Jose Antonio Mira García

Introducción

Se evaluarán los resultados de tres métodos de clasificación (la mixtura de gaussianas no pude evaluarla).

El objetivo de este proyecto es investigar qué método es más apropiado para la clasificación de la colección de dígitos manuscritos MNIST.

Mixturas de gaussianas

mixgaussian.m Ej 2.1

Se calcula la probabilidad a priori para cada índice de la clase y se normaliza respecto al número de clases. No se utiliza el operador '.' porque octave aplica la normalización a toda la matriz.

```
pkGc{ic} =  sumzk/Nc; % hay que sumar por columnas
```

La media para cada índice se calcula transponiendo la muestra de la clase c para que las dimensiones se ajusten a las requeridas para multiplicarlo por zk, y a cada elemento se le divide entre el sumatorio de zk para normalizar.

```
mu{ic} = Xc'*zk ./ sumzk;
```

Se aplica la matriz de covarianzas a cada componente k. Finalmente se aplica un suavizado.

```
for k = 1:K
  covmat = ((Xc-mu{ic}(:,k)')' * (z(:,k).*(Xc-mu{ic}(:,k)')))/sumz(k);
  % D x D           D x N   aplicar a cada elemento   N x D
  sigma(ic,k) = alpha * covmat +(1-alpha)*eye(D);
end
```

mixgaussian-exp.m Ej 2.2

Se recorren tanto las alphas como las Ks y se guardan en edv. Para posteriormente imprimir la tasa de error.

```
for i=1:length(alphas)
    for j=1:length(Ks)
        edv = mixgaussian(Xtr,xltr,Xdv,xldv, Ks(j),alphas(i));
        printf("%.1e %3d %6.3f\n",alphas(i),Ks(j),edv);
    end;
end;
```

Jose Antonio Mira García

pca+mixgaussian-exp.m Ej 2.3

Aplicamos PCA sobre el conjunto de entrenamiento. Así obtenemos la matriz de proyección completa (W) y las medias (m).

```
[m W]=pca(Xtr);
```

A cada dato se le resta su media (en ambos conjuntos).

```
Xtr = Xtr-m;
Xdv = Xdv-m;
```

Se calculan los nuevos conjuntos en función de la proyección en PCA y luego se pasa como parámetro a mixgaussian.

A diferencia de mixgaussian-exp.m aquí se recorren también todas las dimensiones del PCA.

```
for i=1:length(alphas)
	for k=1:length(pcaKs)
	% Proyeccion
	pcaXtr= Xtr * W(:,1:pcaKs(k));
	pcaXdv= Xdv * W(:,1:pcaKs(k));
		%	NxD	*	D x K
		for j=1:length(Ks)
			edv = mixgaussian(pcaXtr,xltr,pcaXdv,xldv, Ks(j),alphas(i));
			printf("%.1e %3d %3d %6.3f\n",alphas(i),pcaKs(k) ,Ks(j),edv);
		end;
	end;
end;
```

No he podido continuar con el experimento debido a que no me funciona el pca.

```
rosalvio@DESKTOP-LIMON90:/mnt/f/Uni/Uni/2020-APR/mixgaussian$ octave pca+mixgaussian-exp.m ../data/mnist/train-images-id
x3-ubyte.mat.gz ../data/mnist/train-labels-idx1-ubyte.mat.gz "[1,2,5,10,20,50,100]"  "[1,2,5,10,20,50,100]" "[1e-4]" 90
10
octave: X11 DISPLAY environment variable not set
octave: disabling GUI features

  alpha pca Ks dv-err
------- --- --- ------
warning: division by zero
warning: called from
    pca at line 12 column 6
    pca+mixgaussian-exp.m at line 33 column 7
error: eig: EIG: matrix contains Inf or NaN values
error: called from
    pca at line 16 column 11
    pca+mixgaussian-exp.m at line 33 column 7
```

Máquinas de vectores de soporte

Ej 3.2

Linealmente separable

Multiplicadores de Lagrange:

```
>> res.sv_coef
ans =

   0.87472
   0.74989
  -1.62461
```

Vectores de soporte:

```
>> full(res.SVs)
ans =

   1   4
   4   2
   3   4
```

Vector de pesos:

```
>> theta = res.sv_coef' * res.SVs
theta =

  -0.99955  -1.49978
```

```
>> theta0 = sign(res.sv_coef(1)) - theta*res.SVs(1,:)'
theta0 =  7.9987
```

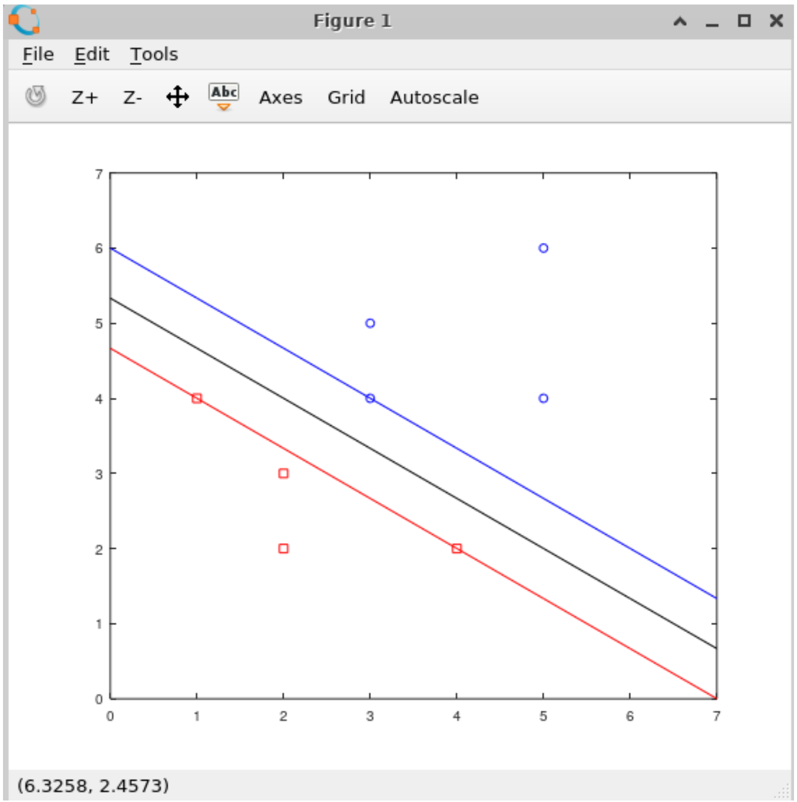
Margen:

```
>> margen = 1 / (theta * theta')
margen =  0.30784
```

Recta de separación:

```
>> x1 = [0:7]
x1 =

   0   1   2   3   4   5   6   7

>> x2 = -theta(1)/theta(2)*x1 - theta0/theta(2)
x2 =

   5.33323   4.66677   4.00030   3.33383   2.66736   2.00090   1.33443   0.66796
```

Jose Antonio Mira García



No separable

Multiplicadores de Lagrange:

```
>> res.sv_coef
ans =

    250.87
    500.75
   1000.00
   -751.62
  -1000.00
```

Vectores de soporte:

```
>> full(res.SVs)
ans =

   1   4
   4   2
   4   4
   3   4
   4   3
```

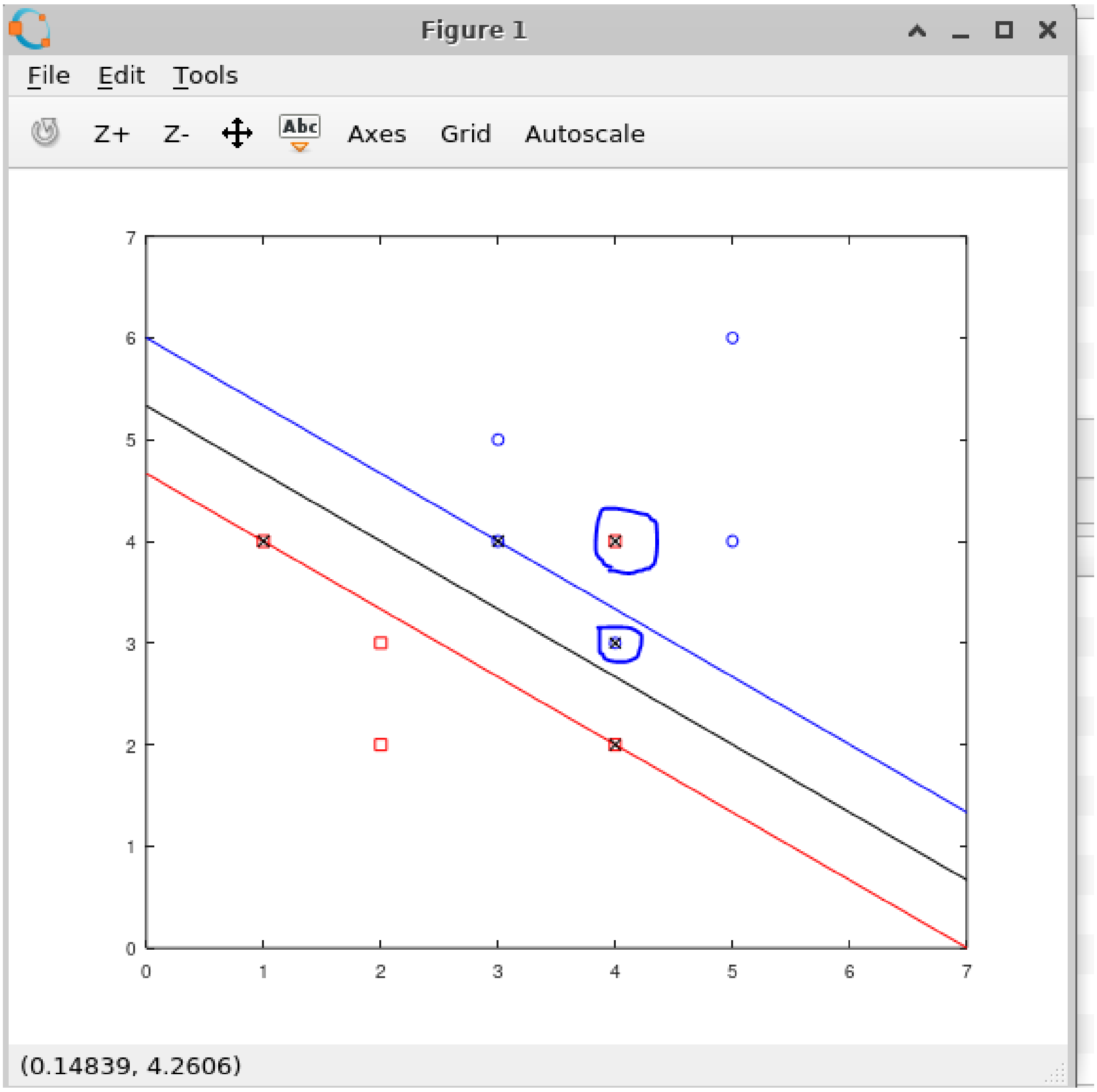
Vector de pesos:

```
>> theta = res.sv_coef' * res.SVs
theta =

  -0.99955  -1.49977

>> theta0 = sign(res.sv_coef(1)) - theta*res.SVs(1,:)'
theta0 =  7.9986
```

Jose Antonio Mira García

Margen:
```
>> margen = 1 / (theta*theta')
margen =  0.30784
```

Recta de separación:
```
>> x2 = -theta(1)/theta(2)*x1 - theta0/theta(2)
x2 =

   5.33323   4.66677   4.00030   3.33384   2.66737   2.00090   1.33444   0.66797
```



Redondeados los vectores de soporte erróneos

Tolerancia de margen:
```
0.00000
0.00000
0.00000
0.00000
0.00000
0.00000
0.00000
0.00000
2.99865
0.50113
```

Ej 3.3

En mi caso sólo pude ejecutar el kernel polinomial por tiempo de cómputo.

```
   C      t   d   err
------- --- --- ------
1         1   1  7.470
1         1   2  1.950
1         1   3  2.090
1         1   4  2.630
1         1   5  3.350
```

Mediante la salida obtenida concluí que los mejores resultados se obtenían mediante un polinomio de segundo grado.

A partir del segundo grado se tiende a empeorar probablemente debido al sobreentrenamiento.

Ej 3.4

```
args = -t 1 -c 1 -d 2 -q

   C      t   d   err    intervalo
------- --- --- ------ ---------
  1     1   2  1.950  0.003
```

En la documentación de MNIST se alcanza un error mínimo de 1.1 mediante un kernel polinomial de grado 4, sin embargo, esto es porque se ha utilizado deskewing como preproceso de los datos.

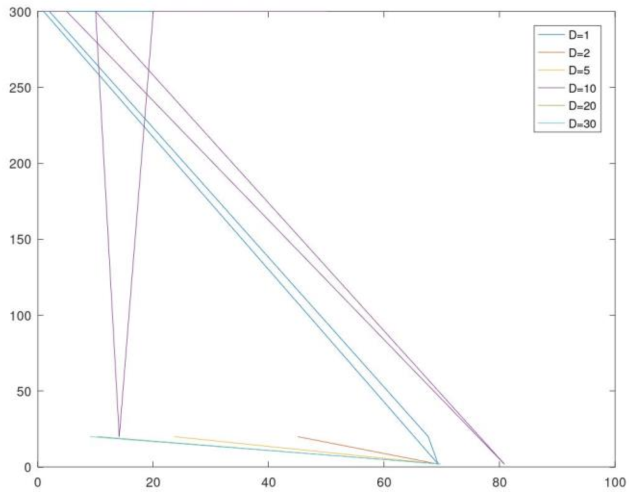
Redes neuronales multicapa

Ej 4.5

Experimento realizado con 40% de test y 10% de validación. El mínimo se consigue con 20 dimensiones de pca y 50 neuronas ocultas. Error = 5.05

NHiddens	D	Epochs	Error
1	1	300	69.317
1	2	300	72.000
1	5	300	69.550
1	10	300	67.617
1	20	300	66.050
1	30	300	65.700
2	1	300	69.233
2	2	300	57.383
2	5	300	52.200
2	10	300	45.017
2	20	300	41.317
2	30	300	45.517
5	1	300	69.450
5	2	300	53.550
5	5	300	32.800
5	10	300	23.583
5	20	300	16.750
5	30	300	14.833
10	1	300	80.783
10	2	300	67.333
10	5	300	27.767
10	10	300	14.100
10	20	300	11.150
10	30	300	9.083
20	1	300	69.533
20	2	300	63.050
20	5	300	25.183
20	10	300	10.333
20	20	300	6.817
20	30	300	7.183
30	1	300	69.800
30	2	300	52.900
30	5	300	24.367
30	10	300	9.067
30	20	300	5.317
30	30	300	5.850
40	1	300	84.383
40	2	300	53.150
40	5	300	24.100
40	10	300	8.433
40	20	300	5.400
40	30	300	5.417
50	1	300	84.733
50	2	300	53.183
50	5	300	89.617
50	10	300	8.200
50	20	300	5.050
50	30	300	5.317

```
filename = "datos.out"
[nH, D, epochs, err] = textread(filename, "%f %f %f %f");
plot(nH(D==1), err(D==1),";D=1;",
nH(D==2), err(D==2), ";D=2;" ,
nH(D==5), err(D==5), ";D=5;",
nH(D==10), err(D==10),";D=10;",
nH(D==20), err(D==20),";D=20;",
nH(D==30), err(D==30),";D=30;");
```



Ej 4.5

nHiddens	D	epochs	dv-err	intervalo
50	20	300	4.880	0.002

Se consigue un error similar al aplicar una red neuronal de 2 capas y 300 neuronas ocultas (4.7), sin embargo se consiguen mejores resultados mediante SVMs, también probablemente por sobreentrenamiento como hemos visto en el caso de SVMs con polinomios de grado alto.

Conclusión

Para la tarea MNIST resulta mucho más eficiente usar SVMs, probablemente podría haber alcanzado incluso mejores resultados si hubiera podido computar otros tipos de kernel.