Name: Sidhantha Poddar

Reg: 17BCE2044

In [33]:

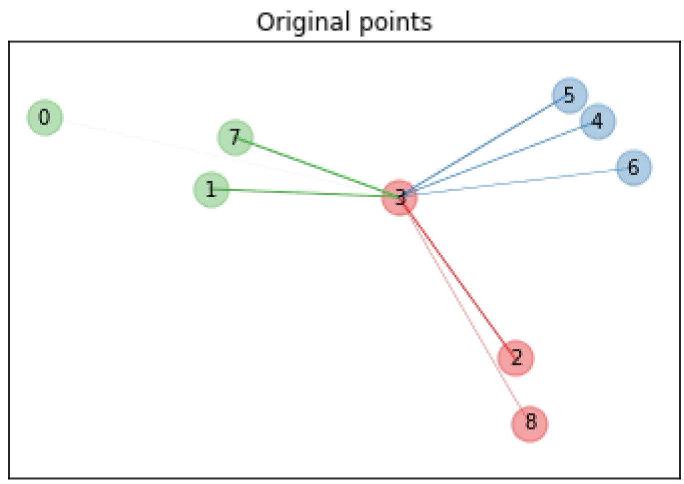
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
#from sklearn.neighbors import NeighborhoodComponentsAnalysis
from matplotlib import cm
from sklearn.utils.fixes import logsumexp
import sklearn.neighbors
print(__doc__)
import pandas as pd
from numpy import hstack,dstack,vstack
```

```
Automatically created module for IPython interactive environment
```

vizulation of KNN using simple dataset

In [34]:

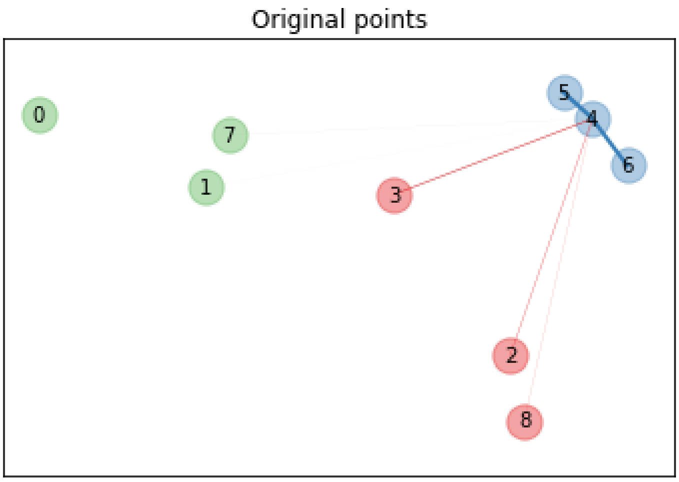
```
X, y = make_classification(n_samples=9, n_features=2, n_informative=2,
                           n_redundant=0, n_classes=3, n_clusters_per_class=1,
                           class_sep=1.0, random_state=0)

plt.figure(1)
ax = plt.gca()
for i in range(X.shape[0]):
    ax.text(X[i, 0], X[i, 1], str(i), va='center', ha='center')
    ax.scatter(X[i, 0], X[i, 1], s=300, c=cm.Set1(y[[i]]), alpha=0.4)

ax.set_title("Original points")
ax.axes.get_xaxis().set_visible(False)
ax.axes.get_yaxis().set_visible(False)
ax.axis('equal')  # so that boundaries are displayed correctly as circles


def link_thickness_i(X, i):
    diff_embedded = X[i] - X
    dist_embedded = np.einsum('ij,ij->i', diff_embedded,
                              diff_embedded)
    dist_embedded[i] = np.inf

    # compute exponentiated distances (use the log-sum-exp trick to
    # avoid numerical instabilities
    exp_dist_embedded = np.exp(-dist_embedded -
                               logsumexp(-dist_embedded))
    return exp_dist_embedded


def relate_point(X, i, ax):
    pt_i = X[i]
    for j, pt_j in enumerate(X):
        thickness = link_thickness_i(X, i)
        if i != j:
            line = ([pt_i[0], pt_j[0]], [pt_i[1], pt_j[1]])
            ax.plot(*line, c=cm.Set1(y[j]),
                    linewidth=5*thickness[j])


i = 3
relate_point(X, i, ax)
plt.show()
```



Knn for point 4

In [35]:

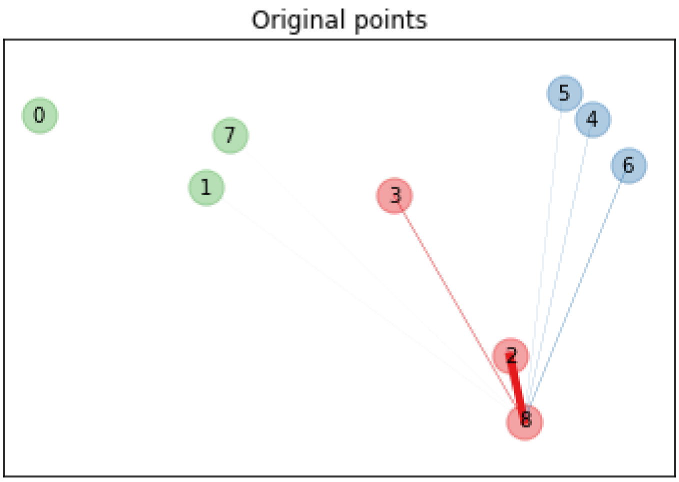
```
X, y = make_classification(n_samples=9, n_features=2, n_informative=2,
                           n_redundant=0, n_classes=3, n_clusters_per_class=1,
                           class_sep=1.0, random_state=0)

plt.figure(1)
ax = plt.gca()
for i in range(X.shape[0]):
    ax.text(X[i, 0], X[i, 1], str(i), va='center', ha='center')
    ax.scatter(X[i, 0], X[i, 1], s=300, c=cm.Set1(y[[i]]), alpha=0.4)

ax.set_title("Original points")
ax.axes.get_xaxis().set_visible(False)
ax.axes.get_yaxis().set_visible(False)
ax.axis('equal')  # so that boundaries are displayed correctly as circles
i = 4
relate_point(X, i, ax)
plt.show()
```



Knn for point 8

In [36]:

```
X, y = make_classification(n_samples=9, n_features=2, n_informative=2,
                           n_redundant=0, n_classes=3, n_clusters_per_class=1,
                           class_sep=1.0, random_state=0)

plt.figure(1)
ax = plt.gca()
for i in range(X.shape[0]):
    ax.text(X[i, 0], X[i, 1], str(i), va='center', ha='center')
    ax.scatter(X[i, 0], X[i, 1], s=300, c=cm.Set1(y[[i]]), alpha=0.4)

ax.set_title("Original points")
ax.axes.get_xaxis().set_visible(False)
ax.axes.get_yaxis().set_visible(False)
ax.axis('equal')  # so that boundaries are displayed correctly as circles
i = 8
relate_point(X, i, ax)
plt.show()
```



KNN classifier for iris dataset

In [37]:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn import datasets
from sklearn.model_selection import train_test_split


n_neighbors = 1

dataset = datasets.load_iris()
X, y = dataset.data, dataset.target

# we only take two features. We could avoid this ugly
# slicing by using a two-dim dataset
X = X[:, [0, 2]]

X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0.7
```

In [38]:

```
pd.DataFrame(X_train).head()
```

Out[38]:

	0	1
0	4.4	1.4
1	6.8	5.5
2	5.5	4.0
3	4.8	1.4
4	5.8	5.1

In [39]:

```
from sklearn.neighbors import KNeighborsClassifier
```

In [40]:

```
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=1, n_neighbors=5, p=2,weights='unif

print("Predictions form the classifier:")
print(knn.predict(X_test))
print("Target values:")
print(y_test)
```

```
Predictions form the classifier:
[0 0 2 0 0 0 2 1 1 1 1 1 0 2 1 0 2 1 0 0 0 0 2 2 1 1 1 1 0 2 1 2 1 1 0
1 1
 0 2 2 2 0 2 1 0 1 0 0 0 1 0 0 2 2 2 0 2 1 0 2 2 0 2 2 1 2 1 2 0 2 2 2
0 0
 2 1 0 1 0 2 2 1 1 2 1 0 1 1 1 0 0 2 1 2 2 2 1 2 1 0 0 1 0 1 1]
Target values:
[0 0 2 0 0 0 2 1 1 1 1 1 0 2 1 0 2 1 0 0 0 0 2 2 1 1 2 1 0 2 1 2 1 1 0
1 1
 0 2 2 2 0 2 1 0 1 0 0 0 1 0 0 2 2 2 0 2 1 0 1 2 0 2 2 1 2 1 2 0 2 2 2
0 0
 2 1 0 1 0 2 2 1 1 2 1 0 1 1 2 0 0 2 1 2 2 2 1 2 1 0 0 1 0 1 1]
```

Acuraccy

In [41]:

```
knn.score(X_test,y_test)
```

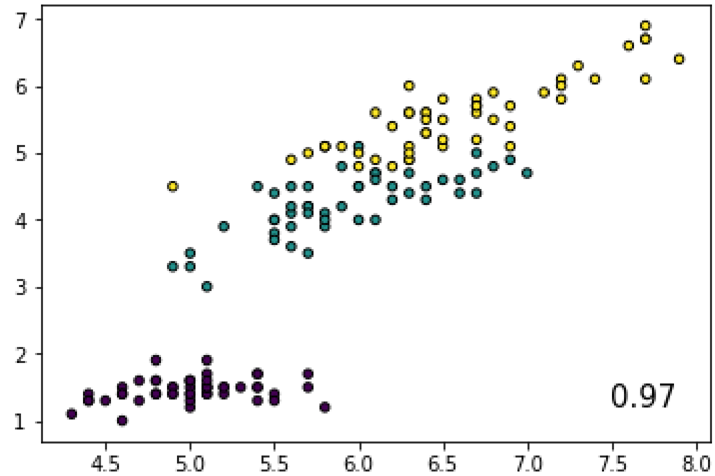
Out[41]:

```
0.9714285714285714
```

Visualization of KNN for iris dataset

In [42]:

```
clf=knn
if True:
    clf.fit(X_train, y_train)
    score = clf.score(X_test, y_test)

    # Plot the decision boundary. For that, we will assign a color to each
    # point in the mesh [x_min, x_max]x[y_min, y_max].
    #Z = clf.predict(X_test)

    # Put the result into a color plot
    #Z = Z.reshape(X_test.shape)
    #plt.figure()
    #plt.pcolormesh(, yy, Z, cmap=cmap_light, alpha=.8)

    # Plot also the training and testing points
    plt.scatter(X[:, 0], X[:, 1], c=y, edgecolor='k', s=20)
    #plt.xlim(xx.min(), xx.max())
    #plt.ylim(yy.min(), yy.max())
    #plt.title("{} (k = {})".format(name, n_neighbors))
    plt.text(0.9, 0.1, '{:.2f}'.format(score), size=15,
             ha='center', va='center', transform=plt.gca().transAxes)

plt.show()
```



KNN for iris dataset using 3 variables

In [6]:

```
import sklearn.neighbors
```

In [12]:

```
import numpy as np
from sklearn import datasets


iris = datasets.load_iris()
iris_data = iris.data
iris_labels = iris.target
print(iris_data[0], iris_data[79], iris_data[100])
print(iris_labels[0], iris_labels[79], iris_labels[100])
```

```
[5.1 3.5 1.4 0.2] [5.7 2.6 3.5 1. ] [6.3 3.3 6.  2.5]
0 1 2
```

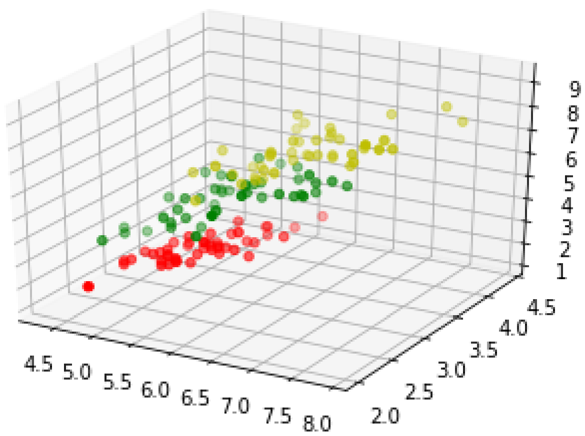
In [13]:

```
np.random.seed(42)
indices = np.random.permutation(len(iris_data))
n_training_samples = 12
learnset_data = iris_data[indices[:-n_training_samples]]
learnset_labels = iris_labels[indices[:-n_training_samples]]
testset_data = iris_data[indices[-n_training_samples:]]
testset_labels = iris_labels[indices[-n_training_samples:]]
print(learnset_data[:4], learnset_labels[:4])
print(testset_data[:4], testset_labels[:4])
```

```
[[6.1 2.8 4.7 1.2]
 [5.7 3.8 1.7 0.3]
 [7.7 2.6 6.9 2.3]
 [6.  2.9 4.5 1.5]] [1 0 2 1]
[[5.7 2.8 4.1 1.3]
 [6.5 3.  5.5 1.8]
 [6.3 2.3 4.4 1.3]
 [6.4 2.9 4.3 1.3]] [1 2 1 1]
```

In [32]:

```
# following line is only necessary, if you use ipython notebook!!!
%matplotlib inline

import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

colours = ("r", "b")
X = []
for iclass in range(3):
    X.append([[], [], []])
    for i in range(len(learnset_data)):
        if learnset_labels[i] == iclass:
            X[iclass][0].append(learnset_data[i][0])
            X[iclass][1].append(learnset_data[i][1])
            X[iclass][2].append(sum(learnset_data[i][2:]))

colours = ("r", "g", "y")

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

for iclass in range(3):
       ax.scatter(X[iclass][0], X[iclass][1], X[iclass][2], c=colours[iclass])
plt.show()
```



Determining the Neighbors

To determine the similarity between two instances, we need a distance function. I am using the Euclidean distance

In [15]:

```
def distance(instance1, instance2):
    # just in case, if the instances are lists or tuples:
    instance1 = np.array(instance1)
    instance2 = np.array(instance2)

    return np.linalg.norm(instance1 - instance2)

print(distance([3, 5], [1, 1]))
print(distance(learnset_data[3], learnset_data[44]))
```

```
4.47213595499958
3.4190641994557516
```

In [16]:

```
def get_neighbors(training_set,
                  labels,
                  test_instance,
                  k,
                  distance=distance):
    """
    get_neighors calculates a list of the k nearest neighbors
    of an instance 'test_instance'.
    The list neighbors contains 3-tuples with
    (index, dist, label)
    where
    index    is the index from the training_set,
    dist     is the distance between the test_instance and the
             instance training_set[index]
    distance is a reference to a function used to calculate the
             distances
    """
    distances = []
    for index in range(len(training_set)):
        dist = distance(test_instance, training_set[index])
        distances.append((training_set[index], dist, labels[index]))
    distances.sort(key=lambda x: x[1])
    neighbors = distances[:k]
    return neighbors
```

Testing the function with taken iris samples

In [17]:

```
for i in range(5):
    neighbors = get_neighbors(learnset_data,
                              learnset_labels,
                              testset_data[i],
                              3,
                              distance=distance)
    print(i,
          testset_data[i],
          testset_labels[i],
          neighbors)
```

```
0 [5.7 2.8 4.1 1.3] 1 [(array([5.7, 2.9, 4.2, 1.3]), 0.141421356237309
95, 1), (array([5.6, 2.7, 4.2, 1.3]), 0.17320508075688815, 1), (array
([5.6, 3. , 4.1, 1.3]), 0.22360679774997935, 1)]
1 [6.5 3.  5.5 1.8] 2 [(array([6.4, 3.1, 5.5, 1.8]), 0.141421356237309
3, 2), (array([6.3, 2.9, 5.6, 1.8]), 0.24494897427831783, 2), (array
([6.5, 3. , 5.2, 2. ]), 0.3605551275463988, 2)]
2 [6.3 2.3 4.4 1.3] 1 [(array([6.2, 2.2, 4.5, 1.5]), 0.264575131106458
6, 1), (array([6.3, 2.5, 4.9, 1.5]), 0.574456264653803, 1), (array([6.
, 2.2, 4. , 1. ]), 0.5916079783099617, 1)]
3 [6.4 2.9 4.3 1.3] 1 [(array([6.2, 2.9, 4.3, 1.3]), 0.200000000000000
18, 1), (array([6.6, 3. , 4.4, 1.4]), 0.2645751311064587, 1), (array
([6.6, 2.9, 4.6, 1.3]), 0.3605551275463984, 1)]
4 [5.6 2.8 4.9 2. ] 2 [(array([5.8, 2.7, 5.1, 1.9]), 0.316227766016837
5, 2), (array([5.8, 2.7, 5.1, 1.9]), 0.3162277660168375, 2), (array
([5.7, 2.5, 5. , 2. ]), 0.33166247903553986, 2)]
```

Voting to get a Single Result

Writing a vote function now. This functions uses the class 'Counter' from collections to count the quantity of the classes inside of an instance list. This instance list will be the neighbors of course. The function 'vote' returns the most common class

In [18]:

```
from collections import Counter

def vote(neighbors):
    class_counter = Counter()
    for neighbor in neighbors:
        class_counter[neighbor[2]] += 1
    return class_counter.most_common(1)[0][0]
```

In [19]:

```
for i in range(n_training_samples):
    neighbors = get_neighbors(learnset_data,
                              learnset_labels,
                              testset_data[i],
                              3,
                              distance=distance)
    print("index: ", i,
          ", result of vote: ", vote(neighbors),
          ", label: ", testset_labels[i],
          ", data: ", testset_data[i])
```

```
index:  0 , result of vote:  1 , label:  1 , data:  [5.7 2.8 4.1 1.3]
index:  1 , result of vote:  2 , label:  2 , data:  [6.5 3.  5.5 1.8]
index:  2 , result of vote:  1 , label:  1 , data:  [6.3 2.3 4.4 1.3]
index:  3 , result of vote:  1 , label:  1 , data:  [6.4 2.9 4.3 1.3]
index:  4 , result of vote:  2 , label:  2 , data:  [5.6 2.8 4.9 2. ]
index:  5 , result of vote:  2 , label:  2 , data:  [5.9 3.  5.1 1.8]
index:  6 , result of vote:  0 , label:  0 , data:  [5.4 3.4 1.7 0.2]
index:  7 , result of vote:  1 , label:  1 , data:  [6.1 2.8 4.  1.3]
index:  8 , result of vote:  1 , label:  2 , data:  [4.9 2.5 4.5 1.7]
index:  9 , result of vote:  0 , label:  0 , data:  [5.8 4.  1.2 0.2]
index:  10 , result of vote:  1 , label:  1 , data:  [5.8 2.6 4.  1.2]
index:  11 , result of vote:  2 , label:  2 , data:  [7.1 3.  5.9 2.1]
```

In [20]:

```
def vote_prob(neighbors):
    class_counter = Counter()
    for neighbor in neighbors:
        class_counter[neighbor[2]] += 1
    labels, votes = zip(*class_counter.most_common())
    winner = class_counter.most_common(1)[0][0]
    votes4winner = class_counter.most_common(1)[0][1]
    return winner, votes4winner/sum(votes)
```

In [21]:

```
for i in range(n_training_samples):
    neighbors = get_neighbors(learnset_data,
                              learnset_labels,
                              testset_data[i],
                              5,
                              distance=distance)
    print("index: ", i,
          ", vote_prob: ", vote_prob(neighbors),
          ", label: ", testset_labels[i],
          ", data: ", testset_data[i])
```

```
index:  0 , vote_prob:  (1, 1.0) , label:  1 , data:  [5.7 2.8 4.1 1.
3]
index:  1 , vote_prob:  (2, 1.0) , label:  2 , data:  [6.5 3.  5.5 1.
8]
index:  2 , vote_prob:  (1, 1.0) , label:  1 , data:  [6.3 2.3 4.4 1.
3]
index:  3 , vote_prob:  (1, 1.0) , label:  1 , data:  [6.4 2.9 4.3 1.
3]
index:  4 , vote_prob:  (2, 1.0) , label:  2 , data:  [5.6 2.8 4.9 2.
]
index:  5 , vote_prob:  (2, 0.8) , label:  2 , data:  [5.9 3.  5.1 1.
8]
index:  6 , vote_prob:  (0, 1.0) , label:  0 , data:  [5.4 3.4 1.7 0.
2]
index:  7 , vote_prob:  (1, 1.0) , label:  1 , data:  [6.1 2.8 4.  1.
3]
index:  8 , vote_prob:  (1, 1.0) , label:  2 , data:  [4.9 2.5 4.5 1.
7]
index:  9 , vote_prob:  (0, 1.0) , label:  0 , data:  [5.8 4.  1.2 0.
2]
index:  10 , vote_prob:  (1, 1.0) , label:  1 , data:  [5.8 2.6 4.  1.
2]
index:  11 , vote_prob:  (2, 1.0) , label:  2 , data:  [7.1 3.  5.9 2.
1]
```

The Weighted Nearest Neighbour Classifier

In [23]:

```
def vote_harmonic_weights(neighbors, all_results=True):
    class_counter = Counter()
    number_of_neighbors = len(neighbors)
    for index in range(number_of_neighbors):
        class_counter[neighbors[index][2]] += 1/(index+1)
    labels, votes = zip(*class_counter.most_common())
    #print(labels, votes)
    winner = class_counter.most_common(1)[0][0]
    votes4winner = class_counter.most_common(1)[0][1]
    if all_results:
        total = sum(class_counter.values(), 0.0)
        for key in class_counter:
            class_counter[key] /= total
        return winner, class_counter.most_common()
    else:
        return winner, votes4winner / sum(votes)
```

In [24]:

```
for i in range(n_training_samples):
    neighbors = get_neighbors(learnset_data,
                              learnset_labels,
                              testset_data[i],
                              6,
                              distance=distance)
    print("index: ", i,
          ", result of vote: ",
          vote_harmonic_weights(neighbors,
                                all_results=True))
```

```
index:  0 , result of vote:  (1, [(1, 1.0)])
index:  1 , result of vote:  (2, [(2, 1.0)])
index:  2 , result of vote:  (1, [(1, 1.0)])
index:  3 , result of vote:  (1, [(1, 1.0)])
index:  4 , result of vote:  (2, [(2, 0.9319727891156463), (1, 0.06802
721088435375)])
index:  5 , result of vote:  (2, [(2, 0.8503401360544217), (1, 0.14965
986394557826)])
index:  6 , result of vote:  (0, [(0, 1.0)])
index:  7 , result of vote:  (1, [(1, 1.0)])
index:  8 , result of vote:  (1, [(1, 1.0)])
index:  9 , result of vote:  (0, [(0, 1.0)])
index:  10 , result of vote:  (1, [(1, 1.0)])
index:  11 , result of vote:  (2, [(2, 1.0)])
```

In [25]:

```
def vote_distance_weights(neighbors, all_results=True):
    class_counter = Counter()
    number_of_neighbors = len(neighbors)
    for index in range(number_of_neighbors):
        dist = neighbors[index][1]
        label = neighbors[index][2]
        class_counter[label] += 1 / (dist**2 + 1)
    labels, votes = zip(*class_counter.most_common())
    #print(labels, votes)
    winner = class_counter.most_common(1)[0][0]
    votes4winner = class_counter.most_common(1)[0][1]
    if all_results:
        total = sum(class_counter.values(), 0.0)
        for key in class_counter:
            class_counter[key] /= total
        return winner, class_counter.most_common()
    else:
        return winner, votes4winner / sum(votes)
```

In [26]:

```
for i in range(n_training_samples):
    neighbors = get_neighbors(learnset_data,
                              learnset_labels,
                              testset_data[i],
                              6,
                              distance=distance)
    print("index: ", i,
          ", result of vote: ", vote_distance_weights(neighbors,
                                                       all_results=True))
```

```
index:  0 , result of vote:  (1, [(1, 1.0)])
index:  1 , result of vote:  (2, [(2, 1.0)])
index:  2 , result of vote:  (1, [(1, 1.0)])
index:  3 , result of vote:  (1, [(1, 1.0)])
index:  4 , result of vote:  (2, [(2, 0.8490154592118361), (1, 0.15098
454078816387)])
index:  5 , result of vote:  (2, [(2, 0.6736137462184478), (1, 0.32638
62537815521)])
index:  6 , result of vote:  (0, [(0, 1.0)])
index:  7 , result of vote:  (1, [(1, 1.0)])
index:  8 , result of vote:  (1, [(1, 1.0)])
index:  9 , result of vote:  (0, [(0, 1.0)])
index:  10 , result of vote:  (1, [(1, 1.0)])
index:  11 , result of vote:  (2, [(2, 1.0)])
```

Another Example for Nearest Neighbor Classification

In [27]:

```
train_set = [(1, 2, 2),
             (-3, -2, 0),
             (1, 1, 3),
             (-3, -3, -1),
             (-3, -2, -0.5),
             (0, 0.3, 0.8),
             (-0.5, 0.6, 0.7),
             (0, 0, 0)
            ]

labels = ['apple',  'banana', 'apple',
          'banana', 'apple', "orange",
          'orange', 'orange']

k = 1
for test_instance in [(0, 0, 0), (2, 2, 2),
                      (-3, -1, 0), (0, 1, 0.9),
                      (1, 1.5, 1.8), (0.9, 0.8, 1.6)]:
    neighbors = get_neighbors(train_set,
                              labels,
                              test_instance,
                              2)

    print("vote distance weights: ", vote_distance_weights(neighbors))
```

```
vote distance weights:  ('orange', [('orange', 1.0)])
vote distance weights:  ('apple', [('apple', 1.0)])
vote distance weights:  ('banana', [('banana', 0.5294117647058824),
('apple', 0.47058823529411764)])
vote distance weights:  ('orange', [('orange', 1.0)])
vote distance weights:  ('apple', [('apple', 1.0)])
vote distance weights:  ('apple', [('apple', 0.5084745762711865), ('or
ange', 0.4915254237288135)])
```

Example with kNN

Use the k-nearest neighbor classifier 'KNeighborsClassifier' from 'sklearn.neighbors' on the Iris data set

In [30]:

```
# Create and fit a nearest-neighbor classifier
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
knn.fit(learnset_data, learnset_labels)
KNeighborsClassifier(algorithm='auto',
                     leaf_size=30,
                     metric='minkowski',
                     metric_params=None,
                     n_jobs=1,
                     n_neighbors=5,
                     p=2,
                     weights='uniform')

print("Predictions form the classifier:")
print(knn.predict(testset_data))
print("Target values:")
print(testset_labels)
```

```
Predictions form the classifier:
[1 2 1 1 2 2 0 1 1 0 1 2]
Target values:
[1 2 1 1 2 2 0 1 2 0 1 2]
```

In [31]:

```
learnset_data[:5], learnset_labels[:5]
```

Out[31]:

```
(array([[6.1, 2.8, 4.7, 1.2],
        [5.7, 3.8, 1.7, 0.3],
        [7.7, 2.6, 6.9, 2.3],
        [6. , 2.9, 4.5, 1.5],
        [6.8, 2.8, 4.8, 1.4]]), array([1, 0, 2, 1, 1]))
```

In [ ]:

```

```