APRENDIZAJE AUTOMÁTICO

Entregable T4: Máquinas vectores soporte

Victoria Beltrán Domínguez

vicbeldo@inf.upv.es

Grupo computación: 4CO11

Universitat Politécnica de Valencia

Índice de contenidos

1. Introducción

En el problema de la clasificación en C clases con SVM, se nos pide comparar las técnicas basadas en votación y DAGs utilizando SVMs de dos clases con el subconjunto de los datos utilizados en clase de prácticas para las clases 0 a 3.

Para ello, primero se comentarán los pasos previos comunes a ambos clasificadores, seguidamente se pasará a explicar un pseudocódigo de la implementación de ambos clasificadores y finalmente, se expondrán algunas conclusiones.

2. Pasos previos a la implementación de los clasificadores

Para la implementación de ambos clasificadores necesitamos primero cargar los datos en octave. Trabajamos con el conjunto de datos MNIST y por ello, tenemos 4 conjuntos de datos: X (entrenamiento), xl (etiquetas de entrenamiento), Y(test) y yl (etiquetas de test). Como el conjunto de datos MNIST tiene 9 clases, deberemos de guardarnos sólo las 4 primeras (0...3). Así, para el caso de testing, ambos clasificadores van a separar las 4 primeras clases al principio de su implementación:

```
te=Y(find(yl==0|yl==1|yl==2|yl==3),:);

telabels=yl(find(yl==0|yl==1|yl==2|yl==3),:);
```

Para las muestras de entrenamiento, iremos seleccionando poco a poco las que necesitamos en cada caso al entrenar los diferentes clasificadores. Así, para cada combinación de clases que hay que entrenar, vamos a realizar los siguientes pasos:

Primero seleccionamos los índices de las clases que nos interesan:

```
indicesclasexclasey=find(xl==clasex | xl==clasey);
```

Seguidamente, nos guardamos las muestras que nos interesan y sus correspondientes clases:

```
trlabels_clasexclasey=xl(indicesclasexclasey);

tr_clasexclasey=X(indicesclasexclasey,:);
```

Finalmente, entrenamos un clasificador con esas dos clases.

```
resclasexclasey=svmtrain(trlabels_clasexclasey,  tr_clasexclasey,'-c x -t y -q');
```

Cabe destacar que estos pasos deberemos repetirlos para todas las combinaciones de clases (01,02,03,12,13,23), y así obtendremos 6 clasificadores de una clase contra otra.

Para saber los valores adecuados para los parámetros *x* e *y*, deberemos probar para ver el error que producen. Esta prueba se realizará una vez tengamos implementados los clasificadores, y por lo tanto se mostrarán los resultados de esta en las conclusiones.

3. Implementación de los clasificadores

A continuación, pasamos a explicar los detalles de cada uno de los clasificadores, una vez hemos realizado los pasos anteriormente explicados. Cabe destacar que a partir de este punto, tenemos ya todos los clasificadores entrenados y la clase de testing te con sus respectivas etiquetas telabels. Utilizaremos resclasexclasey para hacer referencia al clasificador de la clase x contra la clase y.

3.1 Implementación en octave del clasificador por votación

Seguiremos los siguientes pasos para implementar el clasificador por votación:

1. Para cada clasificador diferente(6 en total), clasificamos todas las muestras de testing

```
clasificacionclasexclasey=svmpredict(telabels,te,resclasexclasey,'-q');
```

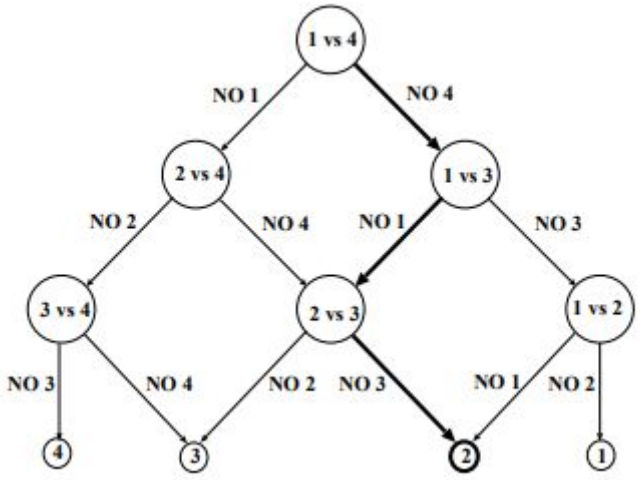
2. Contamos para cada muestra cuántos votos ha recibido de cada clase y lo guardamos en un vector

3. En esta particular implementación, utilizamos la función max, para ver qué clase tiene el mayor número de votos. Así, en caso de empate el comportamiento será determinista y escogerá la clase con menor valor numérico (por ejemplo, si hay empate entre la clase 0 y la clase 3, escogerá siempre la 0).

4. Observamos la tasa de aciertos, comparando la clasificación obtenida con telabels previamente calculada y calculamos los resultados.

3.2 Implementación en octave del clasificador DAG

Seguiremos los siguientes pasos para implementar el clasificador DAG:



Siguiendo el esquema del ejercicio 10 del tema 4 en el boletín de ejercicios, iremos partiendo los datos poco a poco hasta llegar a clasificarlos en una clase según el grafo. Una vez todos han pasado por el grafo y se han clasificado, compararemos los resultados obtenidos con los reales y obtendremos la tasa de aciertos.

Como este clasificador es complicado de explicar con pseudocódigo genérico, se ha explicado el funcionamiento general del clasificador. Para más detalles de implementación consultar el apéndice.

4. Conclusiones

Una vez tenemos implementados ambos clasificadores, sólo necesitamos probar valores para diferentes kernels y de C. Probando diferentes valores de C nos damos cuenta de que no hay diferencias significativas entre ellas, pero que sin embargo, al probar diferentes kernels el error cambia significativamente. La siguiente tabla muestra el porcentaje de aciertos obtenido en cada clasificador dependiendo del kernel utilizado:

Kernel	Votación (% aciertos)	DAG (% aciertos)
Lineal c=0.1	92.470	93.721
Lineal c=1	92.470	93.721

Lineal c=1000	92.470	93.721
Polinomial c=0.1	99.35	99.326
Polinomial c=1	99.35	99.326
Polinomial c=1000	99.35	99.326
Radial c=0.1	27.303	27.303
Radial c=1	27.303	27.303
Radial c=1000	27.303	27.303
Sigmoid c=0.1	27.303	27.303
Sigmoid c=1	27.303	27.303
Sigmoid c=1000	27.303	27.303

Así, para este determinado conjunto de datos (MNIST), el único kernel que realmente se acerca a resultados relevantes es el kernel polinomial. El resto además de tardar un tiempo considerablemente alto, dan peores resultados.

Además, una vez sabemos que el kernel más importante es polinomial, vamos a probar con diferentes grados para ver cuál es el más apropiado. La siguiente tabla muestra el porcentaje de acierto del kernel polinomial probando con diferentes grados:

Grado del kernel polinomial d	Votación (% aciertos)	DAG (% aciertos)
d=2	99.375	99.375
d=3	99.35	99.326
d=4	99.038	98.966
d=5	98.533	98.509
d=8	96.271	96.271

Así, vemos que para este problema, el grado 2 es el que mejor porcentajes obtiene. Con este grado del polinomio y para este problema particular, la tasa de aciertos que obtiene el clasificador por votación es igual que el clasificador DAG (99.375%). Además, midiendo el tiempo que tarda cada uno de los clasificadores, para MNIST

DAG suele ser un poco más rápido que el clasificador por votación, pero para este problema la diferencia no es realmente relevante.

En conclusión, para este conjunto de datos, tanto el clasificador por votación como el clasificador DAG obtienen resultados similares en tiempos similares.

5. Apéndice

4.1 Implementación en octave del clasificador por votación

```
#!/usr/bin/octave -qf

function []=vote(X,xl,Y,yl)

%Nos guardamos el tiempo para poder comparar con DAG

t0 = clock ();

tr=X(find(xl==0|xl==1|xl==2|xl==3),:);

te=Y(find(yl==0|yl==1|yl==2|yl==3),:);

telabels=yl(find(yl==0|yl==1|yl==2|yl==3),:);


%Clase 0 contra 1

indices01=find(xl==0|xl==1);

trlabels01=xl(indices01);

tr01=X(indices01,:);

res01=svmtrain(trlabels01,tr01,'-c 1000 -t 1 -q');

clas01=svmpredict(telabels,te,res01,'-q');


%Clase 0 contra 2

indices02=find(xl==0|xl==2);

trlabels02=xl(indices02);
```

```
tr02=X(indices02,:);
res02=svmtrain(trlabels02,tr02,'-c 1000 -t 1 -d 2 -q');
clas02=svmpredict(telabels,te,res02,'-q');

%Clase 0 contra 3
indices03=find(xl==0|xl==3);
trlabels03=xl(indices03);
tr03=X(indices03,:);
res03=svmtrain(trlabels03,tr03,'-c 1000 -t 1 -d 2 -q');
clas03=svmpredict(telabels,te,res03,'-q');

%Clase 1 contra 2
indices12=find(xl==1|xl==2);
trlabels12=xl(indices12);
tr12=X(indices12,:);
res12=svmtrain(trlabels12,tr12,'-c 1000 -t 1 -d 2 -q');
clas12=svmpredict(telabels,te,res12,'-q');

%Clase 1 contra 3
indices13=find(xl==1|xl==3);
trlabels13=xl(indices13);
tr13=X(indices13,:);
res13=svmtrain(trlabels13,tr13,'-c 1000 -t 1 -d 2 -q');
clas13=svmpredict(telabels,te,res13,'-q');
```

```
%Clase 2 contra 3

indices23=find(xl==2|xl==3);

trlabels23=xl(indices23);

tr23=X(indices23,:);

res23=svmtrain(trlabels23,tr23,'-c 1000 -t 1 -d 2 -q');

clas23=svmpredict(telabels,te,res23,'-q');


%Clasificar muestras de test

vector_oficial=[clas01 clas02 clas03 clas12 clas13 clas23];

k=rows(vector_oficial);

clasification=[];

for i=1:k

  votacion=vector_oficial(i,:);

  votacion0=sum(votacion==0);

  votacion1=sum(votacion==1);

  votacion2=sum(votacion==2);

  votacion3=sum(votacion==3);

  [_,ganador]=max([votacion0 votacion1 votacion2 votacion3]);

  clasification(i)=ganador-1;


endfor


%Testear y sacar el error
```

```
results=(telabels==clasification');

tasa_aciertos=sum(results)/rows(results)*100;

printf("Tasa de aciertos por votacion: %.3f\n",tasa_aciertos);


elapsed_time = etime (clock (), t0);

printf("Duracion: %.3f segundos\n",elapsed_time);


endfunction
```

4.2 Implementación en octave del clasificador por DAG

```
#!/usr/bin/octave -qf

function []=DAG(X,xl,Y,yl)

%Nos guardamos el tiempo para poder comparar con vote

t0 = clock ();

tr=X(find(xl==0|xl==1|xl==2|xl==3),:);

te=Y(find(yl==0|yl==1|yl==2|yl==3),:);

telabels=yl(find(yl==0|yl==1|yl==2|yl==3),:);


%Clase 0 contra 1

indices01=find(xl==0|xl==1);

trlabels01=xl(indices01);

tr01=X(indices01,:);

res01=svmtrain(trlabels01,tr01,'-c 1000 -t 1 -d 2 -q');
```

```
%Clase 0 contra 2
indices02=find(xl==0|xl==2);
trlabels02=xl(indices02);
tr02=X(indices02,:);
res02=svmtrain(trlabels02,tr02,'-c 1000 -t 1 -d 2 -q');

%Clase 0 contra 3
indices03=find(xl==0|xl==3);
trlabels03=xl(indices03);
tr03=X(indices03,:);
res03=svmtrain(trlabels03,tr03,'-c 1000 -t 1 -d 2 -q');
clas03=svmpredict(telabels,te,res03,'-q');

%Clase 1 contra 2
indices12=find(xl==1|xl==2);
trlabels12=xl(indices12);
tr12=X(indices12,:);
res12=svmtrain(trlabels12,tr12,'-c 1000 -t 1 -d 2 -q');

%Clase 1 contra 3
indices13=find(xl==1|xl==3);
trlabels13=xl(indices13);
tr13=X(indices13,:);
res13=svmtrain(trlabels13,tr13,'-c 1000 -t 1 -d 2 -q');
```

```
%Clase 2 contra 3

indices23=find(xl==2|xl==3);

trlabels23=xl(indices23);

tr23=X(indices23,:);

res23=svmtrain(trlabels23,tr23,'-c 1000 -t 1 -d 2 -q');


%Clasificar muestras de test

indices02=find(clas03==0);

telabels02=telabels(indices02);

te02=te(indices02,:);

clas02=svmpredict(telabels02,te02,res02,'-q');


indices13=find(clas03==3);

telabels13=telabels(indices13);

te13=te(indices13,:);

clas13=svmpredict(telabels13,te13,res13,'-q');


%Tengo unos resultados en clas02 y otros en clas13


indices01=find(clas02==0);

telabels01=telabels02(indices01);

te01=te02(indices01,:);

clas01=svmpredict(telabels01,te01,res01,'-q');
```

```
indices23=find(clas13==3);

telabels23=telabels13(indices23);

te23=te13(indices23,:);

clas23=svmpredict(telabels23,te23,res23,'-q');


indices122=find(clas02==2);

telabels122=telabels02(indices122);

te122=te02(indices122,:);

clas122=svmpredict(telabels122,te122,res12,'-q');


indices121=find(clas13==1);

telabels121=telabels13(indices121);

te121=te13(indices121,:);

clas121=svmpredict(telabels121,te121,res12,'-q');

total_muestras=rows(telabels01)+rows(telabels23)+rows(telabels122)+r
ows(telabels121);


%Testear y sacar el error

results=rows(find(clas01==telabels01))+rows(find(clas23==telabels23)
)+rows(find(clas122==telabels122))+rows(find(clas121==telabels121));

tasa_aciertos=results/total_muestras*100;

printf("Tasa de aciertos por DAG: %.3f\n",tasa_aciertos);


elapsed_time = etime (clock (), t0);
```

```
printf("Duracion: %.3f segundos\n",elapsed_time);


endfunction
```