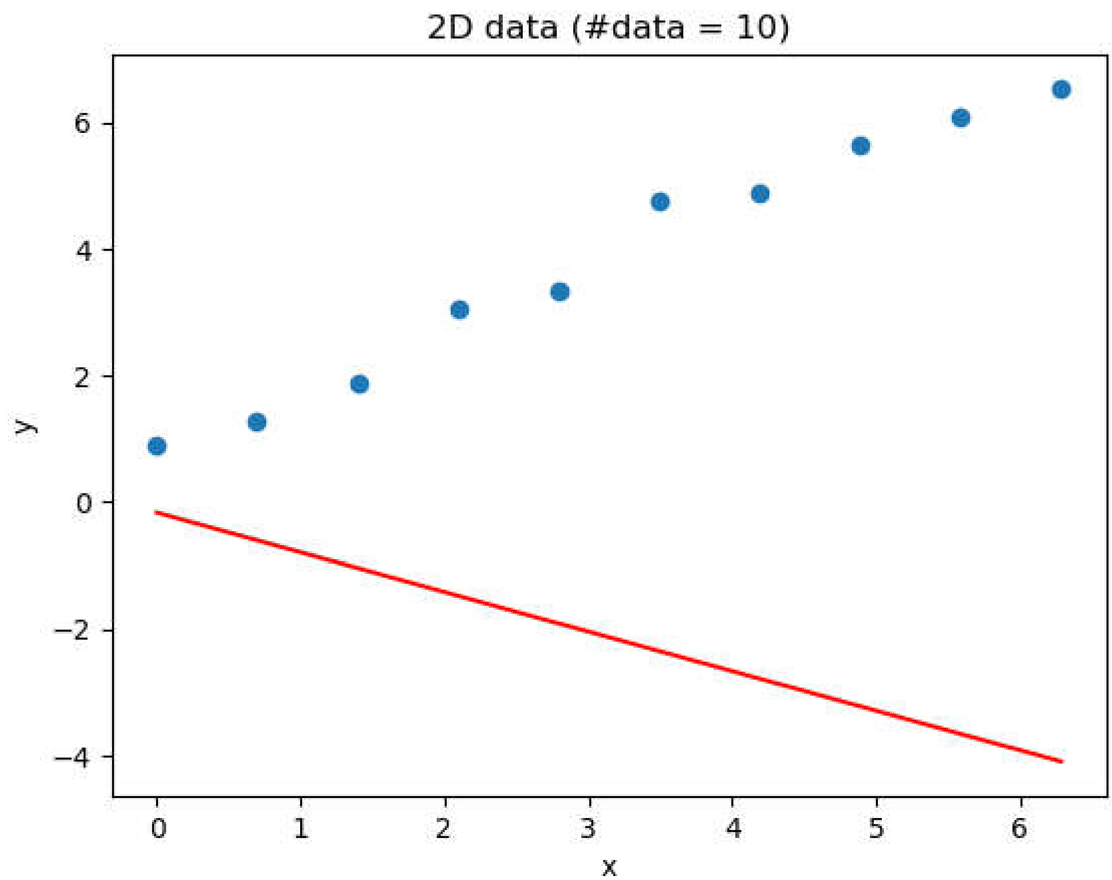
Lets see how well the model has learnt the data

In [27]:

```
x_for_plotting = np.linspace(0, 2*np.pi, 1000)
design_matrix = torch.tensor(np.vstack([np.ones(x_for_plotting.shape), x_
print('Design matrix shape:', design_matrix.shape)

y_for_plotting = model.forward(design_matrix)
print('y_for_plotting shape:', y_for_plotting.shape)
```

```
Design matrix shape: torch.Size([1000, 2])
y_for_plotting shape: torch.Size([1000, 1])
```

In [28]:
```
plt.figure()
plt.plot(x,y,'o')
plt.plot(x_for_plotting, y_for_plotting.data.numpy(), 'r-')
plt.xlabel('x')
plt.ylabel('y')
plt.title('2D data (#data = %d)' % N)
plt.show()
```



III. Recommendation Systems

In [ ]:
```

```

In [ ]:
```

```