1 Introduction

In a unit square, we can generate data points for class $+1$, data points inside a circle, and data points for class -1, data points outside the circle. The circle, inside the unit square, captures a given proportion of the total area of the unit square. This proportion has an upper bound. Given a proportion, the radius of the circle is a function of the area of the square. Figure 1 encloses the area captured by the two classes.

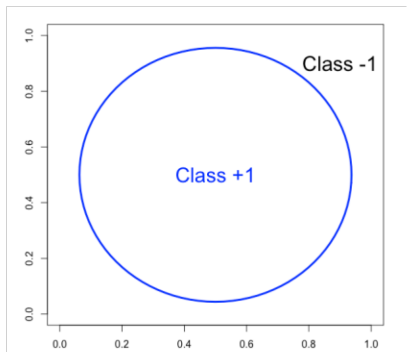

Figure 1: Generated Data for Class $+1$ and Class -1

This classification problem can be extended in two ways, that is, by:

1. Considering an arbitrary length l of the square instead of the unit square.
2. Considering an n-dimensional analogue of a square and a circle.

The first extension is not so interesting. In fact, it can be shown that the predicted class is independent of the length of an n-dimensional analogue of a square for all n. We will be exploring the second extension in this project.

2 The High Dimensional Sphere and Cube

2.1 The High Dimensional Sphere (Hypersphere)

An n-sphere, also called a hypersphere, is a set of points in an $(n+1)-$dimensional euclidean space that are located at a point, called the center. This is the generalization of an ordinary sphere in a three dimensional euclidean space. The radius is the constant distant from its points to the center. The n-sphere is embedded in an $n+1$ dimensional euclidean space and is related to an $n-$ball. Let us consider some examples.

- A $0-$sphere is a line segment, embedded in a $1-$dimensional euclidean space.
- A $1-$sphere is a circle, embedded in a $2-$dimensional euclidean space.
- A $2-$sphere is an ordinary sphere, embedded in a $3-$dimensional euclidean space, etc

The set of points in an $(n+1)-$space, $x_1, \ldots, x_{n+1}$, that defines an n-sphere, $S^n(r)$ is represented by the equation:

$$r^2 = \sum_{i=1}^{n+1} (x_i - d_i)^2 \tag{1}$$

where $d_1, \ldots, d_{n+1}$ is the center point and r is the radius. An $(n+1)-$ball is the spaced enclosed by an $n-$sphere. A $1-$ball, a line, is the interior of a $0-$sphere; A $2-$ball, a disk, is the interior of a $1-$sphere; A $3-$ball, an ordinary sphere, is the interior of a $2-$sphere; etc. The set of points satisfying the inequality,

$$\left\{ \sum_{i=1}^{n} (x_i - d_i)^2 < r^2 \right\}, \tag{2}$$

defines an $n-$ball.

2.1.1 Volume of an $n-$ ball

The volume of an $n-$ball is:

$$V_n(r) = \frac{\pi^{\frac{n}{2}}}{\Gamma(\frac{n}{2}+1)} r^n = \frac{\pi^{\frac{n}{2}}}{(\frac{n}{2})!} r^n \tag{3}$$

The volume is maximal when $n = 5$. It decreases and tends to zero as n gets large.

2.2 The High Dimensional Cube (Hypercube)

A hypercube is an $n-$dimensional analogue of a cube ($n = 3$). It is closed, compact and convex figure. This is usually called an $n-$cube. The name of each hypercube, for $n \in \{1, \dots, 8\}$, is summarised in the table below.

n	1	2	3	4	5	6	7	8
n-cube	1-cube	2-cube	3-cube	4-cube	5-cube	6-cube	7-cube	8-cube
name	line segment	square	cube	tesseract	penteract	hexeract	hepteract	octeract

2.2.1 Volume of an n-cube

The volume, $V_n(l)$, of an $n-$cube with length l is l^n. This is an increasing function of n for $l > 1$ and a constant for $l = 1$.

2.3 The Maximum Proportion of Volume

We are interested in generating data for two classes: class +1, data points inside an $n-$ball and class -1, data points outside an $n - ball$. Note that the $n - ball$ captures a certain proportion, p, of the volume of the $n-$ cube. This proportion, which will be explored, is a function of n.

Theorem 1 *Given an $n - ball$ and an $n - cube$ with radius r and length l respectively, the maximum proportion, $p(n)$, is a decreasing function of n which tends to zero in the limit.*

Proof

Let $V_n(r)$ be the volume of an $n - ball$, $V_n(l)$ the volume of an $n - cube$ and $p = p(n)$ the maximum proportion of the volume of an $n - cube$ that can be captured. The maximum proportion is attained when $l = 2r$.

$$
\begin{aligned}
V_n(r) &= \frac{\pi^{\frac{n}{2}}}{\Gamma(\frac{n}{2}+1)} r^n = pV_n(l) = pl^n \\
\implies p(n) &= \frac{\pi^{\frac{n}{2}}}{\Gamma(\frac{n}{2}+1)} \left(\frac{r}{l}\right)^n = \frac{\pi^{\frac{n}{2}}}{\Gamma(\frac{n}{2}+1)} \left(\frac{r}{2r}\right)^n \\
\implies p(n) &= \frac{(\sqrt{\pi})^n}{\Gamma(\frac{n}{2}+1)2^n} \leq \frac{1}{\Gamma(\frac{n}{2}+1)}
\end{aligned}
$$

Clearly, this is a decreasing function of n and approaches zero in the limit. Also, $p(n)$ does not depend on either l or r.

3 Data Generation

Without loss of generality, we will assume that $l = 1$. We can conclude from section (2) that the result holds for any arbitrary $l \geq 1$. Also, we will assume that the $n-ball$ is centered at $\underbrace{(0, \ldots, 0)}_{\text{n times}}$. Given the proportion, $p^\star \leq p$, of the volume of the $n-cube$ to be captured, the radius of the $n-$ball is $r = \left(\frac{p^\star l^n \Gamma(0.5n+1)}{(\sqrt{\pi})^n}\right)^{\frac{1}{n}}$. This reduces to $r = \left(\frac{p^\star \Gamma(0.5n+1)}{(\sqrt{\pi})^n}\right)^{\frac{1}{n}}$ for $l = 1$.

Steps for generating the data

- Generate random data points,$(x_1, \ldots, x_n)$, from $\textbf{Unif}([-1/2, 1/2]) \ldots \textbf{Unif}([-1/2, 1/2])$.
- If $\sum_{i=1}^{n} x_i^2 < r^2$, then class +1.
- Else if $\sum_{i=1}^{n} x_i^2 \geq r^2$, the class -1.

3.1 The training data

In the unit $n - ball$, we will generate 500 data points for class +1 and 500 data points for class -1 from the data distribution (See figure 2).

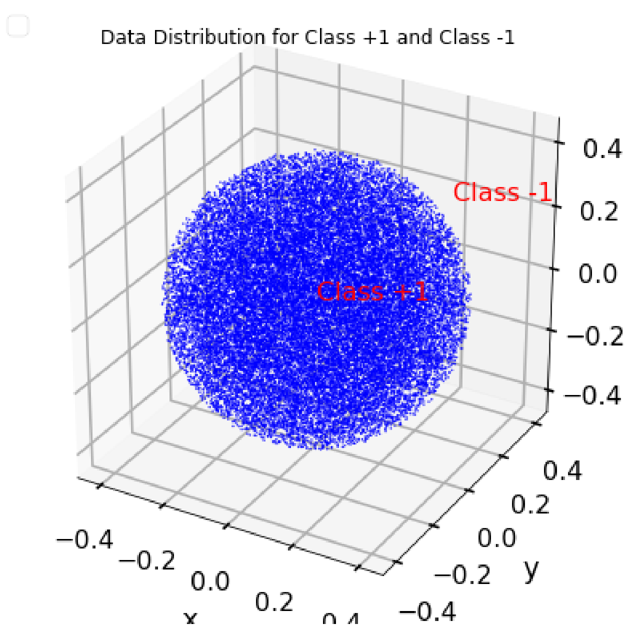

Figure 2: Data distribution for a unit sphere.

3.2 The test Data

The test data will be generated using a discrete grid pixelation. The test data is displayed in figure 3.

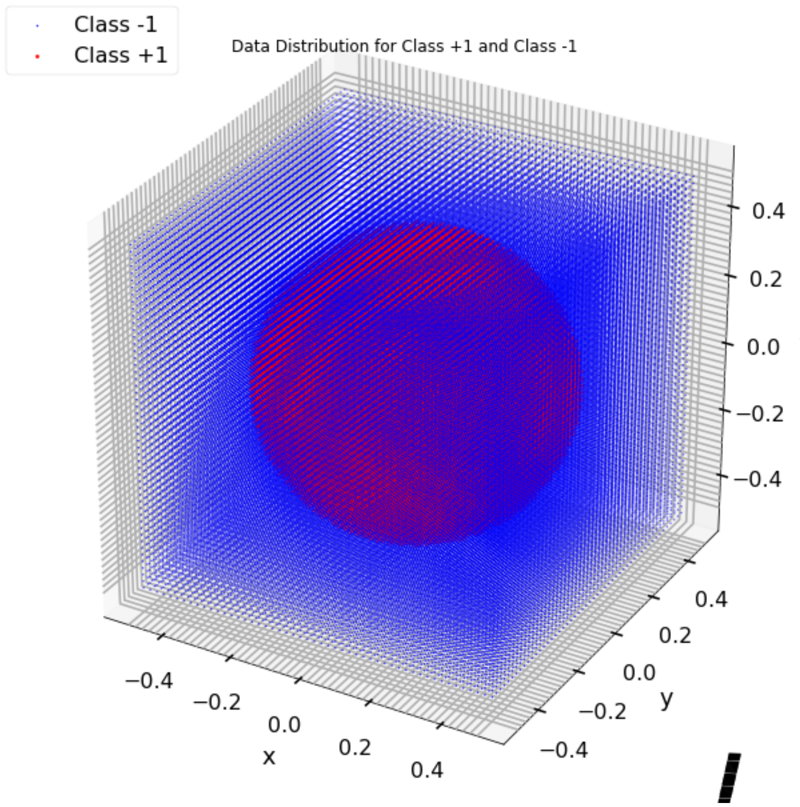

Figure 3: Discrete grid Pixelation Data.

The grid has an increment of 0.02. The test data has $(\frac{1}{0.02} + 1)^3 = 132,651$ observations.

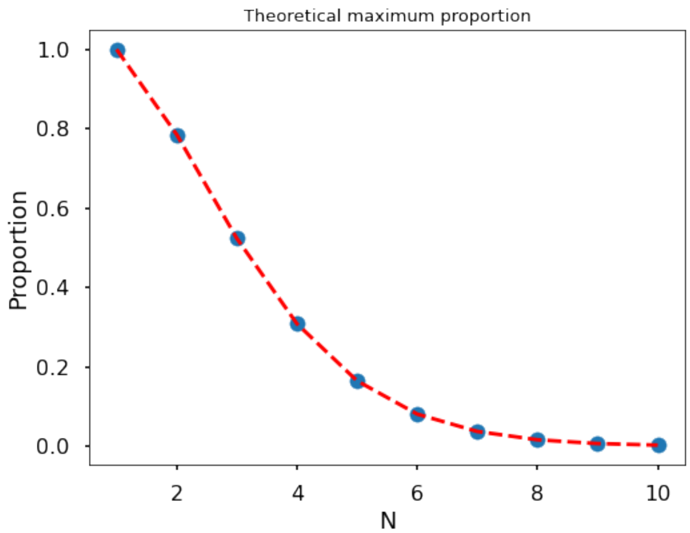

Figure 4: Maximum proportion of volume that can be captured

It can be seen from figure 4 that the maximum proportion of volume that can be captured is a decreasing function of n. This maximum proportion is approximately 0 for $n \geq 8$.

4 Model Training and Evaluation

4.1 Statistical learning models

The following models will be fitted to the training data for $n = 3$, a 3-cube and a 3-ball, and evaluated on the test data.

- Neural Network with back propagation.
- K-Nearest Neighbor
- Adaboost with a custom defined weak classifier

4.1.1 The Weak Classifier

A classifier for classifying a 3-cube and a 3-ball is $ax^2 + by^2 + cz^2 = d$. It can be calculated by picking (x, y, z) randomly from the training data with $-0.5 \leq a, b, c \leq 0.5$. This classifier classifies a data point (x_0, y_0, z_0) via the following criterion

$$(x_0, y_0, z_0) \in \begin{cases} \textbf{Class +1} \ if \ ax_0 + by_0 + cz_0 \geq d \\ \textbf{Class -1} \ \ if \ ax_0 + by_0 + cz_0 < d \end{cases} \tag{4}$$

The minimum, the mean, and the maximum missclassfication error for the weak classifier are $0.01, 0.50$ and 0.99 respectively. The mean error rate is equivalent to flipping a fair coin and selecting Class +1, if a head is observed, or Class -1, if a tail is observed. This is shown in figure 5.

Model	Adaboost	2-NN	Back Propagation
Misclassification Error	0.018	0.045	0.049
Accuracy	0.982	0.955	0.951

Table 1: Misclassification error and accuracy for the 3-cube and the 3-ball on the training data

Model	Adaboost	2-NN	Back Propagation
Misclassification Error	0.021	0.097	0.062
Accuracy	0.979	0.903	0.938

Table 2: Misclassification error and accuracy for the 3-cube and the 3-ball on the test data

Adaboost with the custom defined classifier has the smallest test error rate (largest accuracy) among the three models. This can be seen from table 4.1.1 and figure 7. The result is based on 500 classifiers for the adaboost and 500000 iterations for the neural network with backpropagation.

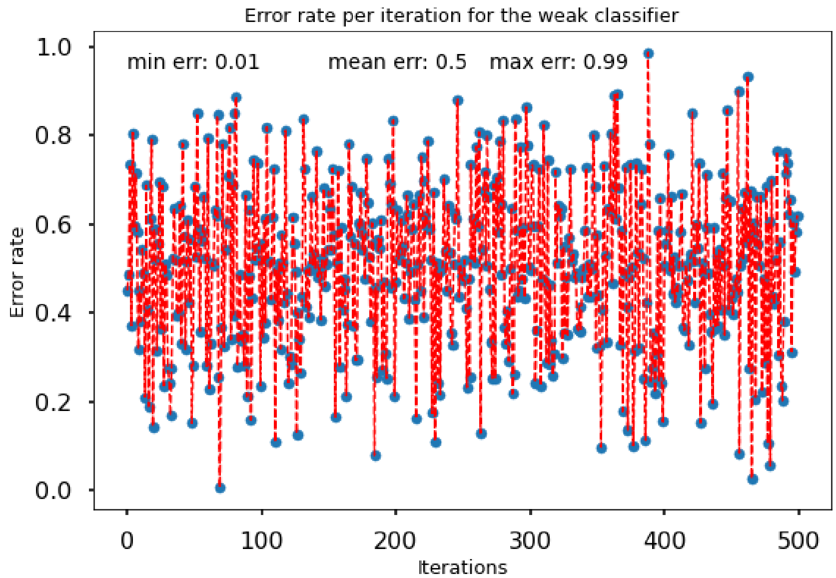

Figure 5: Generated Data for Class +1 and Class −1

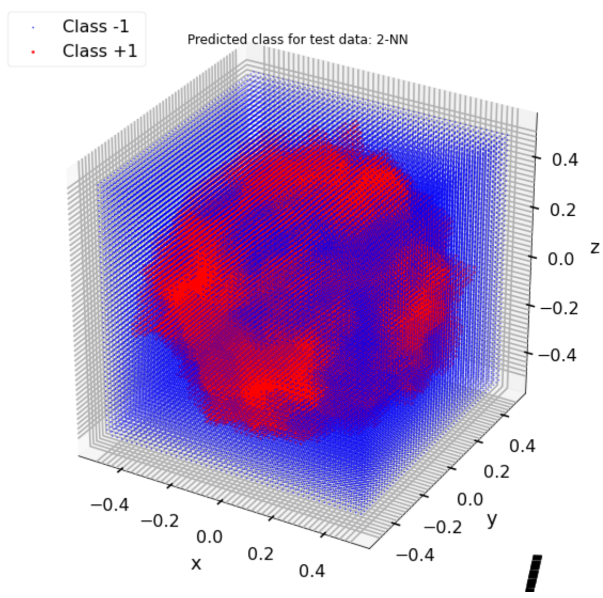

Figure 6: Predicted class for the test data: 2-NN

4.2 Conclusion

Adaboost with the custom defined classifier had the smallest error rate and the greatest accuracy among the models. evaluated. 2-NN had the largest error

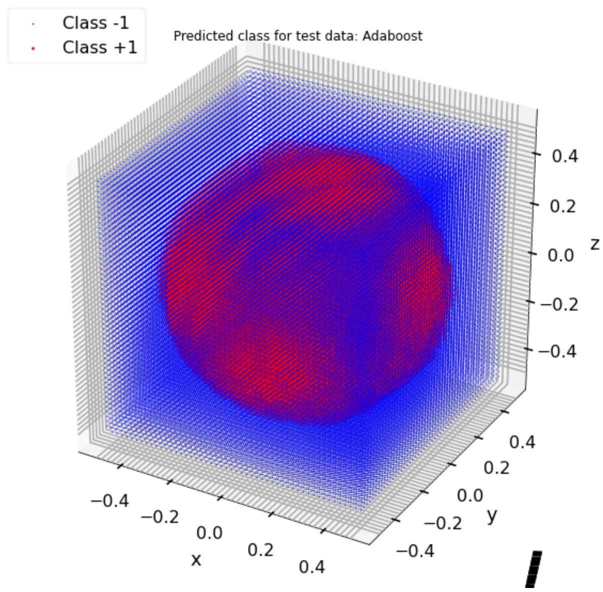

Figure 7: Predicted class for the test data: Adaboost

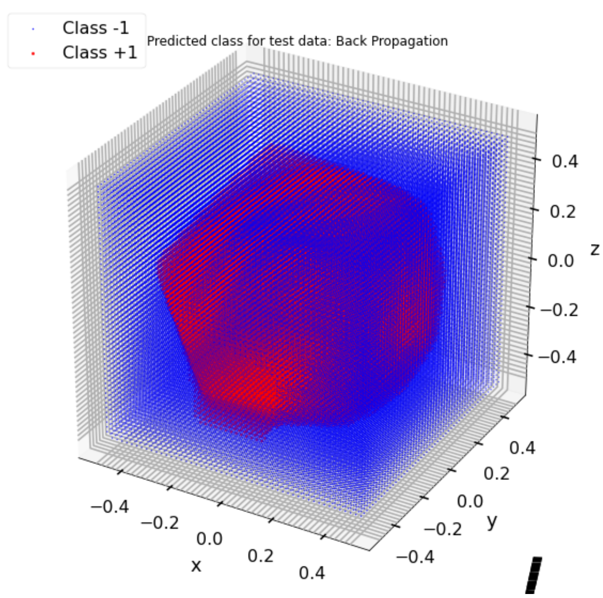

Figure 8: Predicted class for the test data: Back Propagation

rate and the smallest accuracy among the models evaluated. However, we need to evaluate the models for $n \geq 4$ in order to draw any general conclusion.

5 References

https://en.wikipedia.org/wiki/Volume_of_an_n-ball

https://en.wikipedia.org/wiki/N-sphere

https://en.wikipedia.org/wiki/Hypercube

6 Appendix (Code)

```
# -*- coding: utf-8 -*-
"""SubmitFinalProject.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/12z_6Conlr-3HsNNmocp6QjKn2V8Pg7dp
"""

#import modules
from __future__ import division
import torch
import numpy as np
from matplotlib import pyplot as plt
from mpl_toolkits import mplot3d
plt.style.use('seaborn-poster')
from scipy.special import gamma
import pandas as pd

device =  'cuda:0' if torch.cuda.is_available else 'cpu'
dtype = torch.float64
device = 'cpu'
device

def generate_data(size=100000, low=-0.5, high=0.5, type_n=3, prop=0.5, custom_data=None, n_each=None):
    if custom_data is not None:
      size = len(custom_data)
    #output vector
    c1, c2 = [], []
    #define the radius and length
    ll = high - low
    center = np.repeat((low+high)/2, type_n)
    n_volume = ll**type_n

    #checking whether the proportion is valid
    r_max = high - center[0]
    prop_max = (1/n_volume)*(r_max**type_n*(np.pi**(0.5*type_n)))*(1/gamma(type_n*0.5+1))

    #radius
    prop_volume = prop*n_volume
    radius_sq = ((prop_volume*gamma((type_n/2)+1))/(np.pi**(type_n*0.5)))**(2/type_n)

    for j in range(size):
        #generate from uniform distribution
        if custom_data is None:
          tmp = np.random.uniform(low=low, high=high, size=type_n)
        else:
          tmp = custom_data[j]
        tmp_sum = np.sum((tmp-center)**2)

        if tmp_sum <= radius_sq:
```

```
            c2.append(tmp)
        else:
            c1.append(tmp)

    #type conversion
    cube, sphere = np.array(c1), np.array(c2)

    if n_each is not None:
      cube = cube[:n_each]
      sphere = sphere[:n_each]

    features = np.concatenate((cube, sphere), axis=0)
    target = np.concatenate((np.zeros(len(cube)), np.ones(len(sphere))))
    target = pd.get_dummies(target).values

    return cube, sphere, features, target, np.round(prop_max, 4), len(sphere)/size

"""### Loading Data"""

y_train = pd.read_csv('y_train.csv')
x_train = pd.read_csv('x_train.csv')
x_train, y_train = np.array(x_train), np.array(y_train)
x_train, y_train

"""### `Utility Functions`"""

def sigmoid(x):
  x = torch.tensor(x, device=device, dtype=dtype)
  return 1 / (1 + torch.exp(-x))

def mse(target, predicted):
  res = target[:,0] != predicted[:,0]
  return np.mean(res)

"""### Comparing Algorithms

- Back Propagation
"""

class backPropagation():

  def __init__(self, dimension=None, activation_fun = None, device=device, niter=100000, dtype=torch.float64):
    self.dimension = dimension
    self.activation_fun = activation_fun
    self.device = device
    self.niter = niter
    self.dtype=dtype

    #initialize weights
    input_size = self.dimension[0]
    hidden_size = self.dimension[1]
    output_size = self.dimension[2]

    #initializing weight for the hidden layer
```

```
    self.W1 = torch.randn((input_size, hidden_size), device=self.device, dtype=dtype)
    # initializing weight for the output layer
    self.W2 = torch.randn((hidden_size , output_size), device=self.device, dtype=dtype)

    #bias parameter
    bias_W1 = torch.randn((1, hidden_size), device=self.device, dtype=dtype)
    bias_W2 = torch.randn((1, output_size), device=self.device, dtype=dtype)

    #weight with bias
    self.W1_constant = torch.concat((self.W1, bias_W1), axis=0)
    self.W2_constant = torch.concat((self.W2, bias_W2), axis=0)

  def fit(self, xtrain=None, ytrain=None, learning_rate=0.1):
    N, p = xtrain.shape
    xtrain = torch.tensor(xtrain, device=self.device)
    #xtrain = torch.from_numpy(xtrain).type(self.dtype).to(self.device)
    ytrain = torch.tensor(y_train, device=device)
    bias = torch.ones((N,1), device=self.device)
    features = torch.concat((xtrain, bias), axis=1)
    features = torch.tensor(features)

    for itr in range(self.niter):
      Z1 = features @ self.W1_constant
      A1 = self.activation_fun(Z1)
      A1_bias = torch.concat((A1, bias), axis=1)
      #output layer
      Z2 = A1_bias @ self.W2_constant
      A2 = self.activation_fun(Z2)
      E1 = A2 - ytrain
      # backpropagation
      E1 = A2 - ytrain
      #err_norm.append(E1)
      dW1 = E1 * A2 * (1 - A2)
      E2 = dW1 @ self.W2.T
      dW2 = E2 * A1 * (1 - A1)
      #update weight
      W2_update = A1_bias.T @ dW1
      W1_update = features.T @ dW2
      self.W2_constant = self.W2_constant - (learning_rate * W2_update)
      self.W1_constant = self.W1_constant - (learning_rate * W1_update)
      self.W2 = self.W2_constant[:-1, :]

  def predict(self, xtest):
    xtest = torch.tensor(xtest, device=self.device, dtype=self.dtype)
    const = torch.ones((len(xtest),1), dtype=dtype, device=device)
    features_bias= torch.concat((xtest, const), axis=1)
    Z = sigmoid(features_bias @ self.W1_constant)
    Z = torch.concat((Z, const), axis=1)
    out = sigmoid(Z @ self.W2_constant)
```

```
    output = out.cpu().numpy()
    predicted = pd.get_dummies(np.argmax(output, axis=1)).values
    #return
    return predicted


model = backPropagation([3,6,2], device=device, activation_fun=
    sigmoid)
model.fit(x_train, y_train)
ypred = model.predict(x_train)
ypred

mse(ypred, y_train)

"""### `Test Data`"""

h = 0.02
#h = 0.1
dataStep = np.arange(-0.5, 0.5001, h)
dataGrid = np.array([[i,j,k] for i in dataStep for j in dataStep
    for k in dataStep])
df = generate_data(custom_data=dataGrid)
xtest = df[2]
ytest = df[3]
xtest, ytest

xtest.shape, ytest.shape
np.savetxt('x_test.csv', xtest, delimiter=',')
np.savetxt('y_test.csv', ytest, delimiter=',')

ypred_test = model.predict(xtest)
mse(ytest, ypred_test)

"""### Confusion matrix"""

from sklearn.metrics import confusion_matrix
confusion_matrix(ytest[:,0], ypred_test[:,0])

"""### `K-Nearest Neighbors`"""

def euclideanDistance(x,y):
  tmp = x-y
  return np.sqrt(np.dot(tmp, tmp))

def absDistance(x, y):
  return np.sum(x-y)

class kNN():

  def __init__(self, k, distFunc=None):
    self.k = k
    self.distanceFunction = distFunc

  def _fitPredict(self, xrow):
    xtrain = self.xtrain
    ytrain = self.ytrain
    dist = np.apply_along_axis(euclideanDistance, 1, xtrain, xrow)
```

```
    dist_ind = sorted(range(len(dist)), key = lambda sub: dist[sub])[:self.k]
    number_list = ytrain[dist_ind]
    (unique, counts) = np.unique(number_list, return_counts=True)
    prop = counts/self.k
    kSmall = min(prop)
    ind = list(prop).index(kSmall)
    return unique[ind]

  def fitPredict(self, xtrain=None, ytrain=None, xtest=None):
    self.xtrain = xtrain
    self.ytrain = ytrain

    if xtest is None:
      xtest = xtrain
    else:
      xtest = xtest
    ypred = np.apply_along_axis(self._fitPredict, 1, xtest)
    return ypred

  def mse(self, yhat, y):
    return np.mean(yhat != y)

  def accuracy(self, yhat, y):
    return 1 - self.mse(yhat, y)

"""#### Prediction"""

''' model = kNN(1, absDistance)
y_train_mod = y_train[:,0]
y_test = ytest[:,0]
yhat = model.fitPredict(x_train, y_train_mod, x_train)
model.mse(yhat, y_train_mod) '''

model = kNN(2, euclideanDistance)
y_train_mod = y_train[:,0]
y_test = ytest[:,0]
yhat = model.fitPredict(x_train, y_train_mod, x_train)
model.mse(yhat, y_train_mod)

"""#### Test Data"""

model = kNN(2, euclideanDistance)
yhat = model.fitPredict(x_train, y_train_mod, xtest)
model.mse(yhat, y_test)

confusion_matrix(yhat, y_test)

"""#### Prediction for `N=4`"""

y_train1 = pd.read_csv('y_train4.csv')
x_train1 = pd.read_csv('x_train4.csv')
x_train1, y_train1 = np.array(x_train1), np.array(y_train1)
x_train1, y_train1
```

```
"""#### Test Data"""

h = 0.055
#h = 0.1
dataStep = np.arange(-0.5, 0.5001, h)
dataGrid1 = np.array([[i,j,k, w] for i in dataStep for j in
    dataStep for k in dataStep for w in dataStep])
df1 = generate_data(custom_data=dataGrid1, type_n=4)
xtest1 = df1[2]
ytest1 = df1[3]
xtest1, ytest1.shape

np.savetxt('x_test1.csv', xtest1, delimiter=',')
np.savetxt('y_test1.csv', ytest1, delimiter=',')

"""#### Back Propagation"""

model1 = backPropagation([4,5,2], device=device, activation_fun=
    sigmoid, niter=400000)
model1.fit(x_train1, y_train1)
ypred1 = model1.predict(xtest1)
ypred1

mse(ypred1, ytest1)

confusion_matrix(ypred1[:,0], ytest1[:,0])

"""### KNN"""

model = kNN(2, euclideanDistance)
y_train_mod1 = y_train1[:,0]
y_test1 = ytest1[:,0]
yhat = model.fitPredict(x_train1, y_train_mod1, x_train1)
model.mse(yhat, y_train_mod1)

model = kNN(2, euclideanDistance)
y_train_mod1 = y_train1[:,0]
y_test1 = ytest1[:,0]
yhat = model.fitPredict(x_train1, y_train_mod1, xtest1)
model.mse(yhat, y_test1)

confusion_matrix(yhat, y_test1)
```