KNN

A supervised learning technique that considers the k(number) nearest neighbour. Consider the following training and validation set for a movie dataset. Your objective is to identify the movie class/category given in the test data set based on the number of comedy and action scenes. You are advised to use the concept of array to accomplish this task

In [279…

```
import csv
import random as rd
```

In [280…

```
# main KNN class
class Ag_KNN:
    # sub class for holding the label and data
    class Ag_Scatterpoint:
        def __init__(self, label, data):
            self.label = label
            self.data = data

        # sld -> Euclidean distance
        def sld(self, other_data):
            val = 0

            for i in range(len(self.data)):
                val += (self.data[i] - other_data[i]) ** 2
            return val ** (1/2)

    # monoclass for single answers, monoclass = False gives the list of possible la
    def __init__(self, monoclass = False):
        self.monoclass = monoclass
        self.distance_val = []

    # This is for pushing the train and validation data, and other parameters
    def data_set(self, train_d, val_d, k = -1, k_limit = 10, print_all: bool = Fals
        self.train_Ag_num = len(train_d)
        self.k = k

        # data holding varaibles
        self.train_data = []
        self.val_data = []

        # make the data in Scatterpoints
        for sp in val_d:
            self.val_data.append(self.Ag_Scatterpoint(sp[0], sp[1]))
        for sp in train_d:
            self.train_data.append(self.Ag_Scatterpoint(sp[0], sp[1]))

        # if k not initialised find k
        if(self.k == -1):
            if(k_limit > self.train_Ag_num):
                k_limit = int(self.train_Ag_num / 2)
            self.find_best_k(k_limit)

        # display current parameters
        self.Ag_validation(self.k, print_all)
        print("accuracy =", self.accuracy)
        print("k =", self.k)
        print()

    # this is the validation function to check for current k
```

```
    def Ag_validation(self, k, print_all: bool):
        correct = 0

        num = len(self.val_data)
        for i in range(num):
            if(print_all):
                print("data for:", self.val_data[i].label,self.val_data[i].data)

            self.calculate(self.val_data[i].data, print_all)

            # count frequency
            Ag_k_nearest = {}
            for val in (self.distance_val[:k]):
                if(val[1] in Ag_k_nearest):
                    Ag_k_nearest[val[1]] += 1
                else:
                    Ag_k_nearest[val[1]] = 1

            # Recreate the dictionary to make it sorted
            Ag_k_nearest = dict(sorted(Ag_k_nearest.items(), key=lambda item: item[
            Ag_pred_val = list(Ag_k_nearest.keys())[0]

            if(Ag_pred_val == self.val_data[i].label):
                correct += 1

        accuracy = correct / num
        self.accuracy = accuracy

    # to find the best k
    def find_best_k(self, max_k):
        if(max_k > self.train_Ag_num):
            max_k = self.train_Ag_num

        best_k = 1
        Ag_accuracy = -1

        # try validation for each k and then update
        for k in range(1, max_k + 1, 1 + self.monoclass):
            self.Ag_validation(k, False)
            if(self.accuracy > Ag_accuracy):
                Ag_accuracy = self.accuracy
                best_k = k

        # final updates
        self.accuracy = Ag_accuracy
        self.k = best_k

    # calculates all distances and sorts them
    def calculate(self, input_data, print_all: bool = False):
        num = len(self.train_data)
        distances = []

        # clear it for next data point values
        self.distance_val.clear()

        # distances
        for i in range(num):
            distance = self.train_data[i].sld(input_data)
            distances.append((distance, self.train_data[i].label, self.train_data[i
            self.distance_val.append((distance, self.train_data[i].label))

        # Sort by distance and make it internal
        distances.sort(key=lambda d: d[0])
        self.distance_val.sort(key = lambda d: d[0])
```

```
        if(print_all):
            for d in distances:
                print(d[1],d[2],"->",d[0])

    # used for testing and prediction
    def Ag_knn_run(self, input_point, print_all: bool = False):
        if(print_all):
            print("input data =", input_point)
            print("k used =", self.k)

        self.calculate(input_point, print_all)

        # count frequency
        Ag_k_nearest = {}
        for point_tuple in (self.distance_val[:self.k]):
            if(point_tuple[1] not in Ag_k_nearest):
                Ag_k_nearest[point_tuple[1]] = 1 / self.k
            else:
                Ag_k_nearest[point_tuple[1]] += 1 / self.k

        # Recreate the dictionary to make it sorted
        Ag_k_nearest = dict(sorted(Ag_k_nearest.items(), key=lambda item: item[1],

        # Return the most common label
        if(self.monoclass):
            return list(Ag_k_nearest.keys())[0]
        return Ag_k_nearest
```

In [281…

```
# [(label, [comedy, action])]
train_data = [("C", (100,0)), ("A", (0,100)), ("A", (15,90)), ("C", (85,20))]
validation_data = [("A", (10,95)), ("C", (85,15))]

# test data
test_data = [(6,70), (93,23), (50,50)]
```

In [282…

```
# Test knn
smal_Ag_knn = Ag_KNN(True)

# print validation for k = 1
smal_Ag_knn.data_set(train_data, validation_data, 1, print_all= True)

# print validation for k = 3
smal_Ag_knn.data_set(train_data, validation_data, 3, print_all= True)
```

```
data for: A (10, 95)
A (15, 90) -> 7.0710678118654755
A (0, 100) -> 11.180339887498949
C (85, 20) -> 106.06601717798213
C (100, 0) -> 130.86252328302402
data for: C (85, 15)
C (85, 20) -> 5.0
C (100, 0) -> 21.213203435596427
A (15, 90) -> 102.59142264341595
A (0, 100) -> 120.20815280171308
accuracy = 1.0
k = 1

data for: A (10, 95)
A (15, 90) -> 7.0710678118654755
A (0, 100) -> 11.180339887498949
C (85, 20) -> 106.06601717798213
C (100, 0) -> 130.86252328302402
data for: C (85, 15)
C (85, 20) -> 5.0
C (100, 0) -> 21.213203435596427
A (15, 90) -> 102.59142264341595
A (0, 100) -> 120.20815280171308
accuracy = 1.0
k = 3
```

In [283…

```python
print("Test data: ")

# Test Data values
print(test_data[0] ,"->", smal_Ag_knn.Ag_knn_run(test_data[0]))
print(test_data[1] ,"->", smal_Ag_knn.Ag_knn_run(test_data[1]))
print(test_data[2] ,"->", smal_Ag_knn.Ag_knn_run(test_data[2], True))
```

```
Test data:
(6, 70) -> A
(93, 23) -> C
input data = (50, 50)
k used = 3
C (85, 20) -> 46.09772228646444
A (15, 90) -> 53.150729063673246
C (100, 0) -> 70.71067811865476
A (0, 100) -> 70.71067811865476
(50, 50) -> C
```

Parsing the Iris dataset into chunks

In [284…

```python
# Dictionary to hold all data
all_data = {}

# file parsing .csv
with open("iris.csv", "r") as fp:
    csv_r = csv.reader(fp)
    header = True

    # hard coded normalisation values
    norm_val = [7.9, 4.4, 6.9, 2.5]
    n = 5

    for row in csv_r:
        if(header):
            header = False
            print(row)
            continue
```

```
        data = []
        for i in range(n - 1):
            data.append(float(row[i])/norm_val[i])
        label = row[n - 1]

        if(label not in all_data):
            all_data[label] = [(label, data)]
        else:
            all_data[label].append((label, data))
    fp.close()

num = 50

valid_data = []
train_data = []
test_data = []
```

```
['sepal_length 7.9', 'sepal_width 4.4', 'petal_length 6.9', 'petal_width 2.5', 'sp
ecies']
```

In [285…

```
# parameter to divide test_train_validation
# test(0) -> train(1)
Ag_test_train_div = 0.7
# train(0) -> validate(1)
Ag_valid_train_div = 0.3

# no random trials
Ag_rd_num = 0

valid_data.clear()
train_data.clear()
test_data.clear()

# Randomise
def Ag_random(Ag_num = 0):
    rd.seed(Ag_num)
    if(Ag_num > 0):
        for label in all_data:
            print("1",all_data[label][0])
            shf_label = all_data[label]
            rd.shuffle(shf_label)
            all_data[label] = shf_label
            print("2",all_data[label][0],"\n")

Ag_random(Ag_rd_num)

# push the correct lens of data in the lists
for label in all_data:
    valid_data.extend(all_data[label][:int(Ag_valid_train_div  * Ag_test_train_div
    train_data.extend(all_data[label][int(Ag_valid_train_div  * Ag_test_train_div *
    test_data.extend(all_data[label][int(Ag_test_train_div * num):])

print("validation:",len(valid_data))
print("train:",len(train_data))
print("test:",len(test_data))
```

```
validation: 30
train: 75
test: 45
```

In [286…

```
Ag_Iris_Knn = Ag_KNN(True)
Ag_Iris_Knn.data_set(train_data, valid_data)

if(Ag_rd_num > 0):
```

```python
    print("Seeded random", Ag_rd_num)

for d in test_data:
    Ag_pred_val = Ag_Iris_Knn.Ag_knn_run(d[1])
    print(d[0], "prediction:", Ag_pred_val)
```

```
accuracy = 0.9666666666666667
k = 1

setosa prediction: setosa
setosa prediction: setosa
setosa prediction: setosa
setosa prediction: setosa
setosa prediction: setosa
setosa prediction: setosa
setosa prediction: setosa
setosa prediction: setosa
setosa prediction: setosa
setosa prediction: setosa
setosa prediction: setosa
setosa prediction: setosa
setosa prediction: setosa
setosa prediction: setosa
setosa prediction: setosa
versicolor prediction: versicolor
versicolor prediction: versicolor
versicolor prediction: versicolor
versicolor prediction: versicolor
versicolor prediction: versicolor
versicolor prediction: versicolor
versicolor prediction: versicolor
versicolor prediction: versicolor
versicolor prediction: versicolor
versicolor prediction: versicolor
versicolor prediction: versicolor
versicolor prediction: versicolor
versicolor prediction: versicolor
versicolor prediction: versicolor
versicolor prediction: versicolor
virginica prediction: virginica
virginica prediction: virginica
virginica prediction: virginica
virginica prediction: virginica
virginica prediction: virginica
virginica prediction: virginica
virginica prediction: virginica
virginica prediction: virginica
virginica prediction: virginica
virginica prediction: virginica
virginica prediction: virginica
virginica prediction: virginica
virginica prediction: virginica
virginica prediction: virginica
virginica prediction: virginica
```