**Pseudo code for naïve bayes:**

**Calculate diagnosis=“yes”, diagnosis=“no” probabilities Pyes, Pno from training input.**

**For Each Test Input Record**

**For Each Attribute Calculate Category of Attribute Based On Categorical Division**

**Calculate Probabilities Of Diagnosis=“Yes”, Diagnosis=“No” Corresponds To That Category P(Attr,Yes), P(Attr,No) From Training Input .**

**For Each Attribute**

**Calculate The Resultyes= Resultyes\* P(Attr,Yes),Resultno= Resultno\*P(Attr,No);**

**Calculate Resultyes= Resultyes \*Pyes**

**Resultno= Resultno\*Pno;**

**If(Resultyes > Resultno) Then Diagnosis=“Yes”; Else Then Diagnosis =“No”;**

**Formulae for naïve bayes : **

**Pyes=total number of yes/total number of records;  Pno=total number of no/total number of records;  P(attr,yes)=total number of yes in corresponding category/total number of yes;**

** P(attr,no)=total number of yes in corresponding category/total number of yes;**

**Pseudo code for weighted association classifiers:**

**Calculate Frequent itemsets from the items by using apriori algorithm;**

**For each frequent itemset**

**{ Calculate W\_support(frequent itemset );**

**Calculate W\_confidence(frequent itemset**

**->diagnosis\_yes);**

**If(w\_confidence >min\_confidence)**

**Store it in the rulebase\_yes with their w\_supports,w\_confidences;**

**Calculate W\_confidence(frequent itemset ->diagnosis\_no);**

**If(w\_confidence >min\_confidence)**

**Store it in the rulebase\_no with their w\_supports,w\_confidences;**

**}**

**Formulae for weighted association classifiers:**

**Recordweight= summation of weights of the items /total number of items present;**

**Total weight= summation of the Recordweights of all records**

**W\_support(X) =summation of the Record weights of all records which contains X /Totalweight;**

**W\_confidence(X->Y)=W\_support(XUY)/W\_support(X);**

**Pseudo-code for Apriori algorithm:**

**Ck: Candidate itemset of size k**

**Lk : frequent itemset of size k**

**L1 = {frequent items};**

**for (k = 1; Lk !=null; k++) do begin**

**Ck+1 = candidates generated from Lk;**

**Ck+1 = candidates generated from Lk;**

**Ck+1 = candidates generated from Lk;**

**Ck+1 = candidates generated from Lk;**

**Ck+1 = candidates generated from Lk;**

**For each transaction t in database do increment the count of all candidates in Ck+1 that are contained in t Lk+1 = candidates in Ck+1 with min\_support end return U Lk;**