2.3.1.1

**TEST**



**TRAIN**



2.3.1.2.

Sense transformar

error Linear train = 0.0608309

error Linear test = 0.0969817

CM\_Linear\_test =

652 0 1 1 3 2 11 0 5 0

0 546 0 0 1 1 2 0 0 2

5 0 340 16 13 2 4 3 19 2

4 0 5 327 2 8 0 3 7 2

2 15 7 0 317 1 6 0 6 16

7 1 4 18 7 257 3 0 11 3

4 2 3 0 9 2 340 0 3 0

0 1 0 3 7 0 0 289 2 42

5 2 4 14 13 9 3 1 252 5

1 1 0 1 12 0 0 10 2 330

error knn train = 0.0479723

error knn test = 0.0522019

CM\_knn\_test =

670 0 1 1 1 1 1 0 0 0

0 549 0 0 1 0 2 0 0 0

8 4 371 1 3 0 1 10 5 1

3 0 2 342 0 4 0 2 2 3

1 16 5 0 328 0 3 1 1 15

8 1 1 10 1 279 6 0 0 5

3 1 2 0 3 0 354 0 0 0

0 4 2 0 4 0 0 323 1 10

2 10 1 10 4 3 1 2 272 3

2 0 0 1 2 0 0 9 0 343

knn = 10

Visto los resultados de estas matrices podemos ver que:

- Para el clasificador lineal la mayor tasa de error se produce cuando tenemos que clasificar un 7 (muchas veces se decidirá por clasificar como si fuera un 9) y cuando tenemos que hacerlo para un 2 (a veces decide 3 y 8).

- Para el clasificador KNN el problema viene con el 4 (a veces decide un 1 y también un 9)

2.3.1.3

Principalmente, la energía de las transformaciones DCT se concentra en la componente continua (0,0). Dependiendo de cada carácter, se encuentra más o menos energía en el resto de componentes.

Un caso particular es el del dígito 1. Al estar formado en su práctica totalidad por líneas verticales, la energía se concentra en la fila superior de la transformada. Esta fila corresponde a las funciones base con líneas verticales. Los valores por los que la energía está más repartida parecen ser el 2 y el 3.

Una aplicación de la DCT seria la de eliminar las muestras de alta frecuencia. Estas muestras de la DCT contienen poca energía, de manera que se pierde poca información.



2.3.1.4

**Transformació DCT**

Elegir Dimensión Reducida

**Dim** = 8

error Linear train = 0.080366

error Linear test = 0.087333

CM\_Linear\_test =

658 0 1 5 1 1 4 0 5 0

0 547 0 1 1 0 1 0 2 0

3 3 343 6 12 2 8 2 22 3

2 0 4 321 0 11 0 6 12 2

4 11 8 0 316 0 3 2 5 21

9 0 3 16 7 265 3 0 5 3

1 2 5 0 4 4 344 0 3 0

0 2 1 0 6 0 0 300 3 32

4 7 3 16 7 3 1 0 263 4

0 1 0 1 13 0 0 8 2 332

error knn train = 0.0390702

error knn test = 0.0470064

CM\_knn\_test =

672 0 1 2 0 0 0 0 0 0

0 550 0 0 1 0 1 0 0 0

11 3 372 3 1 0 2 5 7 0

2 1 2 342 0 1 0 1 5 4

1 8 3 0 334 0 4 2 1 17

8 0 2 6 0 287 3 0 3 2

7 2 1 0 1 1 350 0 1 0

0 4 1 0 4 0 0 330 0 5

3 6 0 7 4 1 2 3 277 5

1 1 1 1 7 0 0 6 2 338

**Dim** = 4

error Linear train = 0.162216

error Linear test = 0.157595

CM\_Linear\_test =

599 39 3 4 4 1 9 0 16 0

0 546 0 0 1 1 1 0 2 1

4 29 309 18 18 2 5 0 16 3

2 15 2 302 1 14 0 12 3 7

0 42 6 0 304 0 1 2 1 14

4 5 1 36 7 238 4 0 13 3

4 18 3 0 2 1 333 0 2 0

0 18 0 2 3 0 0 274 2 45

5 87 3 5 10 4 1 0 189 4

0 22 0 1 11 1 0 10 1 311

error knn train = 0.0699802

error knn test = 0.0809005

CM\_knn\_test =

665 4 2 2 0 0 1 0 1 0

0 547 0 2 2 0 0 0 1 0

8 2 362 7 5 0 5 3 11 1

6 1 0 328 1 11 0 4 4 3

0 11 9 0 339 0 2 1 0 8

15 0 1 33 3 245 6 0 5 3

6 6 2 0 1 0 348 0 0 0

0 1 0 1 7 0 0 318 4 13

19 40 2 9 4 1 1 0 230 2

0 3 0 1 7 0 0 9 4 333

**Dim** = 2

error Linear train = 0.376113

error Linear test = 0.377041

CM\_Linear\_test =

498 54 39 30 16 0 3 1 34 0

6 525 4 1 13 0 2 0 1 0

70 55 170 25 3 2 69 0 8 2

29 32 2 187 44 34 0 10 2 18

11 118 9 40 98 13 4 2 39 36

29 19 1 40 30 166 0 0 24 2

12 27 45 0 0 3 276 0 0 0

0 26 0 1 2 0 0 229 1 85

63 59 7 11 46 3 0 0 105 14

1 31 0 5 22 0 0 30 4 264

error knn train = 0.313304

error knn test = 0.37234

CM\_knn\_test =

538 27 30 24 6 6 12 0 32 0

11 513 7 1 11 1 3 0 4 1

104 23 159 19 7 9 69 0 13 1

36 9 10 225 34 17 1 5 14 7

24 48 8 56 108 11 6 6 63 40

58 12 7 69 22 77 1 0 59 6

14 2 31 0 2 2 312 0 0 0

0 0 0 4 9 0 0 249 2 80

90 34 13 16 41 7 1 0 95 11

1 4 0 7 24 0 0 54 6 261

**Transformació Hadamard**

**Dim** = 8

error Linear train = 0.0776459

error Linear test = 0.08857

CM\_Linear\_test =

653 2 0 2 2 1 9 0 5 1

0 544 1 1 2 0 1 0 1 2

5 2 345 10 18 4 5 4 9 2

2 2 2 329 0 13 0 4 4 2

2 7 9 0 317 2 2 1 4 26

10 2 1 11 9 268 4 0 5 1

4 2 6 0 5 2 337 0 7 0

1 3 0 1 5 0 0 306 1 27

2 16 0 10 7 6 1 2 258 6

0 3 0 1 12 0 0 13 1 327

error knn train = 0.0430267

error knn test = 0.0566551

CM\_knn\_test =

665 1 4 1 0 1 2 0 0 1

0 549 1 0 2 0 0 0 0 0

8 8 365 2 4 0 3 8 6 0

0 3 3 342 0 5 0 2 1 2

1 7 3 0 335 0 3 1 0 20

11 1 2 9 2 275 8 0 1 2

6 1 1 0 5 0 350 0 0 0

0 6 0 0 3 0 0 326 1 8

3 11 1 8 3 2 2 3 273 2

0 7 0 1 8 0 0 6 2 333

**Dim** = 4

error Linear train = 0.181009

error Linear test = 0.18951

CM\_Linear\_test =

580 31 1 2 4 1 7 0 49 0

0 520 0 0 4 2 2 0 22 2

9 44 305 9 14 2 2 2 17 0

0 35 5 272 1 19 1 8 5 12

1 57 1 0 261 0 3 0 4 43

6 17 0 41 5 226 6 0 7 3

4 15 1 0 2 2 336 0 3 0

0 14 0 0 3 0 0 271 2 54

8 76 6 8 3 5 2 1 194 5

0 20 0 0 15 0 0 10 1 311

error knn train = 0.0870425

error knn test = 0.114547

CM\_knn\_test =

649 14 2 2 1 0 1 0 6 0

0 541 0 1 2 1 3 2 2 0

9 12 362 9 2 1 0 1 7 1

6 7 11 298 0 15 0 6 9 6

2 21 5 0 299 0 2 0 2 39

17 7 0 26 4 242 8 1 5 1

6 5 1 0 4 2 344 0 1 0

0 6 2 0 3 0 0 310 2 21

20 64 1 9 2 4 2 0 205 1

1 5 0 0 8 0 0 9 5 329

**Dim** = 2

error Linear train = 0.347676

error Linear test = 0.33523

CM\_Linear\_test =

544 50 16 18 9 3 4 2 29 0

6 527 5 2 3 0 2 1 6 0

36 46 255 8 9 2 40 0 8 0

23 24 4 193 51 13 0 26 5 19

8 74 20 33 93 51 6 9 12 64

34 14 3 51 26 162 6 0 5 10

6 23 38 1 0 1 294 0 0 0

1 28 0 3 4 0 0 241 4 63

55 79 14 15 28 4 0 3 107 3

1 15 0 12 15 2 0 32 9 271

error knn train = 0.256182

error knn test = 0.304057

CM\_knn\_test =

580 24 13 13 2 2 3 0 38 0

17 511 8 2 1 0 0 2 11 0

51 21 265 4 14 2 33 0 14 0

36 11 3 226 24 13 0 14 17 14

15 38 25 53 113 33 6 19 14 54

41 8 7 37 38 157 4 0 13 6

11 0 31 0 3 1 317 0 0 0

2 7 0 13 8 0 0 245 1 68

74 58 8 25 8 4 1 0 125 5

1 1 0 20 14 2 0 33 12 274

La transformación DCT con 4x4 pixeles es la más eficiente. Su error para la base de datos de test es 0.0809005 . Otras Combinaciones tambén resultan en error en test inferior al 10% como són DCT dimensión 8 y Hadamard dimensión 8 pero són menos eficientes computacionalmente.

2.3.1.5

Para k=1, el sistema está sobreentrenado, ya que asigna la clase del vector más cercano. Como se está trabajando con los datos de training, las clases ya son sabidas.

En training, los vectores que entran al sistema son los de training, por tanto, con k=1, el vecino más cercano a cada vector es él mismo, por tanto, la clase asignada siempre es la del vector mismo.

% Compute KNN for k=1:10

errors = zeros(10,2);

aux= zeros(10,2);

for k=1:10

for i=1:10

[aux(i,:)] = **compute\_knn**(1,4,k);

end

errors(k,:)= mean(aux,1);

end

plot(1:10, errors(:,1), 1:10, errors(:,2)), hold on

grid on

legend('Train Error', 'Test Error', 'best');

hold off

*compute\_knn*

aux\_errors = zeros(1,2);

knnclass = fitcknn(X\_train,Labels\_train,'NumNeighbors',K\_neig);

knn\_out = predict(knnclass,X\_train);

aux\_errors(K\_neig,1)=sum(Labels\_train ~= knn\_out)/length(Labels\_train);

fprintf(1,' error knn train = %g \n', aux\_errors(K\_neig,1))

knn\_out = predict(knnclass,X\_test);

aux\_errors(K\_neig,2)=sum(Labels\_test ~= knn\_out)/length(Labels\_test);

fprintf(1,' error knn test = %g \n', aux\_errors(K\_neig,2))

% Test confusion matrix

if i\_CM==1

CM\_knn\_test=confusionmat(Labels\_test,knn\_out)

end

[knn\_Pe] = aux\_errors;



2.3.2

Probabilidad error test:

h= [1,10,20,100] —> [0.4805 , 0.0727 , 0.0943 , 0.5532]

Probabilidad error train:

h= [1,10,20,100] —> [0 , 0.0032 , 0.0462 , 0.5406]

**SIMULAR METODE PARZEN ENTRE 10 I 20 (12,14,16,18) PROMITJANT CADA 5 I PLOTEJAR EL RESULTAT**

3.2

- k=1

error knn train = 0

CM\_knn\_train =

2 0 0 0 0 0 0 0

0 4 0 0 0 0 0 0

0 0 3 0 0 0 0 0

0 0 0 4 0 0 0 0

0 0 0 0 2 0 0 0

0 0 0 0 0 4 0 0

0 0 0 0 0 0 2 0

0 0 0 0 0 0 0 3

error knn test = 0.36

CM\_knn\_test =

2 0 0 0 0 0 0 0

0 3 1 0 0 0 0 0

0 0 1 0 1 1 0 0

0 2 0 1 0 1 0 0

0 1 0 0 2 0 0 0

0 0 0 0 0 3 0 0

0 0 0 0 0 0 2 1

0 0 0 0 1 0 0 2

* k=2

error knn train = 0.125

CM\_knn\_train =

2 0 0 0 0 0 0 0

0 4 0 0 0 0 0 0

1 0 2 0 0 0 0 0

0 1 0 3 0 0 0 0

0 0 0 1 1 0 0 0

0 0 0 0 0 4 0 0

0 0 0 0 0 0 2 0

0 0 0 0 0 0 0 3

error knn test = 0.28

CM\_knn\_test =

2 0 0 0 0 0 0 0

1 2 0 1 0 0 0 0

0 0 1 1 0 1 0 0

0 2 0 1 0 0 0 1

0 0 0 0 3 0 0 0

0 0 0 0 0 3 0 0

0 0 0 0 0 0 3 0

0 0 0 0 0 0 0 3

* k=3

error knn train = 0.208333

CM\_knn\_train =

2 0 0 0 0 0 0 0

0 4 0 0 0 0 0 0

1 0 2 0 0 0 0 0

1 1 0 2 0 0 0 0

0 1 0 0 1 0 0 0

0 0 0 0 0 4 0 0

0 0 0 1 0 0 1 0

0 0 0 0 0 0 0 3

error knn test = 0.56

CM\_knn\_test =

2 0 0 0 0 0 0 0

1 3 0 0 0 0 0 0

1 0 0 0 0 2 0 0

0 3 0 0 0 0 0 1

0 2 0 0 1 0 0 0

0 1 0 0 0 2 0 0

0 0 0 0 1 0 0 2

0 0 0 0 0 0 0 3

-k=4

error knn train = 0.291667

CM\_knn\_train =

1 0 0 1 0 0 0 0

0 4 0 0 0 0 0 0

0 0 2 0 0 0 0 1

0 3 0 1 0 0 0 0

0 0 0 1 1 0 0 0

0 0 0 0 0 4 0 0

0 0 0 1 0 0 1 0

0 0 0 0 0 0 0 3

error knn test = 0.44

CM\_knn\_test =

1 0 0 1 0 0 0 0

0 3 0 1 0 0 0 0

1 1 0 0 0 0 0 1

0 0 0 4 0 0 0 0

0 1 0 2 0 0 0 0

0 1 1 0 0 1 0 0

0 0 0 0 0 0 2 1

0 0 0 0 0 0 0 3

**Comentario:** Como hemos justificado con anterioridad en training cuando utilizamos k=1 al compararse consigo mismo el error de clasificación nos da 0. Las clases seleccionadas para trabajar han sido la 1,2,3,4,6,7,9 y 12. Analizando estas clases los mejores resultados se producen para k=2 donde el error de test es 0.28. Las clases que producen más error de clasificación para k=2 son la clase 3 y clase 4