1 Gabriel Graph Classifier

In this example we will create two multivariate Gaussian distributions, then the structure of the Gabriel Graph is used to find samples within the margin between the two distributions.

1. Include the following libraries.

```
> library("spdep")
> library("MASS")
> library("prettyGraphs")
> library("igraph")
> library("cccd")
> # library('mvtnorm')
```

2. Creating two multivariate Gaussian distributions.

```
> d=2
> npontos=30
> m1<-3
> sd1<-0.5
> cov1<-diag(sd1,d)
> X1<-mvrnorm(n=npontos, rep(m1,d), cov1)
> m2<-6
> sd2<-0.5
> cov2<-diag(sd2,d)
> X2<-mvrnorm(n=npontos, rep(m2,d), cov2)
> # m1<-matrix(c(2,2),byrow = T, ncol=1)
> # m2<-matrix(c(6,6),byrow = T, ncol=1)
> #
> # N1<-30
> # N2<-30
> # r<-1
> #
> # X1<-matrix(rnorm(2*N1),ncol = 2)+matrix(m1,ncol = 2,nrow=N1,byrow = T)
> # X2<-matrix(rnorm(2*N2),ncol = 2)+matrix(m2,ncol = 2,nrow=N2,byrow = T)
>
```

3. Plotting distributions.

```
> par(pty="s",bty="n")
> plot(X1[,1],X1[,2],xlim=c(0,9),ylim=c(0,9),xlab="X1",ylab="X2",
+      col="blue",pch=20)
> points(X2[,1],X2[,2],col="red",pch=20)
```

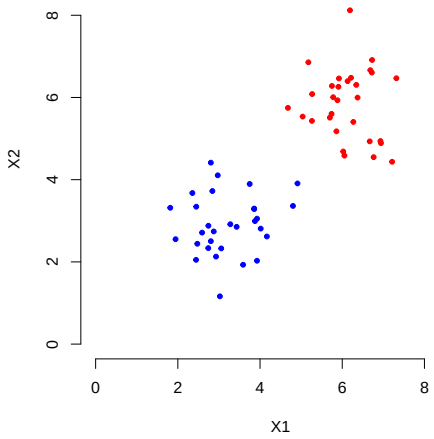

Figure 1: Two data sets.

4. Setting the data structure.

```
> X<-rbind(X1,X2) # Join the two sets.
> D<-c(rep(1,nrow(X1)),rep(-1,nrow(X2))) # Include labels of each data set.
```

5. Finding Gabriel Graph.

```
> G <-gg(X,method = "Euclidean") # Gabriel Graph Structure
```

6. Extracting adjacency matrix (M) from Gabriel graph.

```
> M<-as.matrix(get.adjacency(G))
> rownames(M)<-c(1:nrow(M))
> colnames(M)<-c(1:nrow(M))
```

7. Plotting Gabriel Graph.

```
> V(G)$color <- c(rep("red",30),rep("blue",30))
> # Setting different colors for each class.
> plot(G,layout=X,vertex.color=V(G)$color,
+      vertex.size=4,vertex.label=NA,axes=TRUE)
```

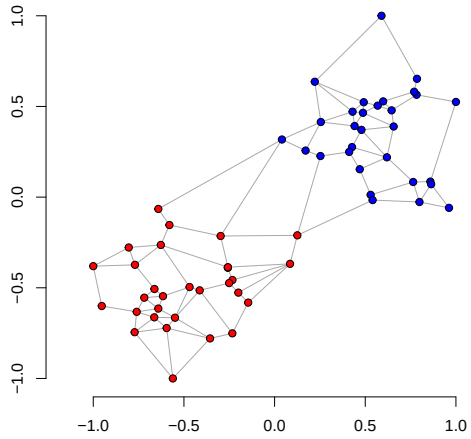

Figure 2: Gabriel Graph of two data sets.

8. Finding the boundary.

```
> D<-as.matrix(D)
> D<-t(D)
> mask<-repmat(D,ncol(D),1)
> mask2<-mask*M
> borda <- matrix(rep(0,nrow(X)),nrow = nrow(X),ncol=1)
> for (i in c(1:nrow(X))){
+   if (sum(-D[i] == mask2[i,])>0){
+     borda[i,1]<-1;
+   }
+ }
```

9. Plotting the boundary.

```
> N<-nrow(X)
> mm<-matrix(1,N,N) - diag(N)
> par(mfrow=c(1,1))
> g <- graph_from_adjacency_matrix(mm,mode="undirected")
> plot(G,layout=X,vertex.color=V(G)$color,vertex.size=4,
+      vertex.label=NA,axes=TRUE,mark.group=which(borda!=0),xlab = "x", ylab = "y")
```

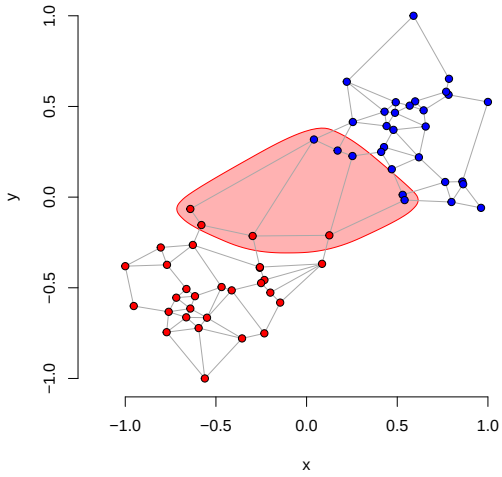

Figure 3: Boundary of Gabriel Graph.

1.1 Database with overlap.

If the distribution has overlapping between classes, as shown in Figure 4, is necessary to remove some data to find the edge.

1. Plotting two classes with overlap.

```
> d<-2
> npontos1<-50
> npontos2<-50
> m1<-4
> sd1<-0.9
> cov1<-diag(sd1,d)
> X1 <- mvrnorm(n=npontos1, rep(m1,d), cov1)
> m2<-6
> sd2<-0.9
> cov2<-diag(sd2,d)
> X2<-mvrnorm(n=npontos2, rep(m2,d), cov2)
> par(pty='s', bty='n')
> plot(X1[,1],X1[,2],xlim=c(0,9),ylim=c(0,9),xlab="X1",ylab="X2",col="blue",pch=20)
> points(X2[,1],X2[,2],col="red",pch=20)
> X<-rbind(X1,X2)
> D<- c(rep(1,nrow(X1)), rep(-1, nrow(X2)))
```

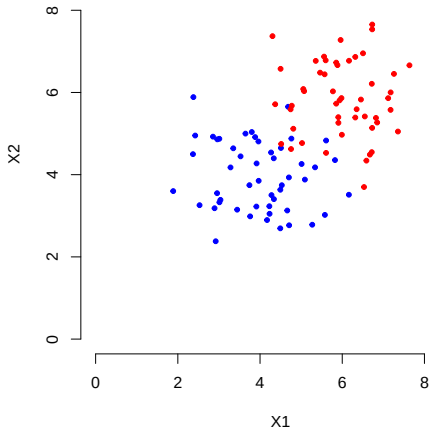

Figure 4: Two classes with overlap.

2. Finding Gabriel Graph.

```
> col.gab.nb<-graph2nb(gabrielneigh(X), sym=TRUE) # Gabriel Graph Structure
```

3. Extracting adjacency matrix (M) from Gabriel graph.

```
> M <- as(nb2listw(col.gab.nb), "CsparseMatrix")
> M <- as.matrix(M)
> M[M>0]<-1
```

4. Plotting Gabriel Graph.

```
> G <- graph_from_adjacency_matrix(M, mode="undirected")
> V(G)$color <- c(rep("red",50),rep("blue",50))
> # Setting different colors for each class.
> plot(G,layout=X,vertex.color=V(G)$color,
+      vertex.size=4,vertex.label=NA,axes=TRUE)
```

5. Finding overlapping samples.

```
> ## Finding quality coefficient Q.
> A1 <- colSums(M[,D==1])
> A2 <- colSums(M[,D==-1])
> A1h <- colSums(M[D==1,D==1])
> A2h <- colSums(M[D==-1,D==-1])
> Q.c1 <- A1h/A1
> Q.c2 <- A2h/A2
> ## Calculate the value of t threshold.
> t.c1 <- ( sum(Q.c1) / length(Q.c1) )
> t.c2 <- ( sum(Q.c2) / length(Q.c2) )
> ##  Data to be removed.
```

```
> retira.c1 <- which(Q.c1 < t.c1)
> retira.c2 <- which(Q.c2 < t.c2)
> retira.c1<-as.numeric(names(retira.c1))
> retira.c2<-as.numeric(names(retira.c2))
> retira.c1c2 <- union(retira.c1, retira.c2)
>
```

6. Plotting overlap (green data).

```
> V(G)$color <- c(rep("red",nrow(X1)),rep("blue",nrow(X2))) # Colocando uma cor difer
> V(G)$color[retira.c1c2]<-"green"
> plot(G,layout=X,vertex.color=V(G)$color,vertex.size=4,vertex.label=NA,axes=TRUE)
```

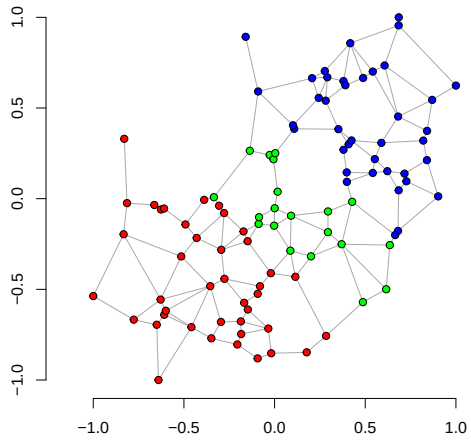

Figure 5: Two classes with overlap.

7. Removing overlap.

```
> X<-X[-retira.c1c2,]
> D<-D[-retira.c1c2]
```

8. Building new graph without overlapping.

```
> # Finding Gabriel Graph.
>
> G <-gg(X,method = "Euclidean") # Gabriel Graph Structure
> # Extracting adjacency matrix $(M)$ from Gabriel graph.
>
> M<-as.matrix(get.adjacency(G))
> graphcolors<-matrix(nrow=nrow(X), ncol=1)
> graphcolors[D==1,1]<-"red"
> graphcolors[D==-1,1]<-"blue"
```

```
> V(G)$color <- graphcolors
> # Setting different colors for each class.
> plot(G,layout=X,vertex.color=V(G)$color,
+      vertex.size=4,vertex.label=NA,axes=TRUE)
```

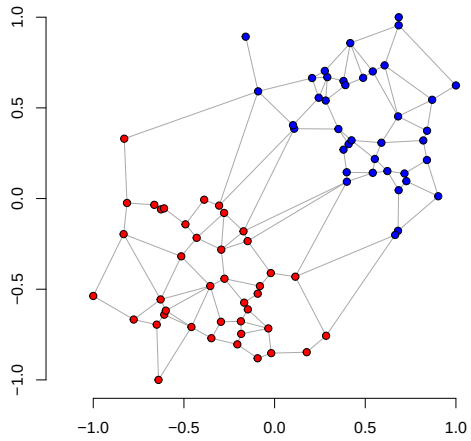

Figure 6: New graph without overlapping.

9. Finding the new boundary.

```
> D<-as.matrix(D)
> D<-t(D)
> mask<-repmat(D,ncol(D),1)
> mask2<-mask*M
> borda <- matrix(rep(0,nrow(X)),nrow = nrow(X),ncol=1)
> for (i in c(1:nrow(X))){
+   if (sum(-D[i] == mask2[i,])>0){
+     borda[i,1]<-1;
+   }
+ }
```

10. Plotting the boundary, i.e the support edges (orange points).

```
> N<-nrow(X)
> mm<-matrix(1,N,N) - diag(N)
> par(mfrow=c(1,1))
> g <- graph_from_adjacency_matrix(mm,mode="undirected")
> graphcolors[borda==1,1]<-"orange"
> V(G)$color <- graphcolors
> plot(G,layout=X,vertex.color=V(G)$color,vertex.size=4,
+      vertex.label=NA,axes=TRUE)
```

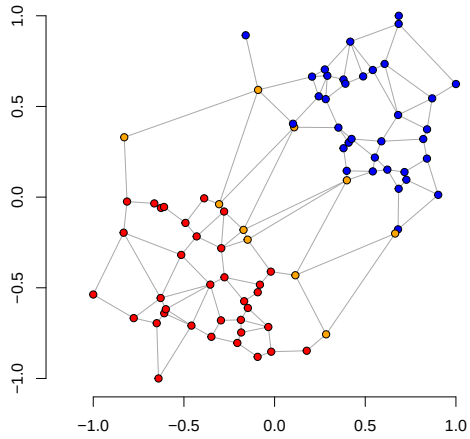

Figure 7: Boundary of Gabriel Graph.

1.2 Classification

1. Generating middle points.

```
> # Normalizing data between -1 and 1.
> X<-((X-mean(X))/max(abs(X-mean(X))))
> maskD<-repmat(D,ncol(D),1)
> maskborda<-repmat(borda==1,1,length(borda));
```

2. Filtering only the rows that match the margin.

```
> aux<-maskborda*t(maskborda)
```

3. Removing edges that do not belong to the graph.

```
> aux<-M*aux
```

4. Discards edges between vertices of the same class.

```
> aux1<-aux + (maskD*aux)
> aux2<-aux - (maskD*aux)
> aux<-aux1*t(aux2)
```

5. Transform matrix in binary.

```
> aux<-(aux!=0)
> arestas<-which(aux==1, arr.ind = TRUE)
```

6. Generating middle points.

```
> pm <- (X[arestas[,1],]+X[arestas[,2],])/2
> edge.C <- X[arestas[,1],]
> edge.D <- X[arestas[,2],]
> edge.Sinal <- t(D[arestas[,1]])
```

7. Plotting middle points (black points).

```
> G <- G + vertex(c(1:nrow(pm)))
> V(G)$color[c((nrow(X)+1):(nrow(pm)+nrow(X)))]<-"black"
> X3 <- rbind(X,pm)
> plot(G,layout=X3,vertex.color=V(G)$color,vertex.size=4,
+      vertex.label=NA,axes=TRUE)
```

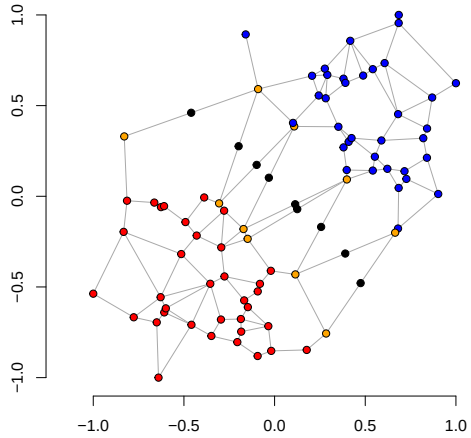

Figure 8: Middle Points.

8. Finding weights.

```
> W <- matrix(0,nro = nrow(pm), ncol = ncol(pm))
> Bias <- as.matrix(rep(0,nrow(pm)))
> Sinal <- as.matrix(rep(0,nrow(pm)))
> for (i in c(1:nrow(pm))){
+   c<-edge.C[i,]
+   d<-edge.D[i,]
+   m <- (c+d)/2;
+   w<-c-d;
+   W[i,]<-w;
+   Bias[i] <- m%*%w
+   Sinal[i]<-D[arestas[i,1]]
+ }
```

9. Classifying data grid.

```
> seqxy <- seq(-1,1,by = 0.01)
> Z<-matrix(nrow = length(seqxy),ncol = length(seqxy))
> distancias <- matrix(rep(0,nrow(Bias)))
> Class <- matrix(rep(0,nrow(Bias)))
```

```
> for (k in c(1:length(seqxy))){
+   for (j in c(1:length(seqxy))){
+     x<-as.matrix(c(seqxy[k],seqxy[j]))
+     x<-t(x)
+     for (i in c(1:nrow(pm))){
+       w <- W[i,]
+       b <- Bias[i]
+       sinal <- Sinal[i]
+       Decisor <- w%*%t(x)
+       Decisor <- Decisor - b
+
+       if (Decisor>0){
+         class<-sinal
+         if (class == -1){
+           class <- -1
+         }else{
+           class <- 1
+         }
+
+       }else{
+         class <- -sinal
+         if (class == -1){
+           class <- -1
+         }else{
+           class <- 1
+         }
+       }
+
+       d<-dist(rbind(x,pm[i,]))
+       distancias[i,]<-d
+       Class[i,] <- class*d
+     }
+
+     s<-sign(Class)
+     menor <- which.min(distancias)
+     Class <- 1 / abs(Class)
+     Class <- (Class)/sum((Class))
+     Class <- sign(Class)*s
+
+     if (Class[menor,1]>0){
+       Y<-1
+     }else{
+       Y<- -1
+     }
+     Z[k,j]<-Y
+   }
+ }
```

10. Generating surface.

```
>  par(pty='s', bty='n')
```

```
>  plot(X[D==1,1],X[D==1,2],xlim=c(-1,1),ylim=c(-1,1),
+       xlab="X1",ylab="X2",col="blue",pch=20)
>  points(X[D==-1,1],X[D==-1,2],col="red",pch=20)
> contour(seqxy,seqxy,Z,
+         nlevels = 1,
+         labels = NULL,
+         xlim = range(seqxy, finite = TRUE),
+         ylim = range(seqxy, finite = TRUE),
+         zlim = range(Z, finite = TRUE),
+         labcex = 0.6, drawlabels = FALSE, method = "flattest", frame.plot = axes,
+         col = par("fg"), lty = par("lty"), lwd = par("lwd"),
+         add = TRUE)
```

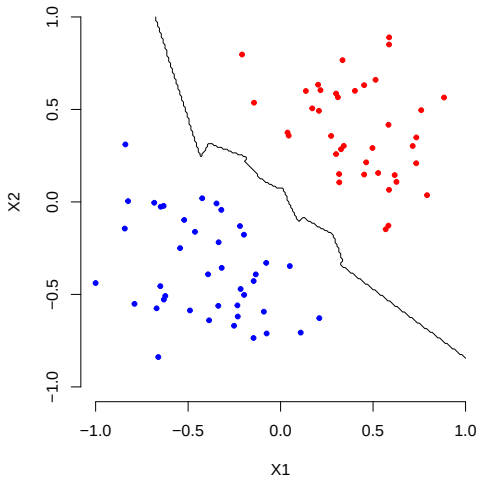

Figure 9: Hyperplane.