Sessió 10

1. App Classification Learner

L’objectiu d’aquesta pràctica és aprendre a classificar mostresa partir dels seus vectors de caracterìstiques. Usarem la App Classification learner de Matlab. Trobareu la informació necessària a:

https://uk.mathworks.com/help/stats/train-decision-trees-in-classification-learner-app.html

https://uk.mathworks.com/help/stats/train-classification-models-in-classification-learner-app.html

Podeu utilitzar el dataset Fisher iris (tot un clàssic) i experimentar amb diferents classificadors. En acabar heu de ser capaços de:

- Tunnejar correctament els paràmetres d’un classificador

- Probar diferents classificadors i escollir-ne els que donin millors resultats

- Fer experiments amb rigor (p.ex: cross-validation)

- Presentar els resultats de forma correcta (corba RoC, matriu de confusió...)

```
fishertable = readtable('fisheriris.csv');
view(trainedModel.ClassificationTree,'Mode','graph');
yfit = trainedModel.predictFcn(fishertable);
```

2. Classificació automàtica d’espècies arbòries

Un cop domineu la app Clasification learner, es planteja un problema de classificació amb imatges reals. Dins la tasca corresponent a la sessió 10 a Atenea, trobareu imatges de fulles de roure, faig i plàtan. Entreneu varios classificadors per aquestes tres espècies usant com a vectors de característiques els seus descriptors de Fourier. Es demana un informe (en pdf) que inclogui el codi usat per a obtenir els descriptors, la descripció dels experiments realitzats, els classificadors que han funcionat millor i els resultats obtinguts. Indiqueu quines caracaterístiques del vector són necessàries per a la correcta classificació, i quines no són significatives.

Al trabajar con los descriptores de Fourier de las respectivas imagenes, se nos planteó el dilema de usar el modulo de los descriptores odividir los descriptores, que son números complejos, en dos partes, real e imaginaria.

Haremos pruebas de ambos casos y decidiremos cual nos dá un mejor resultado.

Clasificación

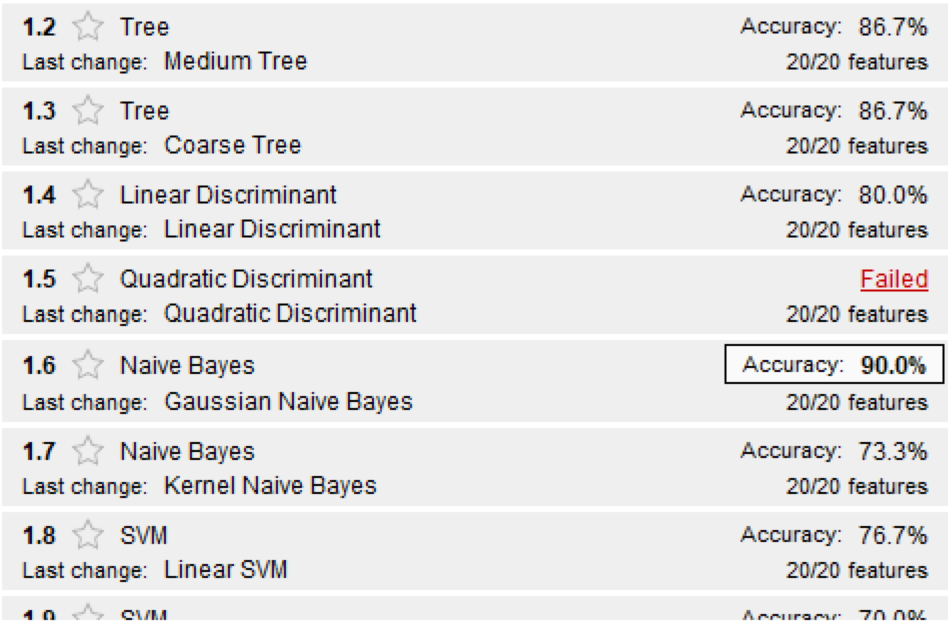
Primero estudiaremos los resultado en función del número de descriptores (5, 20 , 50 ,75)

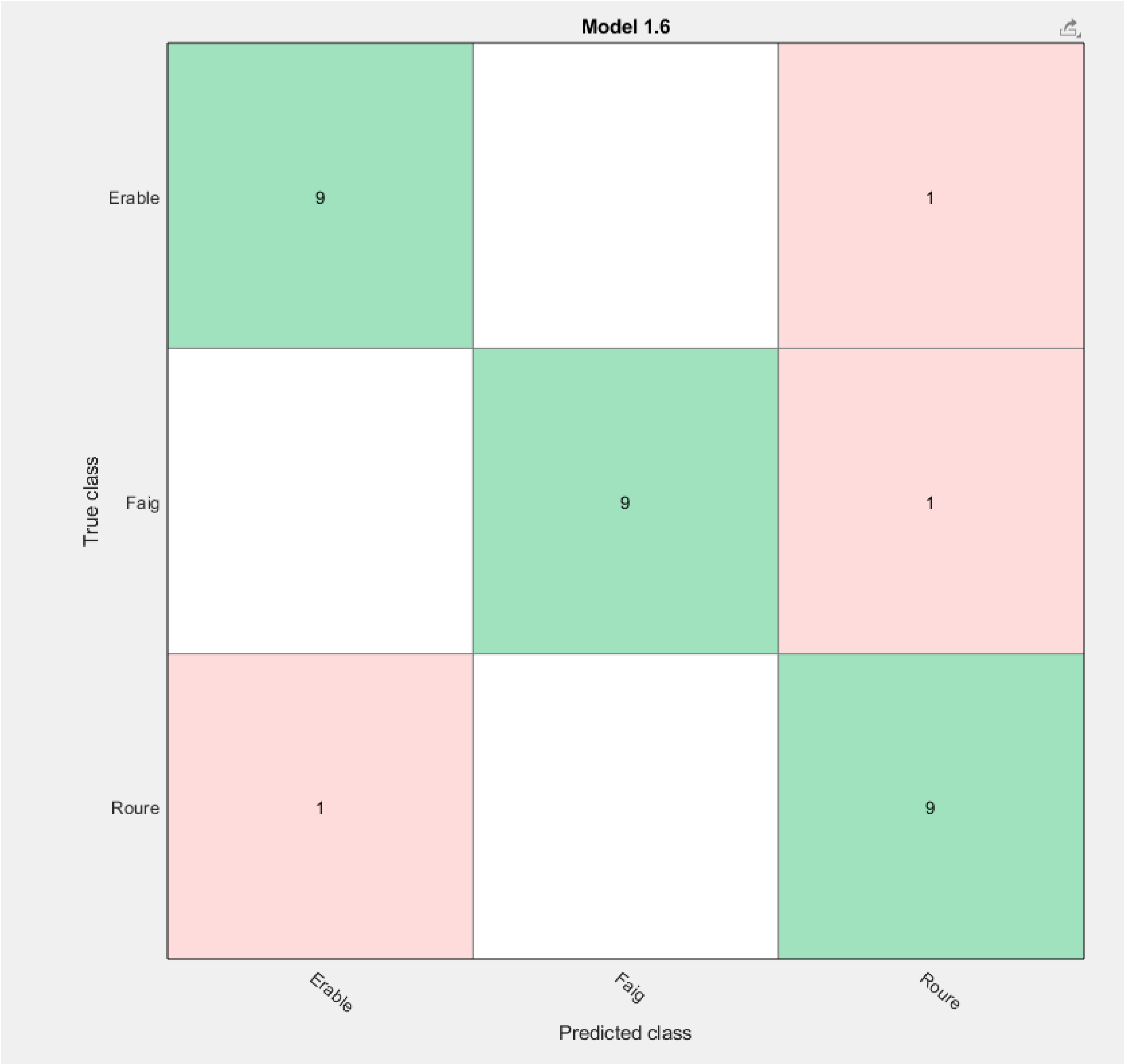
Para las pruebas, hemos decidido dividir nuestros datos en 2 conjuntos: Train y Test.

Test es un 30% del total de los datos y serán escogidos aleatoriamente en cada prueba, sin embargo, sabremos que del 30%, el primer tercio son Arce, el segundo tercio son Haya y el último tercio son Roble.

Clasificación usando parte real e imaginaria

- Usando 5 descriptores nos da los siguientes resultados:



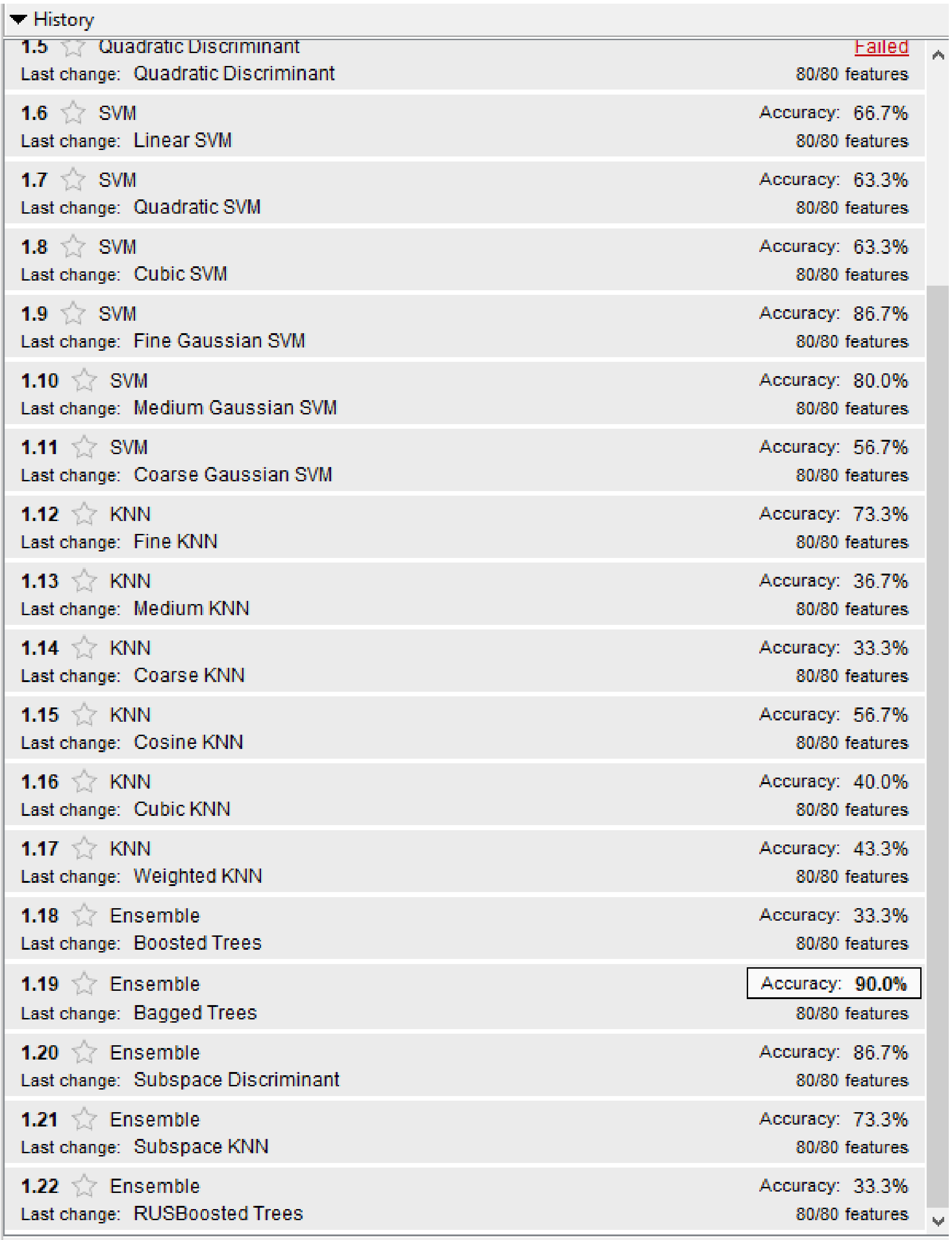


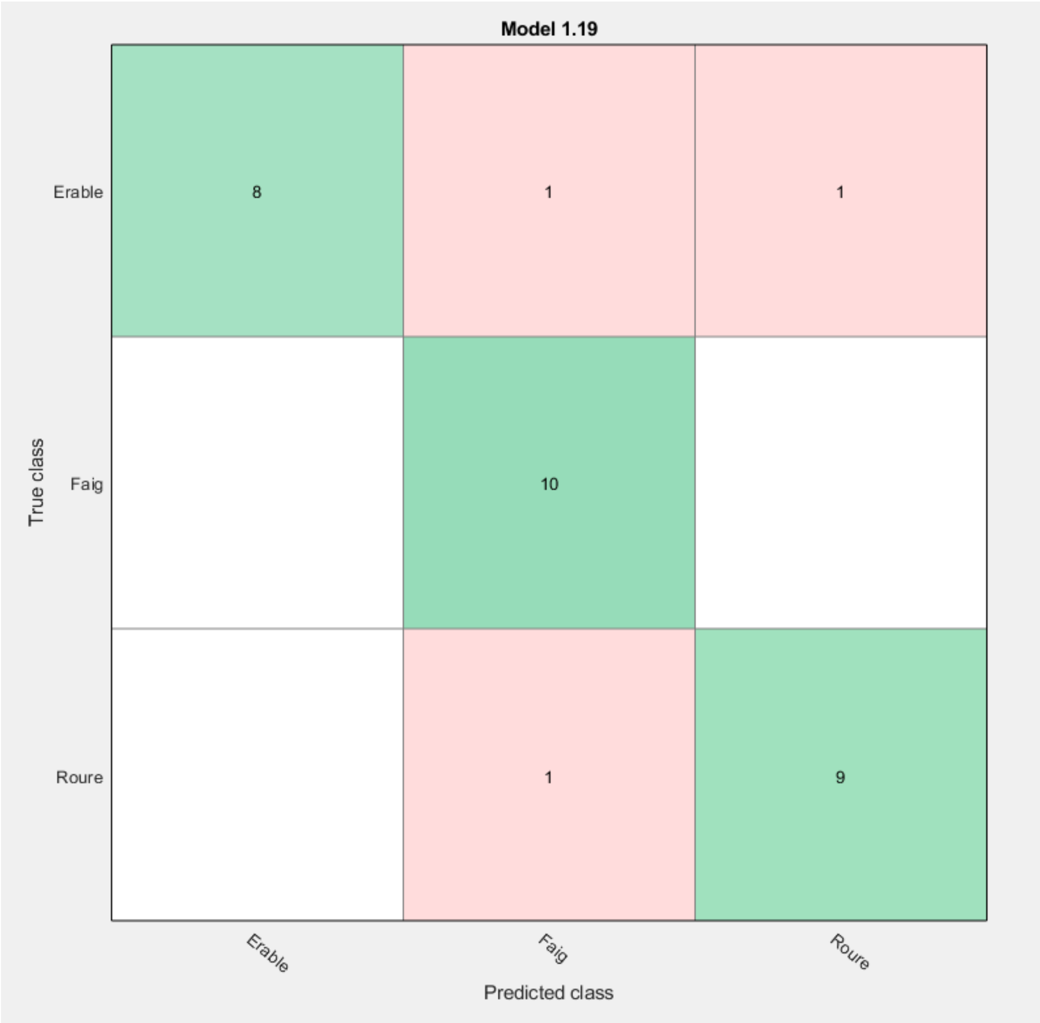
Haremos una prueba con los datos Test:

1	Roure
2	Roure
3	Erable
4	Erable
5	Erable
6	Erable
7	Faig
8	Roure
9	Faig
10	Faig
11	Faig
12	Faig
13	Roure
14	Roure
15	Roure
16	Roure
17	Roure
18	Faig
19	

Como se puede observar y sabiendo los resultados correctos expuestos arriba, hay un gran porcentaje de fallos, un 20% aproximadamente.

- Usando 20 descriptores nos da los siguientes resultados:


History
1.5 Quadratic Discriminant Failed
Last change: Quadratic Discriminant 80/80 features
1.6 SVM Accuracy: 66.7%
Last change: Linear SVM 80/80 features
1.7 SVM Accuracy: 63.3%
Last change: Quadratic SVM 80/80 features
1.8 SVM Accuracy: 63.3%
Last change: Cubic SVM 80/80 features
1.9 SVM Accuracy: 86.7%
Last change: Fine Gaussian SVM 80/80 features
1.10 SVM Accuracy: 80.0%
Last change: Medium Gaussian SVM 80/80 features
1.11 SVM Accuracy: 56.7%
Last change: Coarse Gaussian SVM 80/80 features
1.12 KNN Accuracy: 73.3%
Last change: Fine KNN 80/80 features
1.13 KNN Accuracy: 36.7%
Last change: Medium KNN 80/80 features
1.14 KNN Accuracy: 33.3%
Last change: Coarse KNN 80/80 features
1.15 KNN Accuracy: 56.7%
Last change: Cosine KNN 80/80 features
1.16 KNN Accuracy: 40.0%
Last change: Cubic KNN 80/80 features
1.17 KNN Accuracy: 43.3%
Last change: Weighted KNN 80/80 features
1.18 Ensemble Accuracy: 33.3%
Last change: Boosted Trees 80/80 features
1.19 Ensemble Accuracy: 90.0%
Last change: Bagged Trees 80/80 features
1.20 Ensemble Accuracy: 86.7%
Last change: Subspace Discriminant 80/80 features
1.21 Ensemble Accuracy: 73.3%
Last change: Subspace KNN 80/80 features
1.22 Ensemble Accuracy: 33.3%
Last change: RUSBoosted Trees 80/80 features

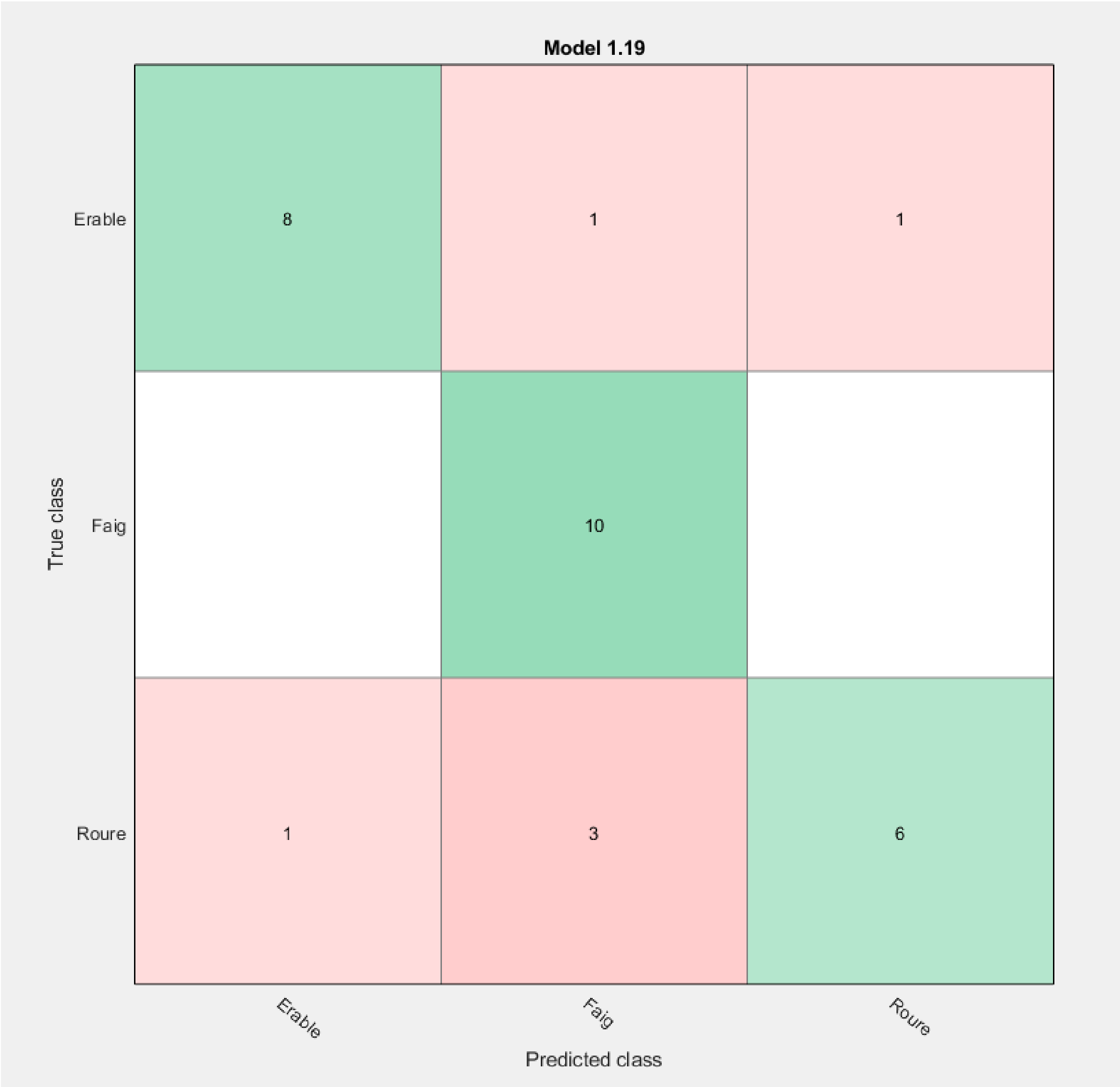


	1
1	Erable
2	Erable
3	Erable
4	Erable
5	Erable
6	Erable
7	Faig
8	Faig
9	Faig
10	Faig
11	Faig
12	Faig
13	Faig
14	Roure
15	Faig
16	Roure
17	Roure
18	Roure

Como podemos observar solo ha fallado 2/18. Lo que demuestra que ha disminuido bastante el error.

- Usando 50 descriptores nos da los siguientes resultados:

Model	Type	Last change	Accuracy	Features
		Quadratic Discriminant		200/200 features
1.6	SVM	Linear SVM	50.0%	200/200 features
1.7	SVM	Quadratic SVM	46.7%	200/200 features
1.8	SVM	Cubic SVM	46.7%	200/200 features
1.9	SVM	Fine Gaussian SVM	66.7%	200/200 features
1.10	SVM	Medium Gaussian SVM	**80.0%**	200/200 features
1.11	SVM	Coarse Gaussian SVM	43.3%	200/200 features
1.12	KNN	Fine KNN	50.0%	200/200 features
1.13	KNN	Medium KNN	33.3%	200/200 features
1.14	KNN	Coarse KNN	33.3%	200/200 features
1.15	KNN	Cosine KNN	40.0%	200/200 features
1.16	KNN	Cubic KNN	33.3%	200/200 features
1.17	KNN	Weighted KNN	33.3%	200/200 features
1.18	Ensemble	Boosted Trees	33.3%	200/200 features
1.19	Ensemble	Bagged Trees	**80.0%**	200/200 features
1.20	Ensemble	Subspace Discriminant	60.0%	200/200 features
1.21	Ensemble	Subspace KNN	76.7%	200/200 features
1.22	Ensemble	RUSBoosted Trees	33.3%	200/200 features

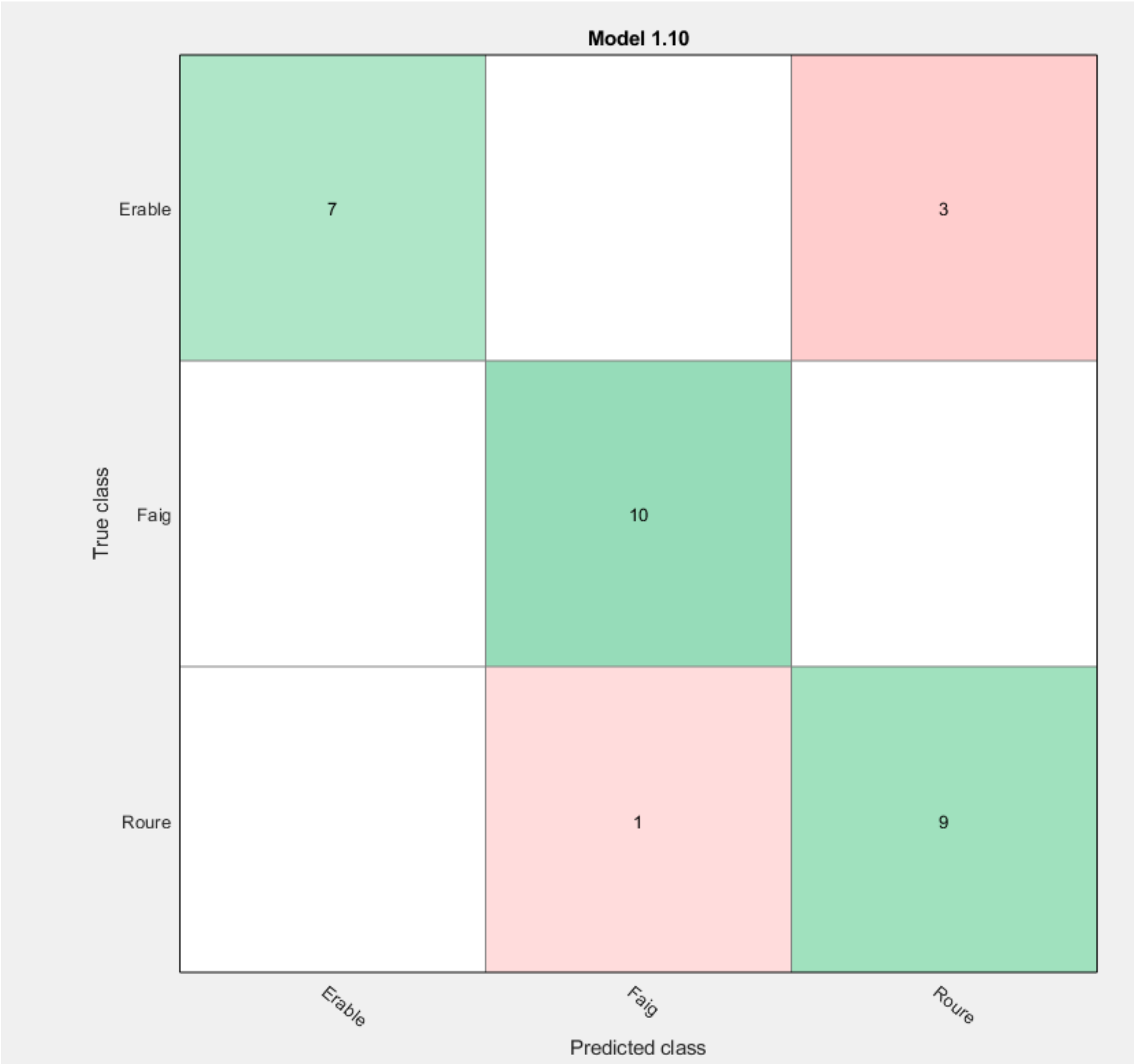

Model 1.19
True class
Erable
Faig
Roure
8
1
1
10
1
3
6
Erable
Faig
Roure
Predicted class

	1
1	Roure
2	Erable
3	Erable
4	Erable
5	Roure
6	Erable
7	Faig
8	Faig
9	Faig
10	Faig
11	Faig
12	Faig
13	Faig
14	Roure
15	Roure
16	Roure
17	Roure
18	Roure

Como podemos observar ha fallado en 3/16. Como se puede admirar los resultados han empeorado respecto a los anteriores experimentos.

- Usando 75 descriptores nos da los siguientes resultados:

Model	Type	Last change	Accuracy	Features
1.5	Quadratic Discriminant	Quadratic Discriminant	Failed	300/300 features
1.6	SVM	Linear SVM	Accuracy: 46.7%	300/300 features
1.7	SVM	Quadratic SVM	Accuracy: 43.3%	300/300 features
1.8	SVM	Cubic SVM	Accuracy: 43.3%	300/300 features
1.9	SVM	Fine Gaussian SVM	Accuracy: 70.0%	300/300 features
1.10	SVM	Medium Gaussian SVM	Accuracy: **86.7%**	300/300 features
1.11	SVM	Coarse Gaussian SVM	Accuracy: 36.7%	300/300 features
1.12	KNN	Fine KNN	Accuracy: 40.0%	300/300 features
1.13	KNN	Medium KNN	Accuracy: 33.3%	300/300 features
1.14	KNN	Coarse KNN	Accuracy: 33.3%	300/300 features
1.15	KNN	Cosine KNN	Accuracy: 46.7%	300/300 features
1.16	KNN	Cubic KNN	Accuracy: 33.3%	300/300 features
1.17	KNN	Weighted KNN	Accuracy: 33.3%	300/300 features
1.18	Ensemble	Boosted Trees	Accuracy: 33.3%	300/300 features
1.19	Ensemble	Bagged Trees	Accuracy: 83.3%	300/300 features
1.20	Ensemble	Subspace Discriminant	Accuracy: 56.7%	300/300 features
1.21	Ensemble	Subspace KNN	Accuracy: 76.7%	300/300 features
1.22	Ensemble	RUSBoosted Trees	Accuracy: 33.3%	300/300 features

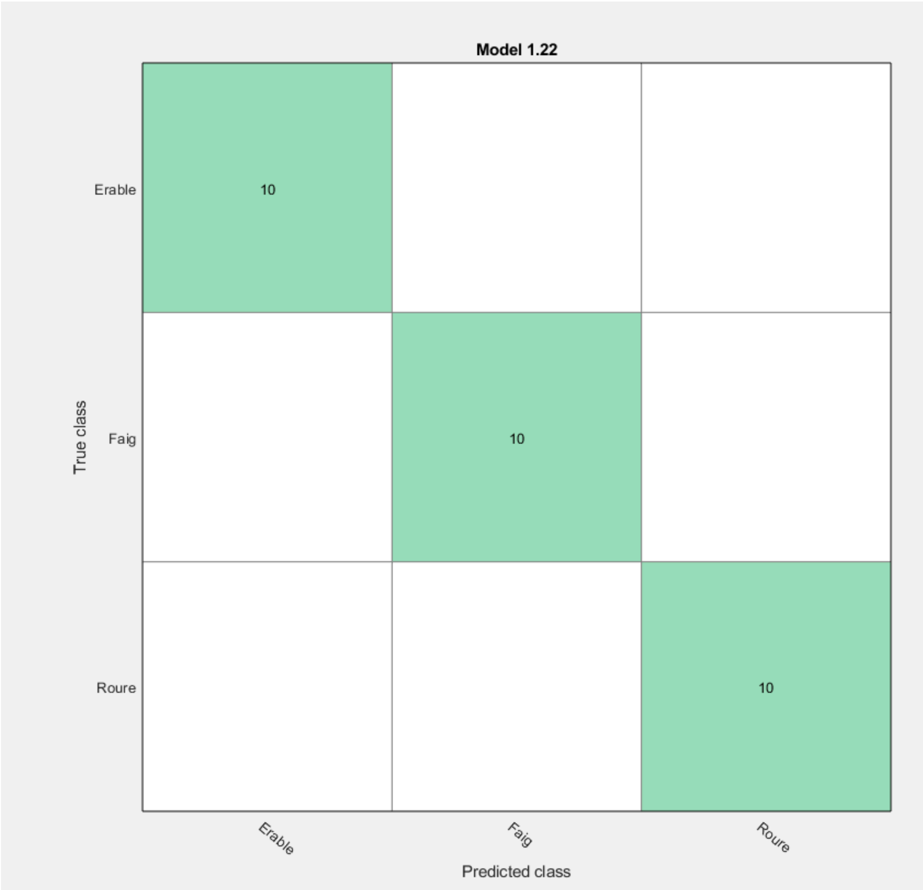

Model 1.10
True class
Erable
Faig
Roure
7
3
10
1
9
Erable
Faig
Roure
Predicted class

	1
1	Roure
2	Erable
3	Erable
4	Roure
5	Erable
6	Erable
7	Faig
8	Faig
9	Faig
10	Faig
11	Faig
12	Faig
13	Roure
14	Roure
15	Roure
16	Roure
17	Roure
18	Roure

Como podemos observar ha fallado en 2/16. Como se puede admirar los resultados han mejorado pero no son superior a los que han dado con 5 y 20 descriptores.

- Usando 5 descriptores nos da los siguientes resultados:

Model	Type	Last change	Accuracy	Features
1.7	Naive Bayes	Kernel Naive Bayes	73.3%	10/10 features
1.8	SVM	Linear SVM	96.7%	10/10 features
1.9	SVM	Quadratic SVM	96.7%	10/10 features
1.10	SVM	Cubic SVM	93.3%	10/10 features
1.11	SVM	Fine Gaussian SVM	80.0%	10/10 features
1.12	SVM	Medium Gaussian SVM	93.3%	10/10 features
1.13	SVM	Coarse Gaussian SVM	73.3%	10/10 features
1.14	KNN	Fine KNN	96.7%	10/10 features
1.15	KNN	Medium KNN	80.0%	10/10 features
1.16	KNN	Coarse KNN	33.3%	10/10 features
1.17	KNN	Cosine KNN	70.0%	10/10 features
1.18	KNN	Cubic KNN	80.0%	10/10 features
1.19	KNN	Weighted KNN	90.0%	10/10 features
1.20	Ensemble	Boosted Trees	33.3%	10/10 features
1.21	Ensemble	Bagged Trees	83.3%	10/10 features
1.22	Ensemble	Subspace Discriminant	**100.0%**	10/10 features
1.23	Ensemble		96.7%	

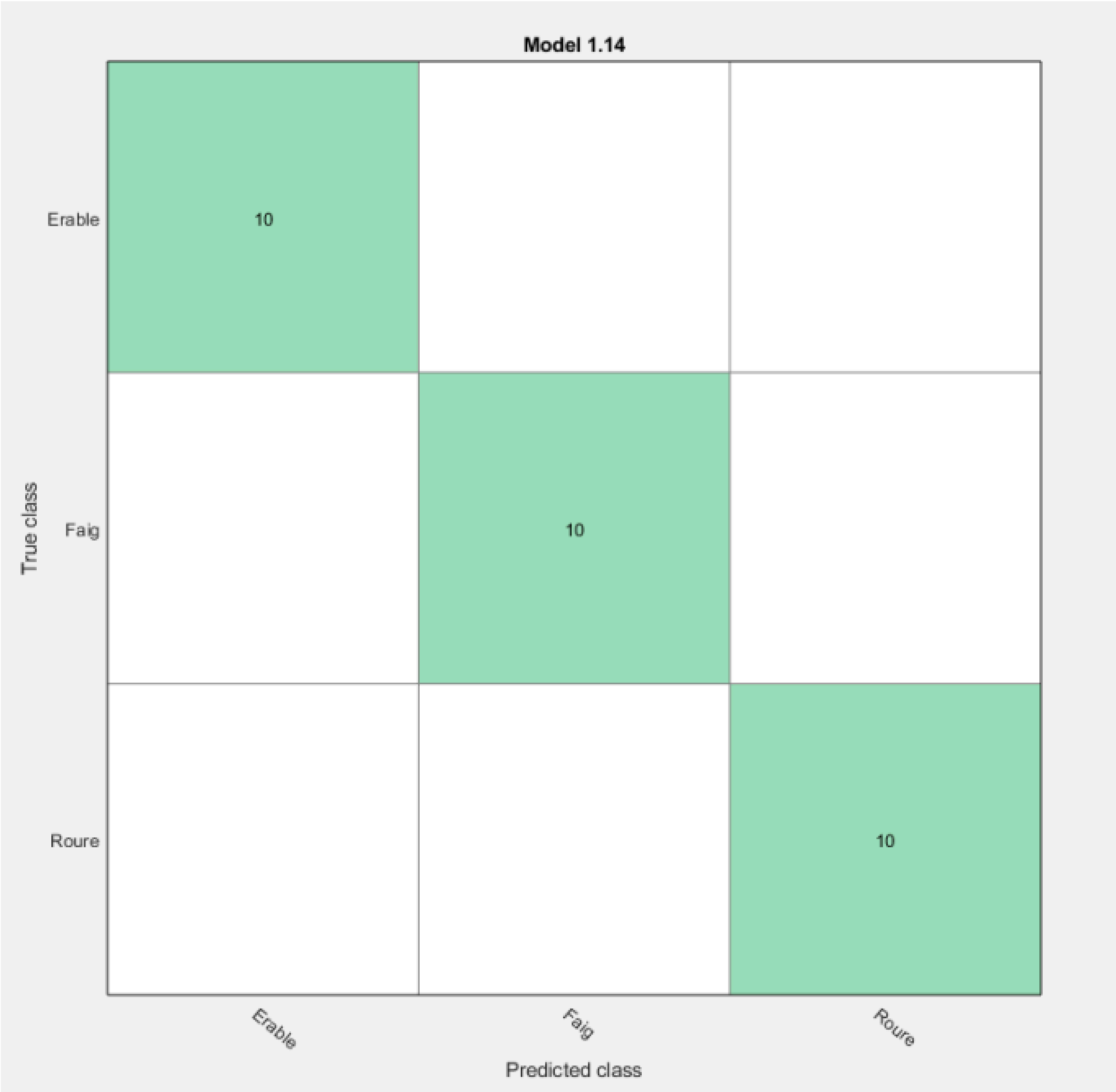


1	Erable
2	Erable
3	Erable
4	Erable
5	Erable
6	Erable
7	Faig
8	Faig
9	Faig
10	Faig
11	Faig
12	Faig
13	Roure
14	Roure
15	Roure
16	Roure
17	Roure
18	Roure

Buenos resultados aunque es posible debido a la poca cantidad de datos

- Usando 20 descriptores nos da los siguientes resultados:

1.1	Tree	Last change: Fine Tree	Accuracy: 93.3%	40/40 features
1.2	Tree	Last change: Medium Tree	Accuracy: 93.3%	40/40 features
1.3	Tree	Last change: Coarse Tree	Accuracy: 93.3%	40/40 features
1.4	Linear Discriminant	Last change: Linear Discriminant	Accuracy: 96.7%	40/40 features
1.5	Quadratic Discriminant	Last change: Quadratic Discriminant	Failed	40/40 features
1.6	Naive Bayes	Last change: Gaussian Naive Bayes	Accuracy: 93.3%	40/40 features
1.7	Naive Bayes	Last change: Kernel Naive Bayes	Accuracy: 90.0%	40/40 features
1.8	SVM	Last change: Linear SVM	Accuracy: 90.0%	40/40 features
1.9	SVM	Last change: Quadratic SVM	Accuracy: 96.7%	40/40 features
1.10	SVM	Last change: Cubic SVM	Accuracy: 96.7%	40/40 features
1.11	SVM	Last change: Fine Gaussian SVM	Accuracy: 56.7%	40/40 features
1.12	SVM	Last change: Medium Gaussian SVM	Accuracy: 93.3%	40/40 features
1.13	SVM	Last change: Coarse Gaussian SVM	Accuracy: 83.3%	40/40 features
1.14	KNN	Last change: Fine KNN	Accuracy: **100.0%**	40/40 features
1.15	KNN	Last change: Medium KNN	Accuracy: 86.7%	40/40 features
1.16	KNN	Last change: Coarse KNN	Accuracy: 33.3%	40/40 features

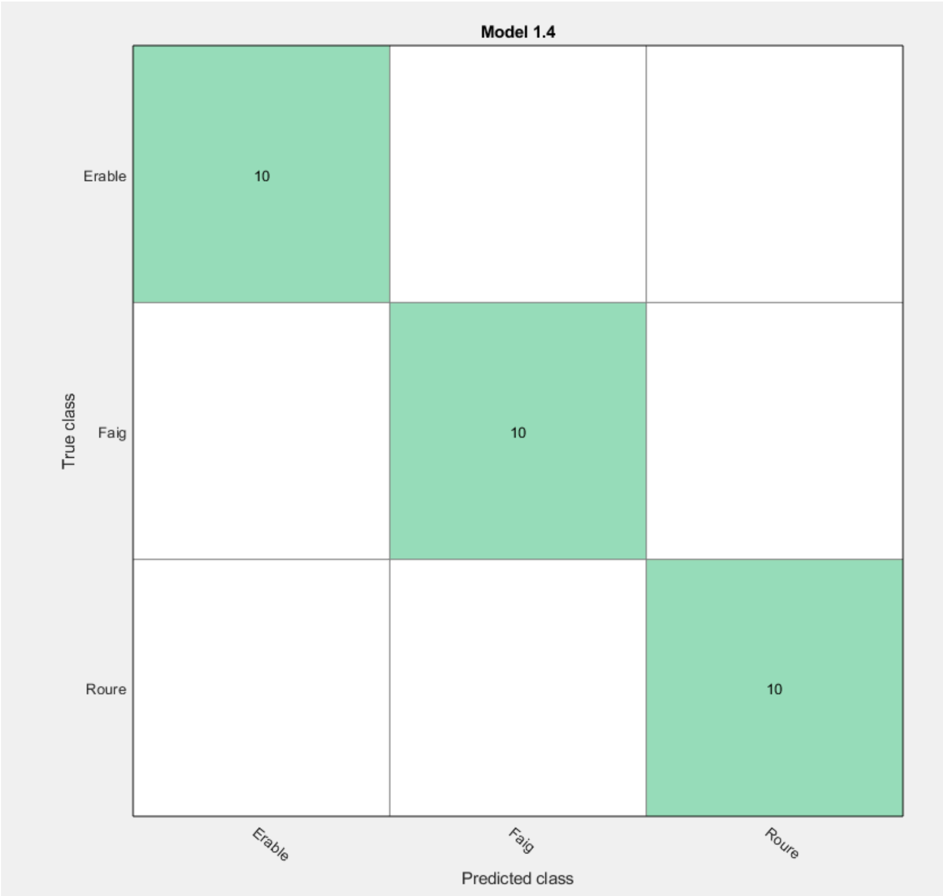

Model 1.14
True class
Erable
Faig
Roure
10
10
10
Erable
Faig
Roure
Predicted class

1	Erable
2	Erable
3	Erable
4	Erable
5	Erable
6	Erable
7	Faig
8	Faig
9	Faig
10	Faig
11	Faig
12	Faig
13	Faig
14	Roure
15	Roure
16	Faig
17	Roure
18	Roure
19	

En este experimento podemos ver que hay un clasificador que funciona con 100% de exito. En contraposición ha fallado 2 veces.

- Usando 50 descriptores nos da los siguientes resultados:

1.1	Tree	Last change: Fine Tree	Accuracy: 76.7%	100/100 features
1.2	Tree	Last change: Medium Tree	Accuracy: 76.7%	100/100 features
1.3	Tree	Last change: Coarse Tree	Accuracy: 76.7%	100/100 features
1.4	Linear Discriminant	Last change: Linear Discriminant	Accuracy: **100.0%**	100/100 features
1.5	Quadratic Discriminant	Last change: Quadratic Discriminant	Failed	100/100 features
1.6	Naive Bayes	Last change: Gaussian Naive Bayes	Accuracy: 90.0%	100/100 features
1.7	Naive Bayes	Last change: Kernel Naive Bayes	Accuracy: 86.7%	100/100 features
1.8	SVM	Last change: Linear SVM	Accuracy: 90.0%	100/100 features
1.9	SVM	Last change: Quadratic SVM	Accuracy: 93.3%	100/100 features
1.10	SVM	Last change: Cubic SVM	Accuracy: **100.0%**	100/100 features
1.11	SVM	Last change: Fine Gaussian SVM	Accuracy: 60.0%	100/100 features
1.12	SVM	Last change: Medium Gaussian SVM	Accuracy: **100.0%**	100/100 features
1.13	SVM	Last change: Coarse Gaussian SVM	Accuracy: 83.3%	100/100 features
1.14	KNN	Last change: Fine KNN	Accuracy: **100.0%**	100/100 features
1.15	KNN	Last change: Medium KNN	Accuracy: 90.0%	100/100 features
1.16	KNN	Last change: Coarse KNN	Accuracy: 33.3%	100/100 features

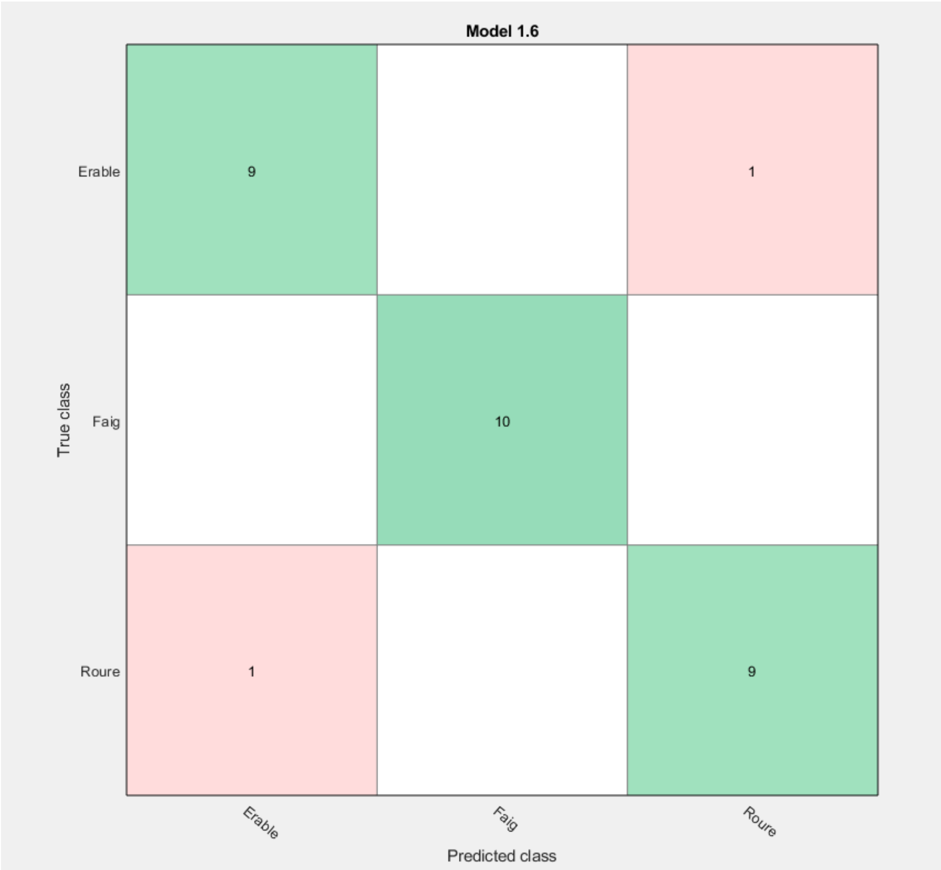


	1
1	Erable
2	Erable
3	Erable
4	Erable
5	Erable
6	Erable
7	Faig
8	Faig
9	Faig
10	Faig
11	Faig
12	Faig
13	Roure
14	Roure
15	Erable
16	Roure
17	Roure
18	Roure
19	

Como podemos observar con 50 descriptores hay un clasificador que nos sale con 100 % de error. En contraposición al usar los test nos sale que se equivoca una vez

- Usando 75 descriptores nos da los siguientes resultados:

1.1	Tree	Last change: Fine Tree	Accuracy: 93.3% — 150/150 features
1.2	Tree	Last change: Medium Tree	Accuracy: 93.3% — 150/150 features
1.3	Tree	Last change: Coarse Tree	Accuracy: 93.3% — 150/150 features
1.4	Linear Discriminant	Last change: Linear Discriminant	Failed — 150/150 features
1.5	Quadratic Discriminant	Last change: Quadratic Discriminant	Failed — 150/150 features
1.6	Naive Bayes	Last change: Gaussian Naive Bayes	Accuracy: 93.3% — 150/150 features
1.7	Naive Bayes	Last change: Kernel Naive Bayes	Accuracy: 90.0% — 150/150 features
1.8	SVM	Last change: Linear SVM	Accuracy: 76.7% — 150/150 features
1.9	SVM	Last change: Quadratic SVM	Accuracy: 76.7% — 150/150 features
1.10	SVM	Last change: Cubic SVM	Accuracy: 80.0% — 150/150 features
1.11	SVM	Last change: Fine Gaussian SVM	Accuracy: 40.0% — 150/150 features
1.12	SVM	Last change: Medium Gaussian SVM	Accuracy: 56.7% — 150/150 features
1.13	SVM	Last change: Coarse Gaussian SVM	Accuracy: 60.0% — 150/150 features



1	Erable
2	Erable
3	Erable
4	Erable
5	Erable
6	Erable
7	Faig
8	Faig
9	Faig
10	Faig
11	Faig
12	Faig
13	Faig
14	Roure
15	Roure
16	Faig
17	Roure
18	Roure

Como podemos observar a partir de 75 empieza a fallar el algoritmo. Probablemente se deba a que utilizamos demasiados descriptores.

Conclusiones

Como conclusión diriamos que el numero de descriptores es un factor importante para crear un clasificador. El hecho de coger poco hace que el clasificador no tenga demasiado features para estudiar en contraposición si cogemos más de 50 features los modelos que salian despues de entrenar un clasificador fallaban. También es importante ver que features se puede extraer para hacer un buen clasificador. Como hemos podido ver en la mayoria de clasificadores cuyo features era el valor absoluto de la transformación de fourier daban buenos resultados.

```
%im = imread('l2nr002.jpg')
%[descriptor] = descriptor_fourier(im,nan)
p = randperm(16);
train = p(1:10);
test = p(11:16);
[T1Train, T2Train, dataTrain] = getdataexercise2(train)
```

```
data = 30×100
    3.8587    7.0293    7.6136    7.5930    7.1094    7.4845    4.9829    6.2418 ...
  -27.1308    7.4173    6.5639    6.5329    7.1705    6.4775    5.7044    6.1339
  -26.5127    9.9683    9.6453    8.8969    7.4996    5.6353    6.9428    7.7369
  -26.8457    8.2493    8.2071    8.0854    7.3447    7.5338    6.3328    3.2623
  -27.3319    6.7330    6.3030    5.7996    6.6231    6.4854    5.1848    5.8195
  -26.1190    8.8138    8.3693    8.6093    8.1314    8.0509    6.1810    6.9407
    3.6478    6.2503    6.0083    5.1783    6.1637    6.3466    5.4748    4.7473
  -27.6601    6.7120    7.0322    7.0347    7.1219    6.6904    7.1124    6.3540
  -25.9204    9.5454    9.4655    9.2270    8.4868    8.3080    7.9425    7.0082
  -26.6238    9.2434    8.9246    8.6632    8.0915    7.3710    7.3055    6.8677
      ⋮
```

T1Train = 30×2 table

...

	Var1					
1	-5.8124	32.2384	-1.8461e+03	1.4800e+03	738.2587	-1.6510e+03
2	-0.0000	-166.4627	677.0051	612.0444	-1.2720e+03	-230.1854
3	-0.0000	-668.8164	-1.4241e+04	5.0108e+03	544.6774	259.2328
4	0.0000	-2.5569e+03	310.1030	2.1587e+03	-1.5302e+03	819.1064
5	-0.0000	-295.3093	265.2105	113.2147	-175.2901	637.2709
6	-0.0000	2.7817e+03	-4.2877e+03	-1.4423e+03	3.3235e+03	1.7034e+03
7	20.5164	-101.2310	256.1201	172.6546	315.1910	324.3557
8	0.0000	-703.9492	-1.1243e+03	-2.7106	1.2333e+03	462.2507
9	0.0000	-1.9026e+03	-6.6437e+03	1.0106e+04	-2.2938e+03	-739.9788
10	-0.0000	-5.6256e+03	-6.4431e+03	3.0559e+03	2.4166e+03	-974.1993

⋮

T2Train = 30×2 table

...

	Var1					
1	3.8587	7.0293	7.6136	7.5930	7.1094	7.4845
2	-27.1308	7.4173	6.5639	6.5329	7.1705	6.4775
3	-26.5127	9.9683	9.6453	8.8969	7.4996	5.6353

	Var1					
4	-26.8457	8.2493	8.2071	8.0854	7.3447	7.5338
5	-27.3319	6.7330	6.3030	5.7996	6.6231	6.4854
6	-26.1190	8.8138	8.3693	8.6093	8.1314	8.0509
7	3.6478	6.2503	6.0083	5.1783	6.1637	6.3466
8	-27.6601	6.7120	7.0322	7.0347	7.1219	6.6904
9	-25.9204	9.5454	9.4655	9.2270	8.4868	8.3080
10	-26.6238	9.2434	8.9246	8.6632	8.0915	7.3710

⋮

```
dataTrain = 30×100
    3.8587    7.0293    7.6136    7.5930    7.1094    7.4845    4.9829    6.2418 ···
  -27.1308    7.4173    6.5639    6.5329    7.1705    6.4775    5.7044    6.1339
  -26.5127    9.9683    9.6453    8.8969    7.4996    5.6353    6.9428    7.7369
  -26.8457    8.2493    8.2071    8.0854    7.3447    7.5338    6.3328    3.2623
  -27.3319    6.7330    6.3030    5.7996    6.6231    6.4854    5.1848    5.8195
  -26.1190    8.8138    8.3693    8.6093    8.1314    8.0509    6.1810    6.9407
    3.6478    6.2503    6.0083    5.1783    6.1637    6.3466    5.4748    4.7473
  -27.6601    6.7120    7.0322    7.0347    7.1219    6.6904    7.1124    6.3540
  -25.9204    9.5454    9.4655    9.2270    8.4868    8.3080    7.9425    7.0082
  -26.6238    9.2434    8.9246    8.6632    8.0915    7.3710    7.3055    6.8677
      ⋮
```

```
[T1Test, T2Test, dataTest] = getdataexercise2(test)
```

```
data = 18×100
  -27.3793    6.7368    6.4260    4.3216    6.5274    6.2917    4.9732    6.1190 ···
  -26.3696    8.6595    8.7784    8.5441    8.3605    7.7050    7.8760    5.5064
    4.2374    7.9803    7.6499    5.5623    7.1377    7.1419    6.4169    6.8696
  -26.9452    9.5737    9.3567    8.8743    7.7975    7.7446    7.0923    7.1415
    3.8152    6.8771    6.1727    5.9180    5.8778    6.2619    5.9195    5.6903
  -26.6475    9.1840    8.9940    8.5383    7.5868    6.6525    6.5862    7.1747
    3.3640    7.0582    4.5301    6.1609    3.5153    5.1074    3.2495    4.8730
  -29.3081    5.9744    3.3381    5.3789    3.9624    4.0620    3.0034    3.6673
    2.9744    5.5075    4.4792    4.9996    3.0557    3.5737    2.8911    3.2622
    3.0321    6.6905    4.6489    5.6019    3.0646    4.7761    3.0004    4.2917
      ⋮
```

```
T1Test = 18×2 table
```

···

	Var1					
1	0.0000	-483.2583	488.8170	-39.2839	170.3581	241.2344
2	0.0000	-3.6158e+03	-5.4124e+03	2.7842e+03	2.7615e+03	-1.3597e+03
3	-1.3150	-1.5565e+03	856.8904	255.3012	419.7425	896.1349
4	-0.0000	-744.7536	-1.1521e+04	1.0325e+03	2.4044e+03	877.7265
5	6.8058	-583.9130	121.7863	358.2363	346.0846	-71.2172
6	0.0000	-4.0645e+03	-6.6488e+03	3.6187e+03	886.7399	216.5933
7	0.4527	255.4347	-63.7501	-18.9704	24.8369	-26.0260

	Var1					
8	-0.0000	-215.0699	16.4410	121.2552	-18.2832	-37.1078
9	-2.4790	36.6957	75.8535	62.0926	-21.2341	-23.2910
10	-4.2229	-20.5771	-80.2211	168.9080	-20.1176	-99.6831

⋮

```
T2Test = 18×2 table
```

...

	Var1					
1	-27.3793	6.7368	6.4260	4.3216	6.5274	6.2917
2	-26.3696	8.6595	8.7784	8.5441	8.3605	7.7050
3	4.2374	7.9803	7.6499	5.5623	7.1377	7.1419
4	-26.9452	9.5737	9.3567	8.8743	7.7975	7.7446
5	3.8152	6.8771	6.1727	5.9180	5.8778	6.2619
6	-26.6475	9.1840	8.9940	8.5383	7.5868	6.6525
7	3.3640	7.0582	4.5301	6.1609	3.5153	5.1074
8	-29.3081	5.9744	3.3381	5.3789	3.9624	4.0620
9	2.9744	5.5075	4.4792	4.9996	3.0557	3.5737
10	3.0321	6.6905	4.6489	5.6019	3.0646	4.7761

⋮

```
dataTest = 18×100
  -27.3793    6.7368    6.4260    4.3216    6.5274    6.2917    4.9732    6.1190 ···
  -26.3696    8.6595    8.7784    8.5441    8.3605    7.7050    7.8760    5.5064
    4.2374    7.9803    7.6499    5.5623    7.1377    7.1419    6.4169    6.8696
  -26.9452    9.5737    9.3567    8.8743    7.7975    7.7446    7.0923    7.1415
    3.8152    6.8771    6.1727    5.9180    5.8778    6.2619    5.9195    5.6903
  -26.6475    9.1840    8.9940    8.5383    7.5868    6.6525    6.5862    7.1747
    3.3640    7.0582    4.5301    6.1609    3.5153    5.1074    3.2495    4.8730
  -29.3081    5.9744    3.3381    5.3789    3.9624    4.0620    3.0034    3.6673
    2.9744    5.5075    4.4792    4.9996    3.0557    3.5737    2.8911    3.2622
    3.0321    6.6905    4.6489    5.6019    3.0646    4.7761    3.0004    4.2917
      ⋮
```

```
%view(trainedModel.ClassificationTree,'Mode','graph');
```

```
yfit = trainedModel.predictFcn(T1Test);
```

```
function [T1, T2, data] = getdataexercise2(listaNum)
    nDesc = 50;
```

```
    descriptorsComp = [];
    descriptorsMod = [];
    namesleaf = [];
    images = ["l2nr0", "l15nr0", "l4nr0"];
    fulles = ["Erable", "Faig", "Roure"];
    for k = 1:length(images)
        image = images(k);
        for i = listaNum
            if i < 10
                im = imread(strcat(fulles(k),'/',image ,'0', int2str(i) ,'.jpg'));
            else
                im = imread(strcat(fulles(k),'/',image ,int2str(i) ,'.jpg'));
            end
            [descriptorComp,descriptorMod] = descriptor_fourier(im, nDesc);
            descriptorsComp = [descriptorsComp; descriptorComp];
            descriptorsMod = [descriptorsMod; descriptorMod];
            namesleaf = [namesleaf; fulles(k)];
        end
    end
    data = descriptorsMod
    T1 = table(double(descriptorsComp), namesleaf);
    T2 = table(double(descriptorsMod), namesleaf);
    writetable(T1,'resultado1.csv');
    writetable(T2,'resultado2.csv');
end


function [descriptorDiv,descriptor] = descriptor_fourier(im, Ndescriptors)
    [d1 d2 d3] = size(im);
    if (d3 == 3)
        im = rgb2gray(im);
    end
    im =im2bw(im, graythresh(im)); % Binarització per Otsu
    im = imcomplement(im);
    im=imresize(im,1/16);
    % obtenim el contorn
    ero=imerode(im,strel('disk',1));
    cont=xor(ero,im);
    % obtenim les coordenades del contorn
    [fila col] = find(im,1); % Busquem el primer píxel
    B = bwtraceboundary(im,[fila col],'E'); %direccio est a l'atzar
    % B conté les coordenades
    % centrem coordenades
    mig=mean(B);
    B(:,1)=B(:,1)-mig(1);
    B(:,2)=B(:,2)-mig(2);
    % Convertim les coordenades a complexes
    s= B(:,1) + i*B(:,2);
    % Cal que la dimensio del vector sigui parell
    [mida bobo]=size(B);
    if(mida/2~=round(mida/2))
        s(end+1,:)=s(end,:); %dupliquem l'ultim
        mida=mida+1;
```

```
    end
    % Calculem la Fast Fourier Transform
    descriptor=fft(s);
    % Obtenim el
    if (~isnan(Ndescriptors))
        descriptor = [descriptor(1:Ndescriptors);descriptor(end-Ndescriptors+1:end)];
        descriptorDiv = [real(descriptor); imag(descriptor)];
    end
    descriptor = log(abs(descriptor));
    descriptor = descriptor';
    descriptorDiv = descriptorDiv';
end
```