In [19]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from IPython.display import display, HTML
```

Preparing The Data

In [20]:
```python
# Training Data
hem = pd.DataFrame({
    "comedy":[100,0,15,85],
    "action":[0,100,90,20],
    "class":["comedy", "action", "action", "comedy"]
},)

hem
```

Out[20]:

	comedy	action	class
0	100	0	comedy
1	0	100	action
2	15	90	action
3	85	20	comedy

In [21]:
```python
#Validation Data
ant = pd.DataFrame({
    "comedy":[10,85],
    "action":[95,15],
    "class":["action","comedy"]
})
ant
```

Out[21]:

	comedy	action	class
0	10	95	action
1	85	15	comedy

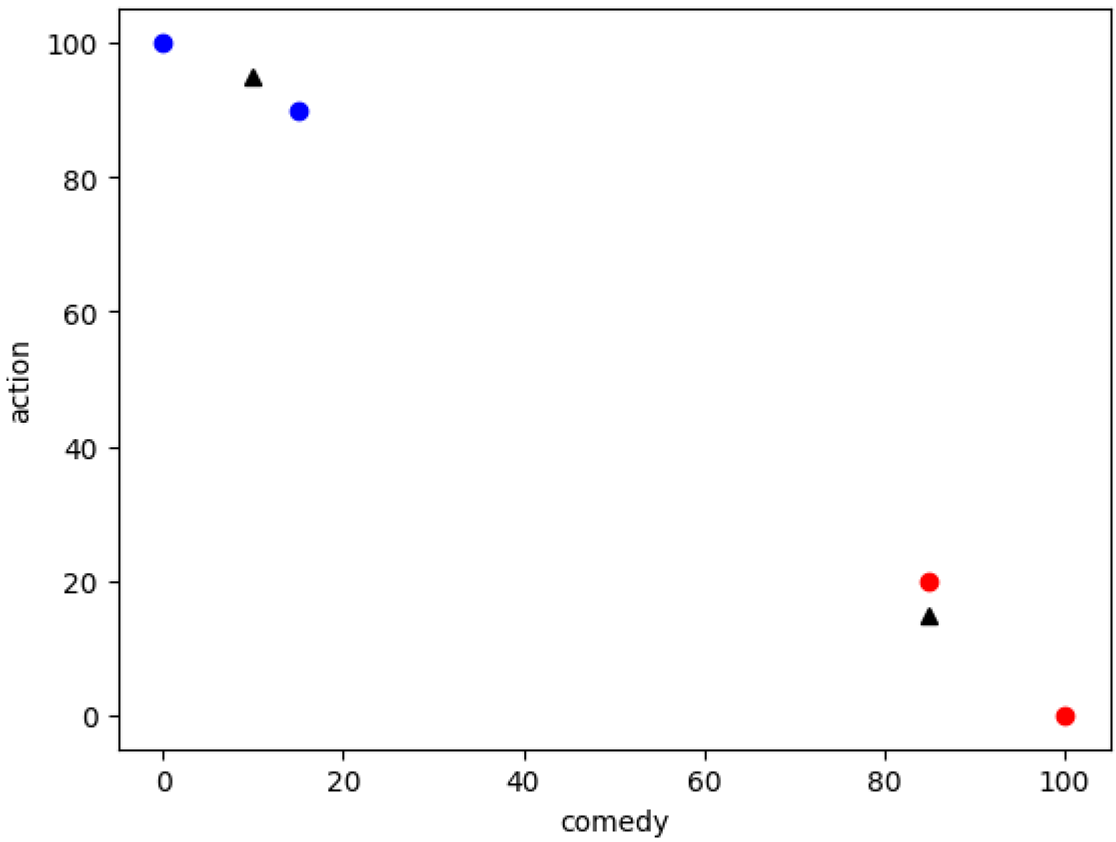
In [22]:
```python
kum = pd.DataFrame({
    "comedy":[6, 93, 50],
    "action":[70, 23, 50]
})
kum
```

Out[22]:

	comedy	action
0	6	70
1	93	23
2	50	50

Visualizing

In [23]:
```
plt.plot(hem[hem['class']=='comedy']['comedy'], hem[hem['class']=='comedy']
['action'],'o', color='r' )
plt.plot(hem[hem['class']=='action']['comedy'], hem[hem['class']=='action']
['action'],'o', color='b', )
plt.plot(ant['comedy'], ant['action'],'k^')
plt.xlabel("comedy")
plt.ylabel("action")
plt.show()
```



Calculating The Distance for n dimensions (Here 2)

In [24]:
```
import math


# call as euclid_dist((1,2,3), (3,4,5)); or euclid_dist((1,2), (3,4));
def euclid_dist(h, k):
    hemant = 0

    #assuming len(p1) == len(p2)
    assert len(h) == len(k)

    for e in range(len(h)):
        hemant += (h[e] - k[e])**2;
    hemant_kumar = math.sqrt(hemant);
    return hemant_kumar
```

In [25]:
```
hem.columns
```

Out[25]:
```
Index(['comedy', 'action', 'class'], dtype='object')
```

Finding k by comparing each point in validation set with neighbours

making it general so that it can be used for any number of dimensions and any number of features

In [26]:
```
def get_best_k(hema, ntku,h_kum = None, hku=True):
    if(h_kum is None):
         h_kum = range(1, len(hema)+1, 2)
    hk = len(h_kum)
    hkr = pd.DataFrame({"k":[i for i in h_kum],"pointsEvaluated":np.zeros(h
k), "correctPred": np.zeros(hk),"accuracy":np.zeros(hk)})

    her = [e for e in hema.columns if e != 'class']
    for e in ntku.index:
        hem_kumar =  ntku.loc[e, 'class'];
        if(hku):
            print("For eval point:", e)
            print("Actual Class:" ,hem_kumar)

        emant = []
        for j in hema.index:
                p1= [ntku.loc[e, f] for f in her]
                p2 = [hema.loc[j,f] for f in her]

                dist = euclid_dist(p1, p2)

                emant.append({"dist":dist, "class":hema.loc[j, 'class'], "v
al":e, "train":j});
        emant = pd.DataFrame(emant)
        emant = emant.sort_values(by='dist')

        if(hku):
            display(HTML(emant.to_html()))
            print("-----------------------------------")

        # Iterating all odd values
        for k in h_kum:
            ekr = emant.iloc[:k]['class'].value_counts().idxmax()
            hkr.loc[hkr['k'] == k, 'pointsEvaluated'] += 1
            hkr.loc[hkr['k'] == k, 'correctPred'] += (1 if hem_kumar == ekr
else 0)
            hkr.loc[hkr['k'] == k, 'accuracy'] = hkr.loc[hkr['k'] == k, 'co
rrectPred']*100/hkr.loc[hkr['k'] == k, 'pointsEvaluated']

    mkr = hkr.sort_values(by=['accuracy'], ascending=False)
    print("Comparing All k\n")
    print(mkr)
    return mkr.iloc[0]['k']


akr = get_best_k(hem, ant, h_kum=[1,3])
print("\nBest K:",akr)
```

```
For eval point: 0
Actual Class: action
```

	dist	class	val	train
2	7.071068	action	0	2
1	11.180340	action	0	1
3	106.066017	comedy	0	3
0	130.862523	comedy	0	0

```
-----------------------------------
For eval point: 1
Actual Class: comedy
```

	dist	class	val	train
3	5.000000	comedy	1	3
0	21.213203	comedy	1	0
2	102.591423	action	1	2
1	120.208153	action	1	1

```
-----------------------------------
Comparing All k

   k  pointsEvaluated  correctPred  accuracy
0  1              2.0          2.0     100.0
1  3              2.0          2.0     100.0

Best K: 1.0
```

Predicting on Test Data

In [27]:
```
def predict(k, hemn, hemt):
  hemt['predicted_class'] = np.nan
  k = int(k)
  emku = [e for e in hemn.columns if e != 'class']
  for e in hemt.index:
    mnt = []
    for u in hemn.index:
      h = [hemt.loc[e, f] for f in emku]
      ku = [hemn.loc[u, f] for f in emku]
      kuma = euclid_dist(h, ku)
      mnt.append({"dist": kuma, "class": hemn.loc[u, 'class']})
    mnt = pd.DataFrame(mnt)
    mnt = mnt.sort_values(by='dist')

    t_kumar = mnt.iloc[:k]["class"].value_counts().idxmax()
    hemt.loc[e,'predicted_class'] = t_kumar
  return hemt
predict(akr, hem, kum);

display(HTML(kum.to_html()))
```

```
C:\Users\Hp\AppData\Local\Temp\ipykernel_11284\2490563322.py:16: FutureWar
ning: Setting an item of incompatible dtype is deprecated and will raise a
n error in a future version of pandas. Value 'action' has dtype incompatib
le with float64, please explicitly cast to a compatible dtype first.
  hemt.loc[e,'predicted_class'] = t_kumar
```

	comedy	action	predicted_class
0	6	70	action
1	93	23	comedy
2	50	50	comedy

Iris Dataset

Since i have created all the functions in a generalized way, we can use the same functions again here

Loading The Dataset

In [28]:
```
from sklearn import datasets

hat = datasets.load_iris()

print(hat.DESCR)
```

.. _iris_dataset:

Iris plants dataset

Data Set Characteristics:

:Number of Instances: 150 (50 in each of three classes)
:Number of Attributes: 4 numeric, predictive attributes and the class
:Attribute Information:
    - sepal length in cm
    - sepal width in cm
    - petal length in cm
    - petal width in cm
    - class:
            - Iris-Setosa
            - Iris-Versicolour
            - Iris-Virginica

:Summary Statistics:

	Min	Max	Mean	SD	Class Correlation
sepal length:	4.3	7.9	5.84	0.83	0.7826
sepal width:	2.0	4.4	3.05	0.43	-0.4194
petal length:	1.0	6.9	3.76	1.76	0.9490 (high!)
petal width:	0.1	2.5	1.20	0.76	0.9565 (high!)

:Missing Attribute Values: None
:Class Distribution: 33.3% for each of 3 classes.
:Creator: R.A. Fisher
:Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
:Date: July, 1988

The famous Iris database, first used by Sir R.A. Fisher. The dataset is ta
ken
from Fisher's paper. Note that it's the same as in R, but not as in the UC
I
Machine Learning Repository, which has two wrong data points.

This is perhaps the best known database to be found in the
pattern recognition literature. Fisher's paper is a classic in the field
and
is referenced frequently to this day. (See Duda & Hart, for example.) Th
e
data set contains 3 classes of 50 instances each, where each class refers
to a
type of iris plant. One class is linearly separable from the other 2; the
latter are NOT linearly separable from each other.

|details-start|
References
|details-split|

- Fisher, R.A. "The use of multiple measurements in taxonomic problems"
  Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to
  Mathematical Statistics" (John Wiley, NY, 1950).
- Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysi
s.

```
  (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.
- Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System
  Structure and Classification Rule for Recognition in Partially Exposed
  Environments".  IEEE Transactions on Pattern Analysis and Machine
  Intelligence, Vol. PAMI-2, No. 1, 67-71.
- Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transactio
ns
  on Information Theory, May 1972, 431-433.
- See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II
  conceptual clustering system finds 3 classes in the data.
- Many, many more ...

|details-end|
```

In [29]:
```python
hat.keys()
```

Out[29]:
```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_na
mes', 'filename', 'data_module'])
```

In [30]:
```python
print("Target Names:", hat.target_names)
print("Feature Names:", hat.feature_names)
```

```
Target Names: ['setosa' 'versicolor' 'virginica']
Feature Names: ['sepal length (cm)', 'sepal width (cm)', 'petal length (c
m)', 'petal width (cm)']
```

In [31]:
```python
heant = pd.DataFrame(hat.data, columns=hat.feature_names)
heant
```

Out[31]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

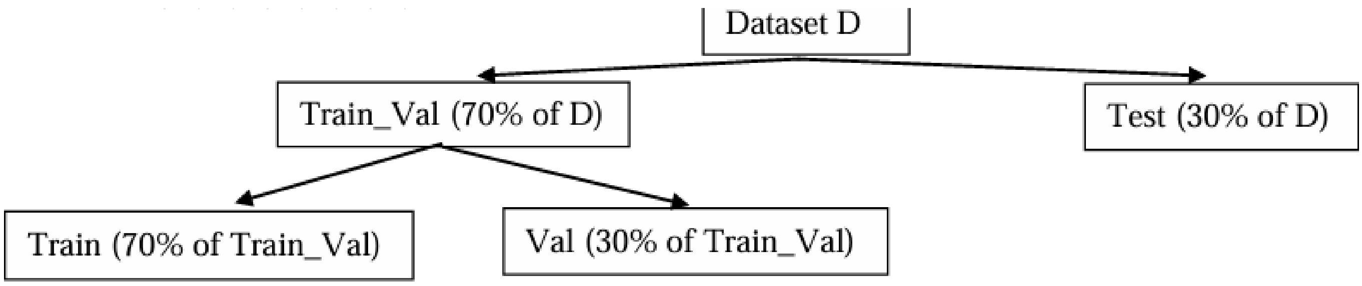
In [32]:
```python
heant['class'] = hat.target
heant
```

Out[32]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	class
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	2
146	6.3	2.5	5.0	1.9	2
147	6.5	3.0	5.2	2.0	2
148	6.2	3.4	5.4	2.3	2
149	5.9	3.0	5.1	1.8	2

150 rows × 5 columns

Train-Test-Validation Split



In [33]:
```python
def split_data_equally(hkr, hkar, hmkar):
  hemantkum = [h.sample(frac=1, random_state=hmkar) for _, h in hkr.groupby
('class')]  # Shuffle within each class
  hemkr = pd.DataFrame()
  hemantkuar = pd.DataFrame()
  for i in hemantkum:
      hemkr = pd.concat([hemkr, i.iloc[:int(hkar * len(i))]])
      hemantkuar = pd.concat([hemantkuar, i.iloc[int(hkar * len(i)):]])

  return hemkr.sample(frac=1, random_state=hmkar), hemantkuar.sample(frac=
1, random_state=hmkar)
hemankumar, kum = split_data_equally(heant,hkar= 0.7, hmkar=42)
hem, ant = split_data_equally(hemankumar,hkar= 0.7, hmkar=42)
print("Train shape:", hem.shape)
print("Validation shape:", ant.shape)
print("Test shape:", kum.shape)
```

```
Train shape: (72, 5)
Validation shape: (33, 5)
Test shape: (45, 5)
```

In [37]:
```python
hemant_kuma = kum['class']
hemant_kum = kum.drop(columns=['class'])
```

In [35]:
```python
akr = get_best_k(hem, ant, h_kum=range(1, 10), hku=False)
print("Best K:", akr)
hemant_kum = predict(akr, hemankumar, hemant_kum)
preds = hemant_kum['predicted_class']
hemant_kum.head()
```

```
Comparing All k

   k  pointsEvaluated  correctPred    accuracy
0  1             33.0         33.0  100.000000
1  2             33.0         33.0  100.000000
2  3             33.0         33.0  100.000000
3  4             33.0         33.0  100.000000
4  5             33.0         33.0  100.000000
5  6             33.0         33.0  100.000000
6  7             33.0         32.0   96.969697
7  8             33.0         32.0   96.969697
8  9             33.0         32.0   96.969697
Best K: 1.0
```

Out[35]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	predicted_class
120	6.9	3.2	5.7	2.3	2.0
57	4.9	2.4	3.3	1.0	1.0
92	5.8	2.6	4.0	1.2	1.0
128	6.4	2.8	5.6	2.1	2.0
110	6.5	3.2	5.1	2.0	2.0

In [36]:
```python
from sklearn.metrics import accuracy_score
accuracy_score(hemant_kuma, preds)
```

Out[36]: 0.9555555555555556