Práctica 2

Reconocimiento automático de dígitos manuscritos

Visión por Computador

2019 – 2020 Q2

Ferran Velasco Olivera
Joaquín Gómez Sánchez

Índice

Características para la modelización de los dígitos

A continuación se describirán las características empleadas por el modelo final presentado y las características que se han probado pero han estado descartadas por ser poco descriptivas o quedar subsumidas en otras.

Características del modelo final

Observando los dígitos, se puede intuir que las características visuales que más diferencian imagenes de dígitos diferentes (y a su vez, acercan aquellas que representan el mismo dígito) son aquellas relacionadas con las concavidades, agujeros y forma general del dígito (Figura 1). Por lo que son estas las que se han seleccionado para el modelo presentado.

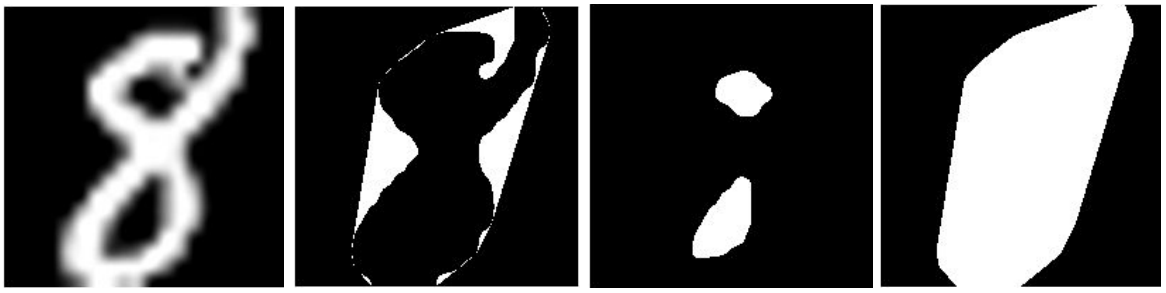

Figura 1: Por orden: imagen original del dígito, imagen de las concavidades, imagen de los agujeros e imagen de la convex hull.

Para cada concavidad, agujero y forma general del dígito (obtenida mediante la *convex hull*) se han extraído los siguientes parámetros:

- **Posición X e Y del centroide:** Describe la posición aproximada en la que se encuentra la concavidad, agujero o forma.
- **Área de las concavidades:** Describe el tamaño total de la concavidad, agujero o forma. Considerar el área en nuestro contexto es válido puesto que todas las imagenes estan normalizadas.
- **Tamaño del axis mayor y tamaño del axis menor:** Describe cual es la forma tendencial de la concavidad, el agujero o la forma, que puede aproximarse a una elipse más vertical o más horizontal.

Tras experimentar, como se detallará más adelante, se ha optado por considerar siempre 5 concavidades y 2 agujeros. El número de los agujeros se debe a que el máximo número posible es dos, en el dígito 8.

A simple vista se puede observar que considerar los agujeros (totalmente cerrados) es altamente discriminatorio ya que permite discernir entre 0, 4, 6 y 9 que únicamente presentan un agujero; el 8 que presenta dos agujeros; y el resto de números que no presentan ningún agujero. Es decir, se puede realizar una rápida clasificación en tres grupos.

Las concavidades son probablemente, tras los agujeros, la siguiente característica de los dígitos más discriminatoria puesto que se encuentran en posiciones muy exactas y no todos los números tienen la misma cantidad, ni del mismo tamaño.

Un ejemplo claro de ello son el 2 y el 9. Ciñéndonos únicamente a las concavidades ambos números presentan una en el lado izquierdo, pero cabría esperar que la del 2 sea siempre más grande que la del 9. Además, el 2 presenta otra concavidad en la izquierda.

Así pues, se dan un total de $5 \cdot 5$ características relativas a las concavidades, $2 \cdot 5$ características relativas a los agujeros, y 5 más relativas a la *convex hull* o forma general del dígito. En total, por cada imagen, se obtiene un vector de 40 características.

Para que estas características se correspondan entre distintas imágenes de números, dados los datos sobre las concavidades y/o agujeros de una imagen, estos se ordenan según el tamaño del área.

Para el caso en el que la imagen observada presente menso concavidades y/o agujeros que los que están considerados, se ha optado por rellenar dichas características con 0.

Además de esto, se eliminan aquellas concavidades y/o agujeros que no llegan a un mínimo de tamaño, por errores en la extracción.

Por último, para no confundir las concavidades con agujeros, se extraen primero los datos sobre los agujeros, y para calcular los datos relativos a las concavidades, se realiza el análisis sobre una imagen con los agujeros rellenados.

Características descartadas

A continuación se detallan las características de los dígitos que se han llegado a implementar pero finalmente han quedado descartadas por ser poco descriptivas o quedar subsumidas en otras.

Dichas características son las siguientes:

- **Número de agujeros**. Pese a que el número de agujeros es claramente una característica discriminadoras, esta queda recogida por la descripción de agujeros descrita anteriormente.
- **Área total de los agujeros**. La suma de las áreas de los agujeros no es una característica muy reveladora, además de ser una unión entre las áreas individuales de los agujeros, característica si empleada finalmente.
- **Área total del número (partes blancas tras la binarización)**. Puesto que el tamaño de los números está normalizado es de esperar que el número de pixeles (blancos) para un 1 no sea el mismo que el de un 9. Aun así, se ha descartado, puesto que es muy dependiente de por ejemplo el lápiz o bolígrafo empleado, o de la estilización de la tipografía. En definitiva, es una variable muy alterable por factores externos al dígito en sí.
- **Número de corners**. El número de corners en los número difiere, por ejemplo, en un 1 perfecto cabría esperar detectar dos corners con Harris en muchos caso, mientras que en un 7 serían tres. Esta característica ha quedado descartada por difícil de calcular con precisión en las imágenes del set MNIST.

Clasificadores empleados

Para el sistema de reconocimiento final, que se encuentra en el fichero `predictDigit.m` con el modelo `quadraticSVM.mat`, únicamente se emplea un único método de clasificación. Este se ha decidido a partir de diferentes pruebas con diferentes posibilidades.

Como se puede observar en los cuatro *scatter plots* de diferentes features (dos a dos) no parece haber ningún tipo de separación entre las diferentes clases, o al menos no con dos dimensiones.

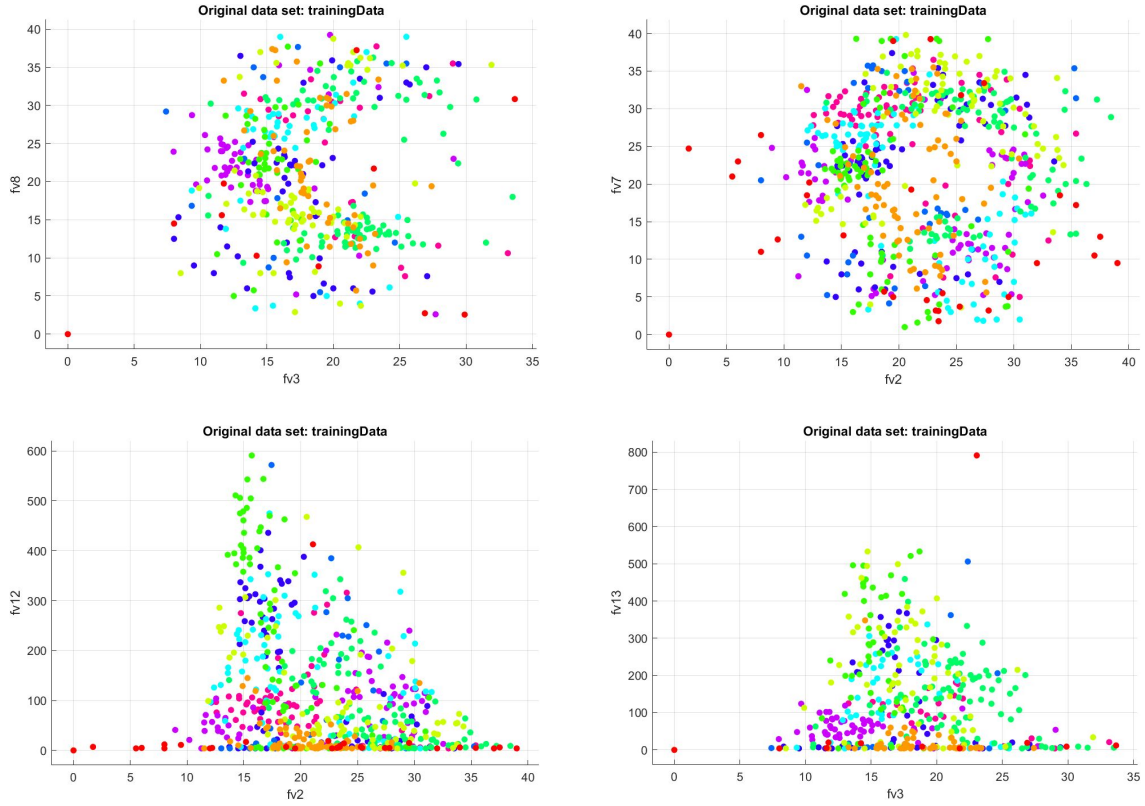

Figura 2: Scatter plots 2D de diferentes características

Esto nos da una idea de las limitaciones de los métodos lineales ante nuestro set de datos, ya que al trabajar con una dimensionalidad elevada (40 *features*) y con datos tan mezclados estos métodos no serán capaces de alcanzar ratios elevados en la clasificación, hecho que nos da a entender que métodos más complejos funcionarán mejor.

El método finalmente escogido ha sido la SVM Cuadrática, decisión tomada a raíz de la experimentación detallada más adelante.

Esta SVM Cuadrática permite realizar una separación entre clases más compleja, gracias a que la función de kernel puede “doblarse” una vez sobre sí misma.

Cabe destacar que aunque las SVMs están originalmente pensadas para la clasificación biclase, la implementación de Matlab para la clasificación multiclase, en nuestro caso, realiza una mejor clasificación que el resto de métodos probados.

Descripción de los experimentos

A continuación se describen los experimentos más relevantes y significativos para la decisión del modelo final.

Tamaño de la imagen del dígito

Todas las imágenes del set están estandarizadas con un tamaño de 20x20. A la hora de analizar las formas, concavidades y agujeros de los dígitos, este tamaño resulta demasiado pequeño para tener en cuenta ciertas peculiaridades.

Es por esto que se ha decidido experimentar con diferentes tamaños de las imágenes al extraer las diferentes características, para observar cuál es el tamaño que da un mejor resultado a la hora de clasificar.

Esto se hará haciendo un *resize* de las imágenes con diferentes tamaños. Para ello se experimentará con un subdataset de training de 100 imágenes y se estudiará los diferentes accuracy para cada tamaño, desde 20x20 (el tamaño original) hasta 100x100.

20x20	**30x30**	**40x40**
73%	70%	83%
50x50	**60x60**	**70x70**
76%	81%	78%
80x80	**90x90**	**100x100**
79%	81%	81%

Tabla 1: Accuracies obtenidos con SVM Cuadrática para diferentes tamaños

Aunque se puede observar que no existe un crecimiento sostenido en la accuracy de los modelos cuando se aumenta el tamaño de la imagen, si se puede ver que existe un considerable crecimiento en la accuracy en todos los tamaños respecto al original (exceptuando 30x30).

El mejor resultado lo da el tamaño 40x40, que será fijado para el resto de experimentos y para el modelo final.

Concavidades a considerar

Tras haber trabajado con las características definidas en el primer apartado, se ha observado que las variables correspondientes a las concavidades de los dígitos tienen un gran peso en el resultado de la clasificación. Es por esto que se ha decidido experimentar con estas para saber cuál es el mejor número de concavidades a considerar, ya que según el dígito se pueden observar más o menos concavidades.

Se ha querido observar el comportamiento del clasificador para un subset de training de 100 imágenes considerando como características aquellas variables de 1, 2, 3, 4, 5, 6 y hasta 7 concavidades. Esto implica que el número de variables consideradas para entrenar el modelo va incrementando en cinco con cada concavidad de más que se considera, puesto que cada concavidad lleva asociada cinco características (x e y del centroide, área y tamaños del eje mayor y menor).

Cabe esperar que cuantas más concavidades sean consideradas, mayor será la exactitud del modelo, aunque también será más alto el tiempo de cómputo y que a partir de cierto punto, no tiene sentido considerar más concavidades ya que la mejoría es mínima. Es este número óptimo de concavidades el que queremos encontrar para definirlo para nuestro modelo final.

Después de entrenar el modelo teniendo en cuenta los diferentes números de concavidades a considerar, hemos obtenido los siguientes resultados:

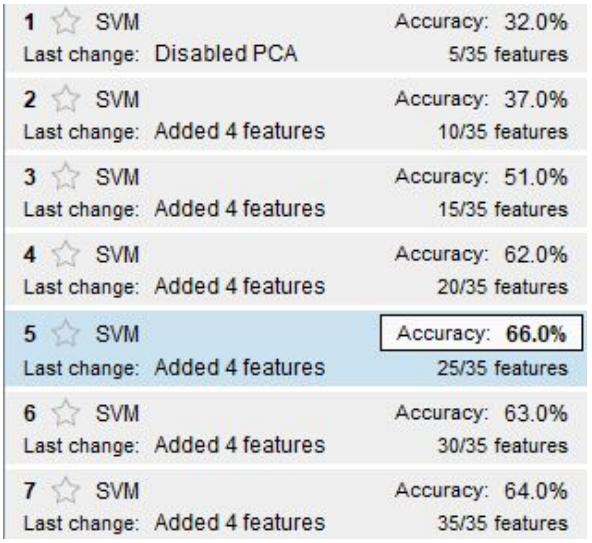

Figura 3: Resultados de la accuracy con diferentes números de concavidades consideradas

Así pues, se observa que es considerando 5 concavidades cuando el clasificador da un mejor resultado con un 66% de accuracy (teniendo en cuenta sólamente las características de las concavidades). Es por esto que para nuestro modelo final fijamos el número de concavidades a considerar en 5.

Significancia de variables

Teniendo en cuenta la estrategia que se ha decidido seguir hay tres tipos de características viéndolo a *grosso modo*: aquellas que corresponden a las concavidades, las relativas a los agujeros y las correspondientes a la forma general del dígito (convex hull). Considerando esto, es preciso saber cuáles son aquellas que tienen más importancia a la hora de discriminar entre las clases de dígitos en el modelo final.

Para realizar un estudio sobre esto, se observará el comportamiento del clasificador para un subdataset de training de 100 imágenes considerando sólo las variables correspondientes a cada grupo y las combinaciones con 2 grupos. Así se podrá observar cuál es el tipo de variables que tiene una mayor fuerza discriminatoria en el modelo final.

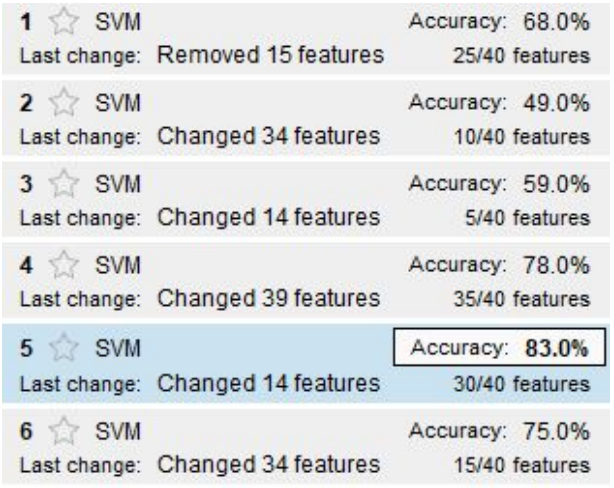

Figura 4: Resultados de la accuracy con diferentes consideraciones

En los resultados vemos 6 modelos diferentes. Estos se corresponden con las siguientes consideraciones:

1. Solo características sobre concavidades
2. Solo características sobre agujeros
3. Solo características sobre la forma general
4. Características sobre concavidades y agujeros
5. Características sobre concavidades y la forma general
6. Características sobre agujeros y la forma general

Tras estudiar estos resultados, se pueden asumir que aquellas variables que tienen una mayor significancia, o que permiten realizar una mejor discriminación entre clases, son aquellas relacionadas con las concavidades, seguidas de las relacionadas con la forma general del dígito y por último las menos influyentes a la hora de discriminar son las relacionadas con los agujeros.

Resultados obtenidos

A continuación se detallan los resultados obtenidos con el las características empleadas y el modelo final seleccionado.
Los resultados expuestos a continuación se han obtenido mediante un set de test con 2.000 muestras. (20% del set total, de tamaño 10.000).

Se han probado valores diferentes de k para realizar el k-fold Cross Validation. En concreto se han probado 5, 10 y 15, dando respectivamente resultados de accuracy en training de 93,8%, 93,9% y 93,9%. Esto invita a asumir que el modelo es igual de fiable con todos estos valores de k y es por eso que se ha elegido k = 10 para los resultados de test finales.

Como se puede observar en la matriz de confusión a continuación, en términos generales, todos los ejemplos de todas las clases han estado muy bien clasificados. Tal vez, la clase con peor clasificación es la del 9, que ha sido clasificada como 4 hasta en 10 ocasiones. Aun así cabe mencionar que no es realmente relevante puesto que en muchos casos la diferencia entre un 4 y un 9 es muy sutil.

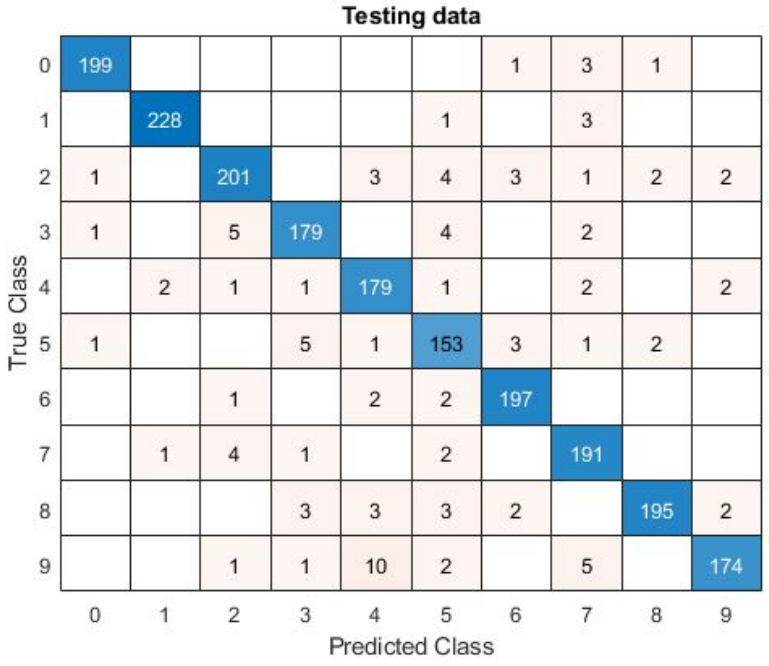

Figura 5: Matriz de confusión del sistema final

Como se puede observar en la tabla a continuación, en términos generales, tanto el accuracy, el recall como la precisión presentan valores remarcables, y considerablemente altos.

Accuracy	***Recall***	***Precision***
94.80%	94.68%	94.70%

Tabla 2: Accuracy, *recall* y *precision* del modelo final

Haciendo un análisis del recall y la *precision* de cada clase podemos observar, en la tabla a continuación, que las clases que mayor precision y recall presentan son la del 0 y 1. Esto entra dentro de lo esperado, puesto que dadas las características empleadas para la clasificación, tanto el 0 como el 1 son dígitos con poca complejidad.

Es decir, el 0 por ejemplo siempre presentará un agujero y ninguna concavidad, por lo que incluso en escrituras complejas será fácil de identificar. El 1 es también sencillo: no presenta ningún agujero y según como se represente ninguna concavidad, o una muy pequeña, además de una forma general del dígito muy clara en cualquier representación.

La clase claramente más perjudicada es el 5 que presenta un recall de 88.95%, es decir, la proporción de 5 clasificados como tal no alcanza un 90%.

Dígito	***Recall***	***Precision***
0	98.51%	97.55%
1	98.70%	98.28%
2	94.37%	92.63%
3	94.21%	93.72%
4	90.40%	95.21%

5	88.95%	92.17%
6	95.63%	97.52%
7	91.83%	95.98%
8	97.50%	93.75%
9	96.67%	90.16%

Tabla 3: Recall y *precision* de las diferentes clases en el modelo final

Funciones empleadas

La **funciones** empleadas para el desarrollo de la presente práctica, y que han sido **desarrolladas únicamente por nosotros**, sin hacer uso de fuentes externas, son las siguientes.

Para la **obtención** del **vector de características de cada imagen**:

- `featuresVector`: Dada una imagen de entrada obtiene las diferentes características de esta y las devuelve en formato tabla.
- `concavitiesFeatures`: Realiza la detección de un máximo de 5 concavidades por imagen, obteniendo para cada concavidad su centroide, su área y su tamaño. En caso de detectar menos concavidades registra ceros para poder obtener un set de entrenamiento coherente.
- `holesFeatures`: Realiza la detección de un máximo de 2 agujeros por imagen, obteniendo para cada agujero su centroide, su área y su tamaño. En caso de detectar menos agujeros registra ceros para poder obtener un set de entrenamiento coherente.
- `convexHullFeatures`: Calcula centroide, área y tamaño en los ejes de la envolvente convexa del número en la imagen.

Para la **verificación el set de test y la obtención de métricas**:

- `testModel`: Dado un modelo y un set de datos de test devuelve la predicciones y muestra la matriz de confusión, la *precision* y el *recall* general, la *precision* y el *recall* de cada clase, el *average*, y la *F-Score*.

El resto de funciones empleadas son **funciones MATLAB**, es decir, no se ha realizado ningún uso de software o funciones diferentes de las proporcionadas en el set estándar de toolboxes de MATLAB.

Para la **lectura del set de datos** se ha empleado la función proporcionada junto al enunciado, `readMNIST`. Para la **división del set en sets de entrenamiento y test** se ha empleado la función `cvpartition`.

En cuanto al **tratamiento de las imágenes** y la **obtención de información** se ha empleado las funciones: `imresize`, para redimensionar las imágenes; `im2bw`, para la binarización de las imágenes; `imfill`, para rellenar agujeros de los dígitos; `bwconvhull`, para obtener la envolvente converxa de los números, `bwlabel`, para etiquetar regiones en las imágenes; y `regionsprops`, para obtener propiedades sobre las regiones en la imagen;

Para la gestión de datos se ha utilizado: `struct2table` y `table2struct`, para la conversión entre tabla y estructura; `sortrows`, para ordenar las filas de las tablas; y `zeros`, para instanciar estructuras de datos.

Finalmente, para la obtención de métricas sobre las predicciones: `confusionchart` y `confusionmat`, para obtener la matriz de confusión.

Conclusiones

La principal conclusión que podemos extraer del presente trabajo, a partir de todo lo detallado anteriormente, es que las características escogidas han estado acertadas desde el punto de vista de la alta exactitud del modelo obtenido.

Podemos concluir además que para la clasificación de dígitos manuscritos, aunque podría ser interesante, no es necesario recurrir a técnicas más complejas o más aproximadas para la detección de dígitos.

Es decir, se puede conseguir un buen sistema de reconocimiento sin emplear descripciones de los dígitos más directas, tales como HoGs o Descriptores de Fourier. Con descriptores “indirectos” como los empleados es suficiente para conseguir buenos ratios.

PRÁCTICA 2 - RECONOCIMIENTO AUTOMÁTICO DE DÍGITOS MANUSCRITOS

Data

Data loading

```
% Data parameters
imgFile = "train-images-idx3-ubyte";
labelFile = "train-labels-idx1-ubyte";
readDigits = 100;%10000;
offset = 0;

[imgs labels] = readMNIST(imgFile, labelFile, readDigits, offset);

% Setting random numbers seed
rng(0, 'twister');
```

Features vectors generation & data set construction

```
featuresVectorsOfImgs = [];

% Features vectors generation
for i = 1:readDigits
    fv = featuresVector(imgs(:, :, i));
    featuresVectorsOfImgs = [featuresVectorsOfImgs ; array2table(fv)];
end

featuresVectorsOfImgs.Labels = labels;
```

Split of set in training set (80%) and test set

```
% Trainin data: 80% ; Test data: 20%
p = cvpartition(readDigits, "HoldOut", 0.2);
i = p.test;

% Splitting data
trainingData = featuresVectorsOfImgs(~i, :);
testData = featuresVectorsOfImgs(i, :);
```

Testing ML Models

```
% Exported models: quadraticSVM
load quadraticSVM.mat

testModel(quadraticSVM, testData)
```

Functions

Features vector generation function

```
function features = featuresVector(img)
    img = imresize(img, [40, 40]);      %????
    img = im2bw(img, graythresh(img));

    features = [];

    % Concavities
    cf = concavitiesFeatures(img);
    features = [features cf];

    % Holes
    hf = holesFeatures(img);
    features = [features hf];

    % Convex Hull
    chf = convexHullFeatures(img);
    features = [features chf];
end
```

Feature extraction functions

Concavities

The extracted concavities features are (for each concavity): centroid_x, centroid_y, area, major_axis_lenght, minor_axis_lenght.

```
function cf = concavitiesFeatures(img)
    imgWithoutHoles = imfill(img, 'holes');

    ch = bwconvhull(imgWithoutHoles, 'objects');

    concavities = ch-imgWithoutHoles;

    [eti num] = bwlabel(concavities,4);

    Dades = regionprops(eti,'all');

    N = 5;
    if size(Dades, 1) > 1
        T = struct2table(Dades);
        sortedT = sortrows(T, 'Area', "descend");
        Dades = table2struct(sortedT);
        Dades = Dades(1:min(size(Dades, 1), N));
    end

    centroidsX = zeros(1, N);
    centroidsY = zeros(1, N);
    area = zeros(1, N);
    majorAxisLength = zeros(1, N);
    minorAxisLength = zeros(1, N);

    for i=1:size(Dades, 1)
        dada = Dades(i);
```

```
        centroidsX(1, i) = dada.Centroid(1);
        centroidsY(1, i) = dada.Centroid(2);
        area(1, i) = dada.Area;
        majorAxisLength(1, i) = dada.MajorAxisLength;
        minorAxisLength(1, i) = dada.MinorAxisLength;
    end

    cf = [centroidsX centroidsY area majorAxisLength minorAxisLength];
end
```

Holes

The extracted holes features are (for each hole): centroid_x, centroid_y, area, major_axis_lenght, minor_axis_lenght.

```
function hf = holesFeatures(img)
    imgWithoutHoles = imfill(img, 'holes');

    holes = imgWithoutHoles-img;

    [eti, num] = bwlabel(holes,4);

    Dades = regionprops(eti,'all');

    N = 2;
    if size(Dades, 1) > 1
        T = struct2table(Dades);
        sortedT = sortrows(T, 'Area', "descend");
        Dades = table2struct(sortedT);
        Dades = Dades(1:min(size(Dades, 1), N));
    end

    centroidsX = zeros(1, N);
    centroidsY = zeros(1, N);
    area = zeros(1, N);
    majorAxisLength = zeros(1, N);
    minorAxisLength = zeros(1, N);

    for i=1:size(Dades, 1)
        dada = Dades(i);
        centroidsX(1, i) = dada.Centroid(1);
        centroidsY(1, i) = dada.Centroid(2);
        area(1, i) = dada.Area;
        majorAxisLength(1, i) = dada.MajorAxisLength;
        minorAxisLength(1, i) = dada.MinorAxisLength;
    end

    hf = [centroidsX centroidsY area majorAxisLength minorAxisLength];
end
```

Convex hull

The extracted holes features are: centroid_x, centroid_y, area, major_axis_lenght,

minor_axis_lenght.

```
function chf = convexHullFeatures(img)
    ch = bwconvhull(img, 'objects');

    [eti num] = bwlabel(ch,4);

    dada = regionprops(eti,'all');

    % si s'extrau més d'un objecte que agafi el més gran
    % ja que serà aquell que representi la convex hull
    if size(dada, 1) > 1
        T = struct2table(dada);
        sortedT = sortrows(T, 'Area', "descend");
        dada = table2struct(sortedT);
        dada = dada(1);
    end

    centroidsX = dada.Centroid(1);
    centroidsY = dada.Centroid(2);
    area = dada.Area;
    majorAxisLength = dada.MajorAxisLength;
    minorAxisLength = dada.MinorAxisLength;

    chf = [centroidsX centroidsY area majorAxisLength minorAxisLength];
end
```

Test model & metrics

For a given model and test set shows the confusion matrix, the accuracy, the recall and the precision of each class and general.

Also shows the F-Score and returns the prediction.

```
function predicciones = testModel(model, testSet)
    % Predicciones en test
    predicciones = model.predictFcn(testSet);

    % Extracción de información en test
    figure, confussionTest = confusionchart(testSet.Labels,predicciones), title('Testir

    % METRICS
    % Confusion matrix
    [confMat, order] = confusionmat(testSet.Labels,predicciones);

    % Accuracy
    accuracy = 0;
    for i =1:size(confMat,1)
        accuracy=confMat(i,i) + accuracy;
    end
    Accuracy = accuracy/size(predicciones,1)
```

```
    % Recall
    for i =1:size(confMat,1)
        recall(i)=confMat(i,i)/sum(confMat(:,i));
    end
    recall(isnan(recall))=[];

    recall
    Recall = sum(recall)/size(confMat,1)

    % Precision
    for i =1:size(confMat,1)
        precision(i)=confMat(i,i)/sum(confMat(i,:));
    end
    precision
    Precision = sum(precision)/size(confMat,1)

    % F-score
    F_score=2*Recall*Precision/(Precision+Recall)
end
```