# ID3

In [ ]:

```
import pandas as pd
import numpy as np
import math
```

In [70]:

```
df = pd.read_csv("../WeatherDataset.csv")
```

In [71]:

```
class Node:
    def __init__(self):
        self.children = []
        self.value = ""
        self.isLeaf = False
        self.pred = ""
```

In [72]:

```
features = [ f for f in df  ]
features.remove("answer")
features
```

Out[72]:

```
['outlook', 'temperature', 'humidity', 'wind']
```

In [73]:

```
def entropy(examp):
    pos = 0.0
    neg = 0.0
    for _,row in examp.iterrows():
        if row["answer"] == "yes":
            pos+=1
        else:
            neg+=1
    if pos == 0.0 or neg == 0.0:
        return 0.0
    else:
        p = pos / (pos + neg)
        n = neg / (pos+neg)

        return(-p * math.log(p,2) - n * math.log(n,2))
```

In [82]:

```
def info_gain(examp,attr):
    uniq = np.unique(examp[attr])
    total_gain = entropy(examp)

    for u in uniq:
        subdata = examp[examp[attr] == u]
        entropy_sub = entropy(subdata)
        total_gain -= float(len(subdata)/len(examp)) *entropy_sub
    return total_gain
```

In [90]:

```
def ID3(examp,features):
    root = Node()
    max_gain = 0
    max_feat = ""

    for feature in features:
        gain = info_gain(examp,feature)
        if gain > max_gain:
            max_gain = gain
            max_feat = feature
    root.value = max_feat

    uniq = np.unique(examp[max_feat])
    for u in uniq:
        subdata = examp[examp[max_feat] == u]

        if entropy(subdata) == 0.0:
            leaf = Node()
            leaf.value = u
            leaf.isLeaf = True
            leaf.pred = np.unique(subdata["answer"])[0]
            root.children.append(leaf)
        else:
            dummy = Node()
            dummy.value = u
            new_features = features.copy()
            new_features.remove(max_feat)
            child =ID3(subdata,new_features)
            dummy.children.append(child)
            root.children.append(dummy)
    return root
```

In [91]:

```
def printTree(root : Node , depth = 0 ):
    for i in range(depth):
        print("\t",end="")
    print(root.value,end="")

    if root.isLeaf == True:
        print("->",root.pred)
    print()
    for child in root.children:
        printTree(child,depth+1)
root = ID3(df,features)
printTree(root)
```

```
outlook
        overcast-> YES

        rain
                wind
                        strong-> NO

                        weak-> YES

        sunny
                humidity
                        high-> NO

                        normal-> YES
```

# Candidate Elimination

In [92]:

```
import pandas as pd
import numpy as np
```

In [181]:

```
df = pd.read_csv("../sport.csv")
df.head()
```

Out[181]:

|   | sky | air_temp | humidity | wind | water | forecast | enjoy_sport |
|---|---|---|---|---|---|---|---|
| 0 | sunny | warm | normal | strong | warm | same | yes |
| 1 | sunny | warm | high | strong | warm | same | yes |
| 2 | rainy | cold | high | strong | warm | change | no |
| 3 | sunny | warm | high | strong | cool | change | yes |
| 4 | sunny | warm | normal | strong | warm | same | yes |

In [182]:

```
concepts = np.array(df.iloc[:,:-1])
target = np.array(df.iloc[:,-1])
print(target,concepts)
```

```
['yes' 'yes' 'no' 'yes' 'yes' 'yes' 'no' 'yes'] [['sunny' 'warm' 'normal' 'st
rong' 'warm' 'same']
 ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
 ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
 ['sunny' 'warm' 'high' 'strong' 'cool' 'change']
 ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
 ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
 ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
 ['sunny' 'warm' 'high' 'strong' 'cool' 'change']]
```

In [184]:

```
def Learn(concepts,target):
    specific = concepts[0].copy()
    general = [ ["?" for i in range(len(specific)) ] for i in range(len(specif
ic)) ]


    for i,row in enumerate(concepts):
        if target[i] == "yes":
            for x in range(len(specific)):
                if row[x] != specific[x]:
                    specific[x] = "?"
                    general[x][x] = "?"

        else:
            for x in range(len(specific)):
                if row[x] != specific[x]:
                    general[x][x] = specific[x]
                else:
                    general[x][x] = "?"



    print()
    general = list(filter(lambda val : val != ["?","?","?","?","?","?"] , gene
ral))
#     for i,val in enumerate(general):
#         if val == ["?","?","?","?","?","?"]:
#             general.remove(["?","?","?","?","?","?"])
    return specific,general


s,g=Learn(concepts,target)
print(s,g,sep='\n')
```

```
['sunny' 'warm' '?' 'strong' '?' '?']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]
```

# SVM

In [405]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
```

In [406]:

```
df = pd.read_csv("../svm.csv")
df.head()
```

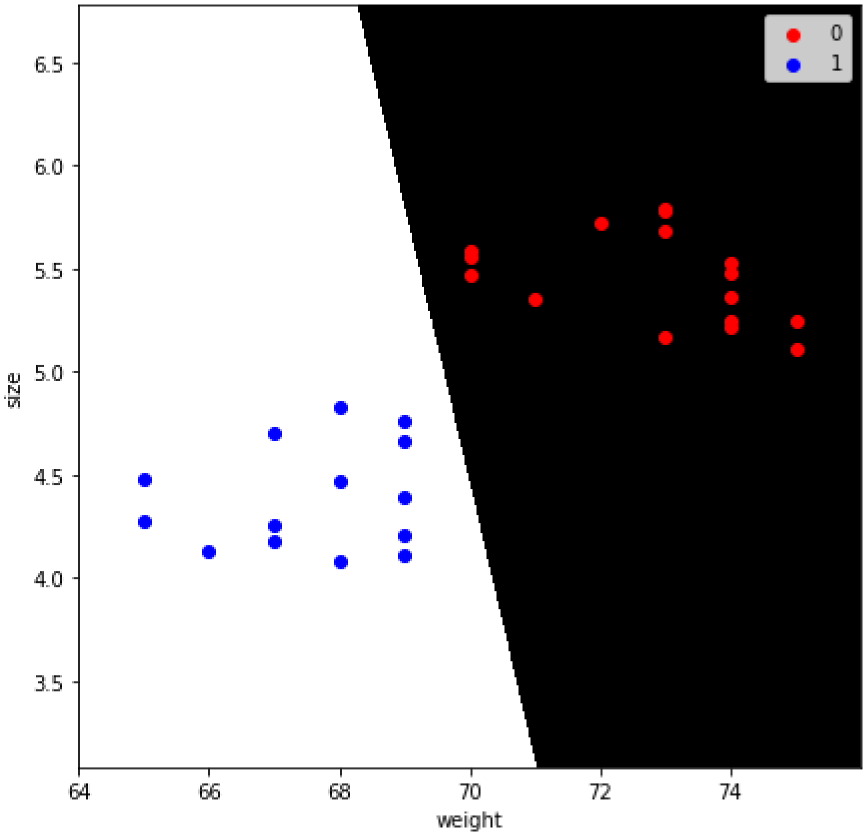
Out[406]:

| | Weight | Size | Class |
|---|---|---|---|
| 0 | 69 | 4.39 | orange |
| 1 | 69 | 4.21 | orange |
| 2 | 65 | 4.09 | orange |
| 3 | 72 | 5.85 | apple |
| 4 | 67 | 4.70 | orange |

In [437]:

```
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix,classification_report
from sklearn.preprocessing import LabelEncoder


X = np.array(df.iloc[:,:-1])
Y  = np.array(df.iloc[:,-1])


xtrain,xtest,ytrain,ytest = train_test_split(X,Y,test_size=0.25,random_state=1
)

print(xtrain,ytrain)
classifier = SVC(kernel='linear',random_state=5,gamma='auto')
classifier.fit(xtrain,ytrain)

ypred = classifier.predict(xtest)

cm = confusion_matrix(ytest,ypred)
print(cm)
accuracy = float(cm.diagonal().sum()) / len(ytest)
print(accuracy)

le = LabelEncoder()
ytrain=le.fit_transform(ytrain)

classifier = SVC(kernel='linear',random_state=5,gamma='auto')
classifier.fit(xtrain,ytrain)

plt.figure(figsize=(7,7))

X1,X2 = np.meshgrid(np.arange(start=xtrain[:,0].min()-1,stop=xtrain[:,0].max()
+1,step=0.01),
                   np.arange(start=xtrain[:,1].min()-1,stop=xtrain[:,1].max()+
1,step=0.01))
plt.contourf(X1,X2,
             classifier.predict(np.array([X1.ravel(),X2.ravel()]).T).reshape(X
1.shape),
                                 cmap=ListedColormap(('black','white')))


for i,j in enumerate(np.unique(ytrain)):
    plt.scatter(xtrain[ytrain == j,0],xtrain[ytrain == j ,1],
                c=ListedColormap(('red','blue'))(i),label=j)

plt.xlim(X1.min(),X1.max())
plt.ylim(X2.min(),X2.max())







plt.xlabel("weight")
plt.ylabel("size")
plt.legend()
plt.show()
```

```
[[72.    5.72]
 [75.    5.25]
 [73.    5.78]
 [69.    4.76]
 [73.    5.17]
 [74.    5.25]
 [67.    4.7 ]
 [74.    5.22]
 [73.    5.79]
 [69.    4.11]
 [68.    4.08]
 [67.    4.25]
 [68.    4.83]
 [66.    4.13]
 [67.    4.18]
 [71.    5.35]
 [70.    5.56]
 [68.    4.47]
 [75.    5.11]
 [70.    5.59]
 [69.    4.21]
 [69.    4.66]
 [69.    4.39]
 [65.    4.48]
 [73.    5.68]
 [70.    5.47]
 [65.    4.27]
 [74.    5.36]
 [74.    5.53]
 [74.    5.48]] ['apple' 'apple' 'apple' 'orange' 'apple' 'apple' 'orange' 'a
pple' 'apple'
 'orange' 'orange' 'orange' 'orange' 'orange' 'orange' 'apple' 'apple'
 'orange' 'apple' 'apple' 'orange' 'orange' 'orange' 'orange' 'apple'
 'apple' 'orange' 'apple' 'apple' 'apple']
[[4 0]
 [1 5]]
0.9

'c' argument looks like a single numeric RGB or RGBA sequence, which should b
e avoided as value-mapping will have precedence in case its length matches wi
th 'x' & 'y'.  Please use a 2-D array with a single row if you really want to
specify the same RGB or RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should b
e avoided as value-mapping will have precedence in case its length matches wi
th 'x' & 'y'.  Please use a 2-D array with a single row if you really want to
specify the same RGB or RGBA value for all points.
```



## Naive Bayes

In [342]:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer
```

In [396]:

```
df = pd.read_csv("../naive.csv",names=["message","tag"])

df.loc[df["tag"] == "pos","tag"] = 1
df.loc[df["tag"] == "neg","tag"] = 0

X = df["message"]
Y = df["tag"]

xtrain,xtest,ytrain,ytest=train_test_split(X, Y,train_size=0.75)

cv = CountVectorizer()
xtrain_t = cv.fit_transform(xtrain)
ytrain_t = ytrain.astype('int')

xtest_t=cv.transform(xtest)
ytest_t = ytest.astype('int')
```

In [397]:

```
# cv.get_feature_names()
```

In [398]:

```
from sklearn.naive_bayes import MultinomialNB

clf = MultinomialNB()

clf.fit(xtrain_t,ytrain_t)
predicted=clf.predict(xtest_t)
print(predicted
,np.array(ytest_t),sep="\n\n")
```

```
[0 0 0 0 1]

[1 0 0 1 1]
```

In [401]:

```
from sklearn.metrics import accuracy_score,confusion_matrix

print(accuracy_score(predicted,ytest_t))
print(confusion_matrix(predicted,ytest_t))
```

```
0.6
[[2 2]
 [0 1]]
```

In [ ]:

In [ ]:

# Svm updated

In [ ]:

```
# # Support Vector Machine
# # Importing the libraries

# import numpy as np
# import matplotlib.pyplot as plt
# import pandas as pd

# # Importing the datasets

# datasets = pd.read_csv('Social_Network_Ads.csv')
# X = datasets.iloc[:, [2,3]].values
# Y = datasets.iloc[:, 4].values

# # Splitting the dataset into the Training set and Test set

# from sklearn.model_selection import train_test_split
# X_Train, X_Test, Y_Train, Y_Test = train_test_split(X, Y, test_size = 0.25,
 random_state = 0)

# # Feature Scaling

# from sklearn.preprocessing import StandardScaler
# sc_X = StandardScaler()
# X_Train = sc_X.fit_transform(X_Train)
# X_Test = sc_X.transform(X_Test)

# # Fitting the classifier into the Training set

# from sklearn.svm import SVC
# classifier = SVC(kernel = 'linear', random_state = 0)
# classifier.fit(X_Train, Y_Train)

# # Predicting the test set results

# Y_Pred = classifier.predict(X_Test)

# # Making the Confusion Matrix

# from sklearn.metrics import confusion_matrix
# cm = confusion_matrix(Y_Test, Y_Pred)

# # Visualising the Training set results

# from matplotlib.colors import ListedColormap
# X_Set, Y_Set = X_Train, Y_Train
# X1, X2 = np.meshgrid(np.arange(start = X_Set[:, 0].min() - 1, stop = X_Set
[:, 0].max() + 1, step = 0.01),
#                      np.arange(start = X_Set[:, 1].min() - 1, stop = X_Set
[:, 1].max() + 1, step = 0.01))
# plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).
T).reshape(X1.shape),
#              alpha = 0.75, cmap = ListedColormap(('red', 'green')))
# plt.xlim(X1.min(), X1.max())
# plt.ylim(X2.min(), X2.max())
# for i, j in enumerate(np.unique(Y_Set)):
#     plt.scatter(X_Set[Y_Set == j, 0], X_Set[Y_Set == j, 1],
#                  c = ListedColormap(('red', 'green'))(i), label = j)
# plt.title('Support Vector Machine (Training set)')
# plt.xlabel('Age')
# plt.ylabel('Estimated Salary')
# plt.legend()
# plt.show()

# # Visualising the Test set results

# from matplotlib.colors import ListedColormap
# X_Set, Y_Set = X_Test, Y_Test
# X1, X2 = np.meshgrid(np.arange(start = X_Set[:, 0].min() - 1, stop = X_Set[:, 0].max() + 1, step = 0.01),
#                       np.arange(start = X_Set[:, 1].min() - 1, stop = X_Set[:, 1].max() + 1, step = 0.01))
# plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
#              alpha = 0.75, cmap = ListedColormap(('red', 'green')))
# plt.xlim(X1.min(), X1.max())
# plt.ylim(X2.min(), X2.max())
# for i, j in enumerate(np.unique(Y_Set)):
#     plt.scatter(X_Set[Y_Set == j, 0], X_Set[Y_Set == j, 1],
#                  c = ListedColormap(('red', 'green'))(i), label = j)
# plt.title('Support Vector Machine (Test set)')
# plt.xlabel('Age')
# plt.ylabel('Estimated Salary')
# plt.legend()
# plt.show()
```

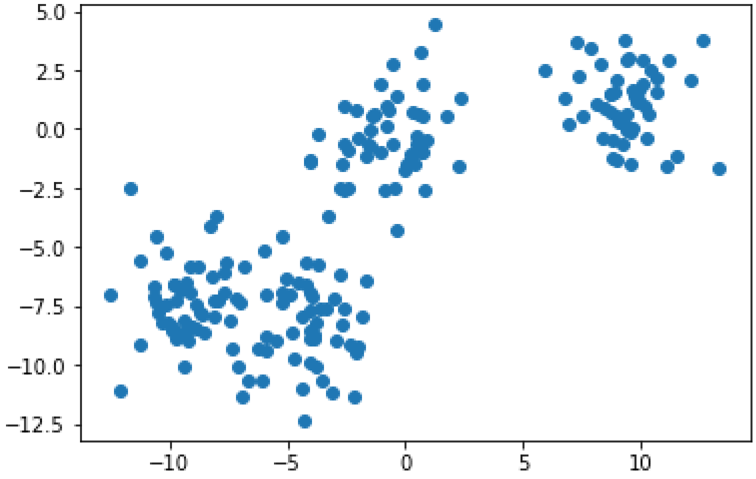
In [ ]:

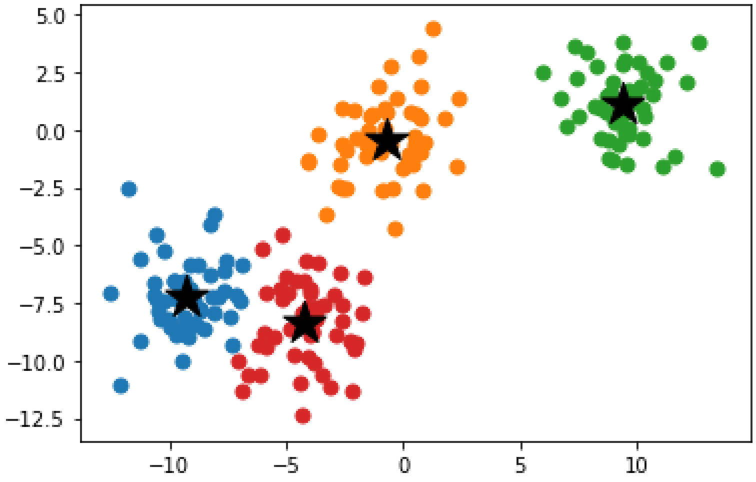
# Clustering

In [463]:

```
#Import The libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans

#Generate our dataset
dataset=make_blobs(n_samples=200,
                   centers=4,
                   n_features=2,
                   cluster_std=1.6,
                    random_state=45)

points=dataset[0]
print(points[:1])

#kmeans and fitting the data
kmeans=KMeans(n_clusters=4)
kmeans.fit(points)

plt.scatter(points[:,0],points[:,1])
plt.show()

clusters=kmeans.cluster_centers_

y_ks=kmeans.fit_predict(points)

for i,j in enumerate(y_ks):
    plt.scatter(points[y_ks == i,0],points[y_ks==i,1],s=50)

# plt.scatter(points[y_ks == 0,0],points[y_ks == 0,1],s=50,color='red')
# plt.scatter(points[y_ks == 1,0],points[y_ks == 1,1],s=50,color='green')
# plt.scatter(points[y_ks == 2,0],points[y_ks == 2,1],s=50,color='yellow')
# plt.scatter(points[y_ks == 3,0],points[y_ks == 3,1],s=50,color='cyan')

plt.scatter(clusters[0][0],clusters[0][1],marker='*',s=500,color="black")
plt.scatter(clusters[1][0],clusters[1][1],marker='*',s=500,color="black")
plt.scatter(clusters[2][0],clusters[2][1],marker='*',s=500,color="black")
plt.scatter(clusters[3][0],clusters[3][1],marker='*',s=500,color="black")
plt.show()
```

```
[[10.21510966  0.98359586]]
```





In [ ]:

In [ ]:

# Knn

In [464]:

```python
from sklearn.model_selection import train_test_split
from sklearn import datasets
from sklearn.neighbors import KNeighborsClassifier as knn
from sklearn.metrics import classification_report, confusion_matrix

iris = datasets.load_iris()
x = iris.data
y = iris.target

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.1)

print("Size of training data and its label", x_train.shape, y_train.shape)

for i in range(len(iris.target_names)):
    print("Label", i, "-", str(iris.target_names[i]))

classifier = knn(n_neighbors=1)
classifier.fit(x_train, y_train)
y_pred = classifier.predict(x_test)
print("Results of Classification using K-nn with K=1")
print("Sample\t\t\tActual-label\tpredicted-label")
for r in range(0,len(x_test)//4):
    print(str(x_test[r]), "\t", str(y_test[r]), "\t\t\t",str(y_pred[r]))
print("Classification Accuracy:", classifier.score(x_test, y_test))

print('Confusion Matrix')
print(confusion_matrix(y_test,y_pred))
print('Accuracy Metrics')
print(classification_report(y_test,y_pred))
```

```
Size of training data and its label (135, 4) (135,)
Label 0 - setosa
Label 1 - versicolor
Label 2 - virginica
Results of Classification using K-nn with K=1
Sample                  Actual-label    predicted-label
[7.9 3.8 6.4 2. ]        2                       2
[4.7 3.2 1.3 0.2]        0                       0
[4.8 3.4 1.9 0.2]        0                       0
Classification Accuracy: 1.0
Confusion Matrix
[[8 0 0]
 [0 3 0]
 [0 0 4]]
Accuracy Metrics
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         8
           1       1.00      1.00      1.00         3
           2       1.00      1.00      1.00         4

    accuracy                           1.00        15
   macro avg       1.00      1.00      1.00        15
weighted avg       1.00      1.00      1.00        15
```

In [ ]: