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 [1]:

```
class KNN:
    class ScatterPoints:
        def __init__(self, label, data):
            self.label = label
            self.data = data

        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)

    def __init__(self, monoclass = False):
        self.monoclass = monoclass
        self.distance_val = []

    def data_set(self, train_d, val_d, k = -1, k_limit = 20):
        self.num = len(train_d)
        self.k = k

        self.train_data = []
        self.val_data = []

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

        if(self.k == -1):
            if(k_limit > self.num):
                k_limit = int(self.num / 2)
            self.find_best_k(k_limit)

        self.validate_model(self.k)
        print("accuracy =", self.accuracy)
        print("k =", self.k)

    def validate_model(self, k, print_all: bool = False):
        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)

            k_nearest = {}
            for val in (self.distance_val[:k]):
                if(val[1] in k_nearest):
                    k_nearest[val[1]] += 1
                else:
                    k_nearest[val[1]] = 1

            k_nearest = dict(sorted(k_nearest.items(), key=lambda item: item[1], re
            pred_val = list(k_nearest.keys())[0]

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

        accuracy = correct / num
        self.accuracy = accuracy

    def find_best_k(self, max_k):
        if(max_k > self.num):
            max_k = self.num

        best_k = 1
        best_accuracy = -1

        for k in range(1, max_k + 1, 1 + self.monoclass):
            self.validate_model(k)
            if(self.accuracy >= best_accuracy):
                best_accuracy = self.accuracy
                best_k = k

        self.accuracy = best_accuracy
        self.k = best_k

    def calculate(self, input_data, print_all: bool = False):
        num = len(self.train_data)
        distances = []
        self.distance_val.clear()

        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])

    def run_knn(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)

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

        k_nearest = dict(sorted(k_nearest.items(), key=lambda item: item[1], revers

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

In [2]:

```
# [(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 [3]:

```
knn = KNN(True)

# print validation for k = 1
knn.data_set(train_data, validation_data, 1)
knn.validate_model(1, True)
```

```
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
```

In [7]:

```
# print validation for k = 3
knn.data_set(train_data, validation_data, 3)
knn.validate_model(1, True)

print("\nTest data: ")

for t in test_data:
    print(t,"->", knn.run_knn(t))
    #print("answer:", knn.run_knn(t, True),"\n")
```

```
accuracy = 1.0
k = 3
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

Test data:
(6, 70) -> A
(93, 23) -> C
(50, 50) -> C
```

Parsing the Iris dataset into chunks

In [8]:

```
import csv

all_data = {}
```

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

    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 [9]:

```
# parameter to divide test_train_validation
# test(0) -> train(1)
t_div = 0.7

# train(0) -> validate(1)
t1_div = 0.3

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

for label in all_data:
    valid_data.extend(all_data[label][:int(t1_div  * t_div * num)])
    train_data.extend(all_data[label][int(t1_div  * t_div * num):int(t_div * num)])
    test_data.extend(all_data[label][int(t_div * num):])

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

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

In [12]:

```
Iris_Knn = KNN(True)
Iris_Knn.data_set(train_data, valid_data)

for d in test_data:
    pred_val = Iris_Knn.run_knn(d[1])
    print(d[0], "prediction:", pred_val)
```

```
accuracy = 0.9666666666666667
k = 19
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
```