UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

Pràctica 2
Reconeixement Automàtic de dígits manuscrits

Jesús Molina Roldán
Victor Vidal Rojas Condori

INDICE

Introducción

En este proyecto, intentaremos realizar un estudio del funcionamiento de distintos clasificadores para reconocer dígitos en una serie de imágenes. Haremos un estudio de los descriptores que podemos extraer de las imágenes y que clasificadores los utilizan mejor.
Al final mostraremos el clasificador que mejores resultados nos ha dado y nuestras conclusiones.

Descripción de los descriptores

Fast-Fourier

La transformada de Fourier es una transformada matemática que descompone una función (normalmente una función de tiempo, o de señal) al espacio de frecuencias. Para obtener los features a partir de la transformada de fourier cogemos la imagen principal y obtenemos los contornos de la imagen. Una vez tenemos la imagen cogemos las coordenadas del contorno para aplicar la transformada de fourier y así obtener esos datos en el espacio de frecuencias. El resultado como sabemos nos da un número complejo por lo que para obtener un único número cogemos el módulo del número complejo. De todas las frecuencias cogemos las n más significativas. Este descriptor es invariante a la rotación y a la translación. En contraposición es variante al escalado, haciendo más grande o más pequeño los factores.

$$f(x) = \int_{-\infty}^{\infty} F(u) e^{2\pi i u x} dx$$

HOG Feature(Histogram of oriented gradients).

El histograma de orientación del gradiente consiste en dividir la imagen en pequeñas regiones conectada, llamadas celdas, y por cada celda compilar un histograma de direcciones de gradientes por cada píxel dentro de la casilla. La combinación de todos estos histogramas representa el descriptor. Es invariante a la rotación , ya que una rotación de la imagen solo implicaría un desplazamiento del histograma. También es invariante al escalado, para ello existen varios métodos como SIFT (Scale-invariant feature transform) o la técnica de ir comparando histogramas a partir de la imagen a diferentes escalas. También es invariante al

trasladado. Aún así para la extracción de features ,en nuestra práctica, el escalado y la rotación es variante.

K-Means

Normal K-Means

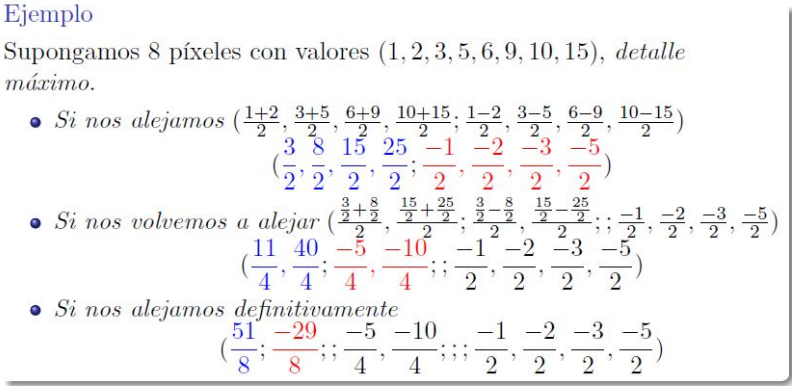
Utilizaremos K-means para encontrar los 8 puntos que representen mejor la forma de cada imagen. Una vez seleccionados los puntos de manera aleatoria debido a K-means, los ordenaremos respecto al eje X, de esta manera reducimos el error de la aleatoriedad ya que es posible que 2 imágenes del mismo número generen centros similares pero distribuidos de manera distinta, lo que reducirá el acierto.

K-Means Ordered (Descartado)

En un intento de mejorar las features recogidas por K-means, hemos decidido hacer un preprocesado de los puntos.
Primero cogemos 10 imágenes, una por número, de las cuales extraemos los centros representativos y los ordenaremos.
Una vez acabado lo que haremos es que por cada imagen, calcularemos los centros y compararemos todas las permutaciones posibles de los centros para encontrar la más parecida a los centros representativos, es decir, realizamos un tipo de object matching.
Esta manera es menos eficaz y lenta pero conseguimos mejorar el porcentaje de acierto.
El problema más importante es que ya realizamos una pequeña preselección de los números, lo cual no tiene sentido al extraer los centros ordenados en las imágenes de testeo. Por eso nos decantamos por la feature anterior aunque, a primera vista, da peor resultado.

Wavelet scattering

Wavelet scattering es un tipo de transformada matemática en la que representa una señal en términos de versiones trasladadas y dilatadas de una onda finita. Es muy parecido a la transformada de Fourier ya que representamos una señal en combinación de otras señales. Si consideramos una señal por ejemplo X, esta señal la descompone a partir de dos señales, la mother wavelet(wavelet function) y la father wavelet(scaling function). Las funciones que forman el espacio de las father wavelet se vuelven a descomponen otra vez a partir de la función señal mother y father. Aquí podemos ver un ejemplo numérico que representa la idea de wavelet.



Ejemplo

Supongamos 8 píxeles con valores $(1, 2, 3, 5, 6, 9, 10, 15)$, *detalle máximo.*

- *Si nos alejamos* $(\frac{1+2}{2}, \frac{3+5}{2}, \frac{6+9}{2}, \frac{10+15}{2}, \frac{1-2}{2}, \frac{3-5}{2}, \frac{6-9}{2}, \frac{10-15}{2})$
  $(\frac{3}{2}, \frac{8}{2}, \frac{15}{2}, \frac{25}{2}; \frac{-1}{2}, \frac{-2}{2}, \frac{-3}{2}, \frac{-5}{2})$
- *Si nos volvemos a alejar* $(\frac{\frac{3}{2}+\frac{8}{2}}{2}, \frac{\frac{15}{2}+\frac{25}{2}}{2}; \frac{\frac{3}{2}-\frac{8}{2}}{2}, \frac{\frac{15}{2}-\frac{25}{2}}{2}; ; \frac{-1}{2}, \frac{-2}{2}, \frac{-3}{2}, \frac{-5}{2})$
  $(\frac{11}{4}, \frac{40}{4}; \frac{-5}{4}, \frac{-10}{4}; ; \frac{-1}{2}, \frac{-2}{2}, \frac{-3}{2}, \frac{-5}{2})$
- *Si nos alejamos definitivamente*
  $(\frac{51}{8}; \frac{-29}{8}; ; \frac{-5}{4}, \frac{-10}{4}; ; ; \frac{-1}{2}, \frac{-2}{2}, \frac{-3}{2}, \frac{-5}{2})$

Matlab utiliza la descomposición de Gabor wavelet para hacer el proceso explicado anteriormente. Realmente la transformada de wavelet funciona mejor para procesamiento de señales principalmente en el campo de la compresión de imágenes, en la transcripción de música o en datos donde se presente una relación entre frecuencia y tiempo. Aún así nosotros hemos realizado la transformada de wavelet para la imagen completa en escala de grises y esa descomposición será la que pasaremos como features a los algoritmos de machine learning. Como características podemos decir que es variante a la rotación, traslación y escalado. Aún así si el cambió no es muy grande puede funcionar correctamente como hemos podido observar en el estudio.

Harris Corner

Harris Corner es un algoritmo utilizado para extraer las esquinas y algunas características de la imagen. En este caso, utilizaremos Harris para encontrar el números de esquinas que tendría cada número. Sabemos que la cantidad puede producir muchos falsos positivos pero consideramos que es lo suficientemente diferenciador para incluirlo como feature junto a otros.

Propiedades Geométricas

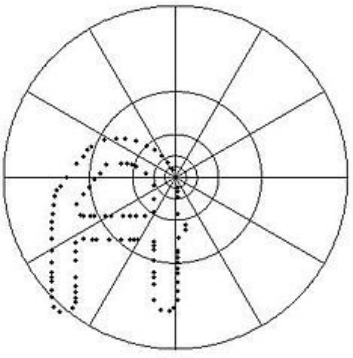
Para extraer las features hemos considerados varios descriptores basados en región. El primero que hemos realizado nos calcula el área total de la figura.
A partir de la área que engloba el total de los objetos (área tras realizar convex hull) le restamos el área del número para obtener otro feature. Finalmente para aumentar la probabilidad de acierto en la búsqueda de números como el 8 obtenemos el número total de agujeros que tiene la figura como diferenciador.

All Points (Descartado)

Esta feature consistirá en un solo vector que contendrá la lista de píxeles de la imagen. Para eso, es necesario que todas las imágenes tengan el mismo tamaño y, si es necesario, aplicar un resize. Es una idea bastante simple pero curiosamente no da unos resultados muy negativos, llegando a alcanzar un máximo de 80% de acierto en función del clasificador.

Shape Context Multicenter (Descartado)

Los descriptores Shape Context consiste en, dado un centro y una distribución de zonas con forma de rueda, etiquetar los puntos del contorno de una figura en función de su posición. Luego contabilizaremos el número de puntos por zona y utilizaremos ese dato como descriptor. Para este descriptor, seleccionaremos los centros de manera manual.



Tras realizar diversas pruebas, hemos descartado diversas features ya sea por dar malos resultados en general o que los métodos de extracción de dicha feature, no nos convencía del todo.

Entrenamiento clasificadores

En este apartado, hemos decidido entrenar nuestros descriptores más sobresalientes con diversos clasificadores ofrecidos por Matlab.

Estudiaremos, tanto descriptores por separado como combinaciones de descriptores para comprobar cuales nos podrán ofrecer mejores resultados.

Al final de todo, también explicaremos un pequeño experimento acerca de redes neuronales que no pudimos probar de manera exhaustiva pero que, a simple vista, daba resultados bastantes prometedores.

Para el entrenamiento de los clasificadores probamos distintos clasificadores con distintos valores:

- **Decision Tree**: para el clasificador decision tree usamos siempre el mismo criterio de división del árbol (gdi), lo único que variamos era el número máximo de divisiones que hacía el árbol, con los valores 100, 20 y 4.

- **Discriminant Analysis Models**: para el clasificador discriminant Analysis models probamos dos tipo de discriminantes, cuadrático y lineal.

- **Naive Bayes**: Para la clasificación bayesiana probamos la distribución normal y realizar la clasificación bayesiana a partir de una estimación de densidad Kernel(KDE).

- **Support Vector Machine(SVM)**: para el clasificador SVM utilizamos como función kernel dos posibles valores, gaussiana y polinomial. Cuando la función de transformación era polinomial testeamos con tres posibles grados de polinomio, lineal, cúbica y cuadrática. En contraposición si la función de transformación era gaussiana entonces lo único que modificamos era el escalado que hacía el kernel por los valores 1.8, 7 y 28.

- **k-Nearest Neighbor**: para el clasificador KNN utilizabamos 10 neighbors. Lo único que modificaremos era la medida de distancia para clasificar: distancia coseno, distancia euclídea, distancia minkowski y la inversa de la distancia euclídea.

- **Ensemble classification**
  Utilizamos ensemble classification ya que algunos clasificadores son considerados débiles, dado dos conjuntos ligeramente distintos nos puede dar arboles totalmente diferentes. Por ello cogemos un conjunto de

clasificadores entrenados “Ensemble” y nos quedamos con la ponderación todos los clasificadores obtenidos.
Según el conjunto de clasificadores que usamos, podemos hablar de:

- Tree ensemble classification: en este caso el conjunto ensemble está formado por árboles. En nuestro experimento utilizamos tres métodos de ensamblar Bag (Bootstrap aggregation), Adaptive boosting multiclass y Random undersampling boosting.
- Discriminant ensemble classification: en este caso el conjunto ensemble está formado por modelos discriminante de análisis. El método de ensamblar que usamos se llama “Random subspace method”.
- KNN ensemble classification: el grupo de ensemble estaba compuesto por K-Nearest Neighbor classifiers. El método que utilizamos para ensamblar fue “Random subspace method”.

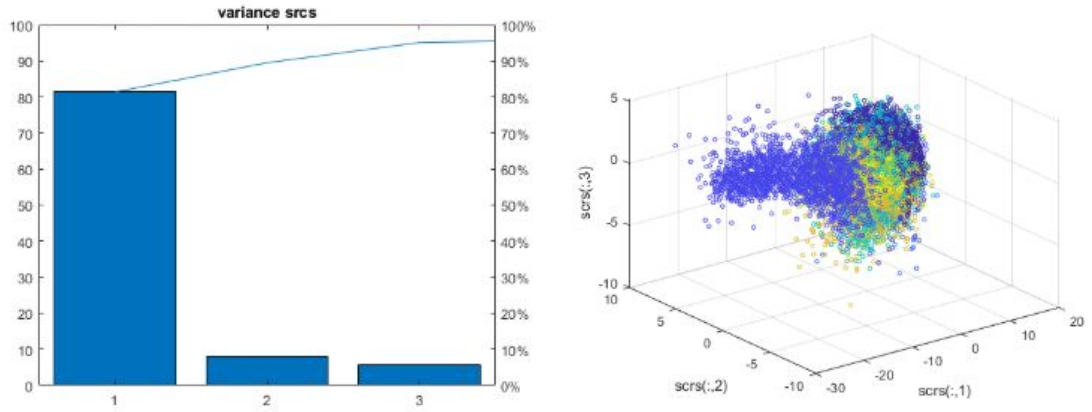
Cada uno de los descriptores y la unión de varios descriptores para formar un único descriptor, eran probados por el conjunto clasificadores. Para evitar el overfitting de los datos intentamos variar el número de Cross-Validation folds entre los valores 2,5 10 y 50.

Los features siempre eran normalizados y en algunas ocasiones se utilizaba la transformación Principal Component Analysis para hacer una reducción significativa de la dimensionalidad de los features.

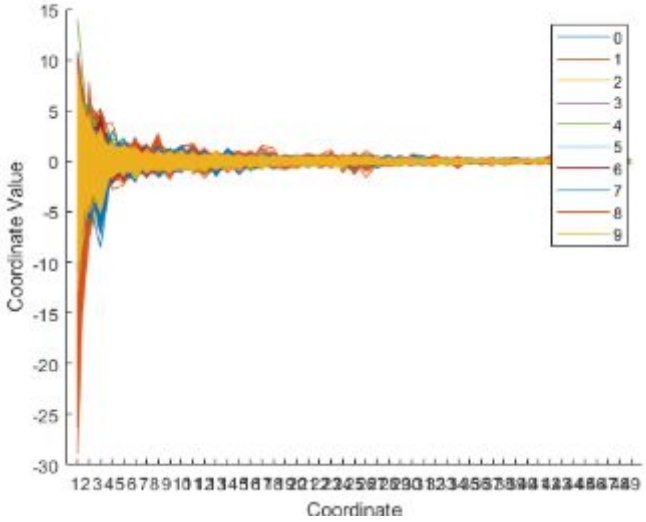
Resultados Clasificadores

Experimentos con Wavelet Scattering

Estudio de los datos



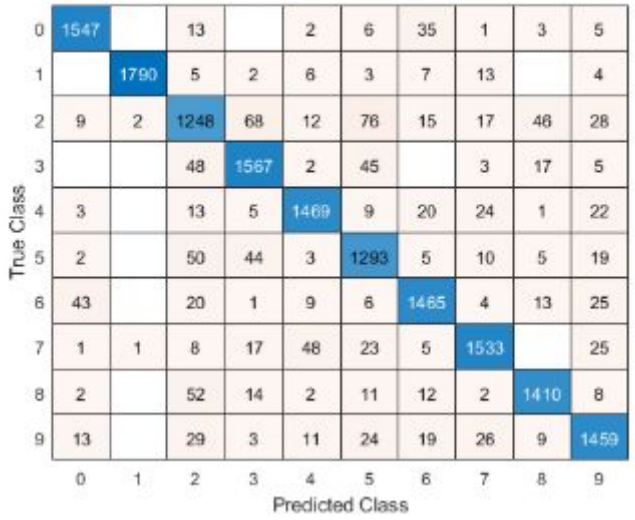
Aquí podemos observar como hay una variable que provoca gran parte de la varianza en los predictores. Este resultado lo obtenemos gracias a aplicarle la técnica PCA sobre los predictores. Gracias a esto podemos descartar predictores que no aportan mucho a la diferenciación de los datos.



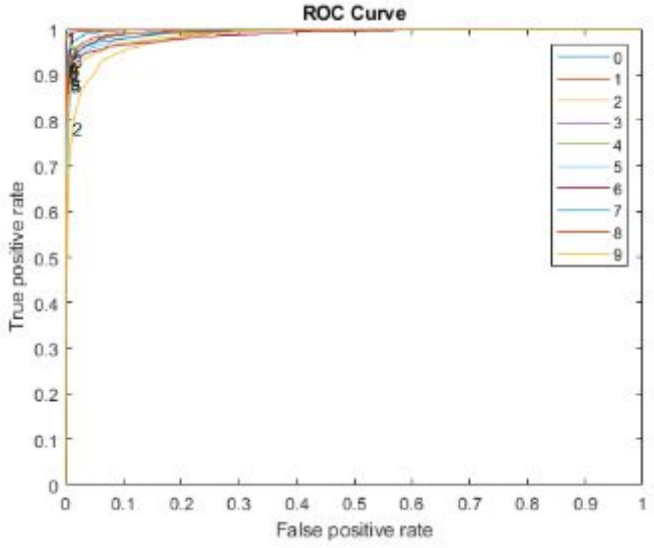
Una vez realizada la técnica PCA, este gráfico nos muestra como se ha reducido la dispersión de los predictores, lo que nos facilitará y mejorará el entrenamiento del clasificador.

Confusion Matrix y ROC Curve

```
maxAcc = 0.8091
```

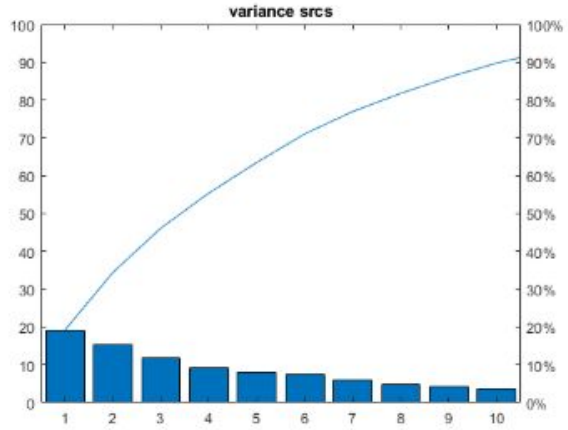


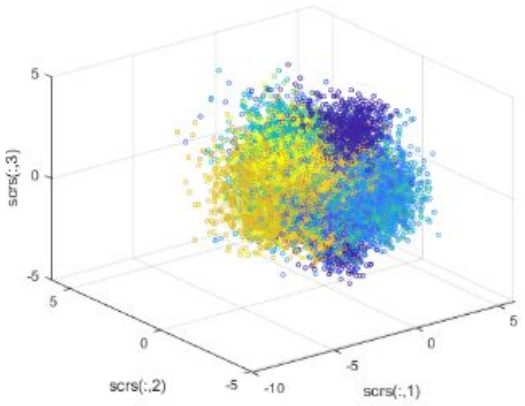

```
accuracy = 0.8091
    "QuadraticSVM"
```



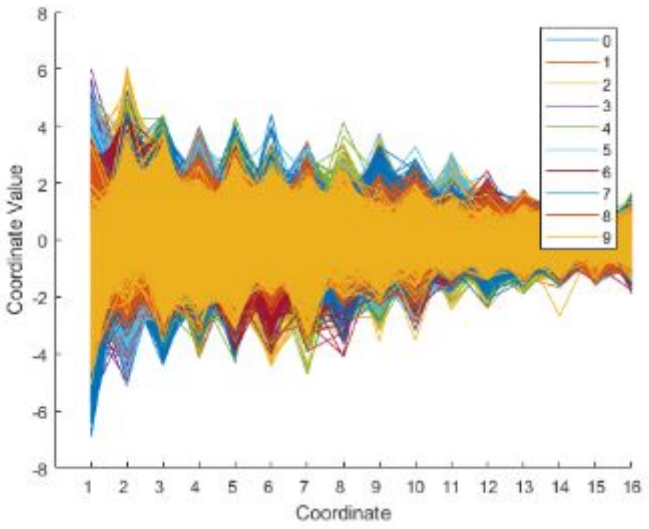
La mejor precisión lo obtenemos con el clasificador SVM con función de transformación polinomial cuadrática con casi un 80%. También se puede observar tanto en la matriz de confusión como en la curva ROC, que el mayor fallo suele recaer en el número 2, llegando a confundir mayoritariamente por el 5 u 8.

K-means



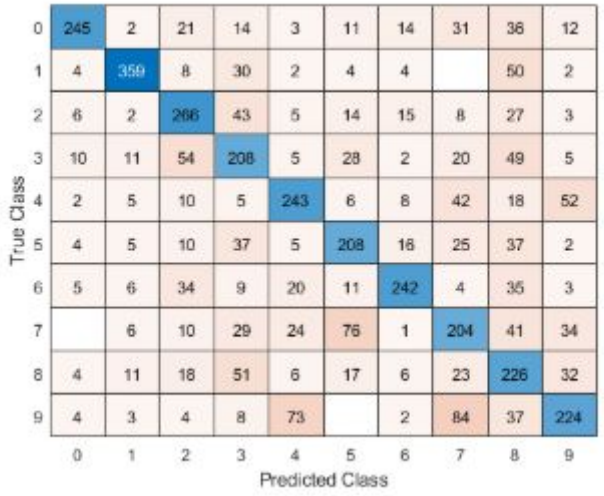


En este caso, la varianza está bastante distribuida entre los distintos predictores, por lo tanto, la técnica PCA no nos ofrecerá una mejora significativa.

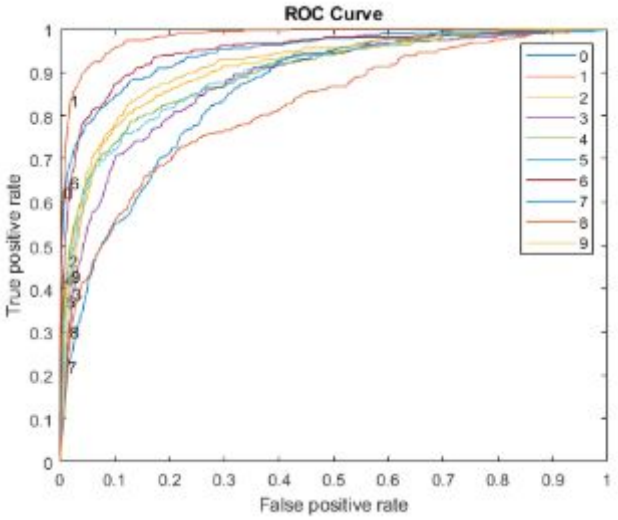


Aquí vemos como la mejora en la dispersión de los descriptores es mínima, algo lógico teniendo en cuenta que los predictores en este caso son puntos que representan una forma, la supresión de alguno puede cambiar por completo dicha interpretación.

Confusion Matrix y ROC Curve



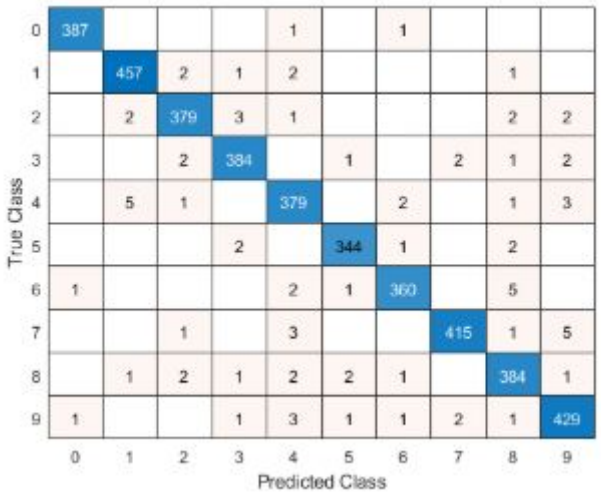

```
accuracy = 0.9073
"CubicSVM"
```



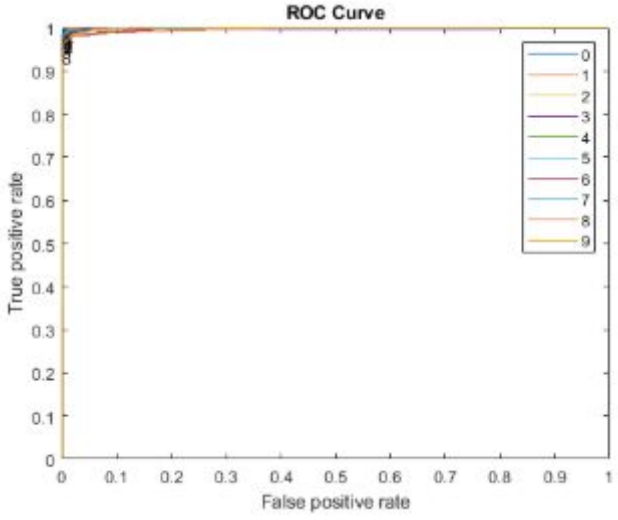
Aquí comenzamos a notar un patrón y es que el clasificador SVM nos dá modelos bastantes precisos para este problema aunque, en este caso en su variante cúbica. En el pasado, este descriptivo no tenía resultados tan prometedores pero al realizarle un pre estudio de los datos (PCA), hemos conseguido mejorar considerablemente su precisión.

HOG

Confusion Matrix y ROC Curve




```
accuracy = 0.9817
    "QuadraticSVM"
maxAcc = 0.9817
```

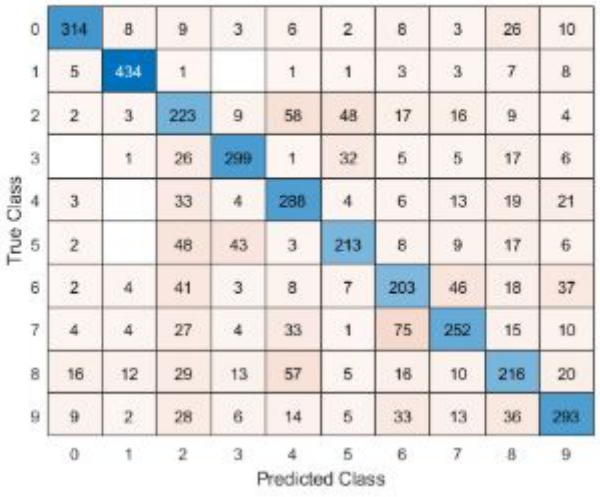


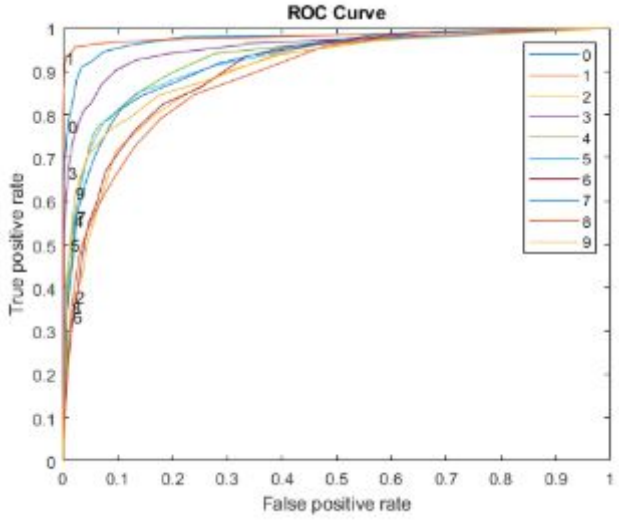
Seguimos la tendencia de los anteriores. Los clasificadores SVM han obtenido resultados bastantes buenos con los predictores de HOG. Por si mismo, Los descriptores de HOG son suficientes para obtener un modelo con una nivel alto de precisión. No sabemos hasta qué punto, el combinarlo con otros, puede afectar su precisión.

Fourier

Confusion Matrix y ROC Curve

```
accuracy = 0.6851
    "BaggedTrees"

maxAcc = 0.6851
```



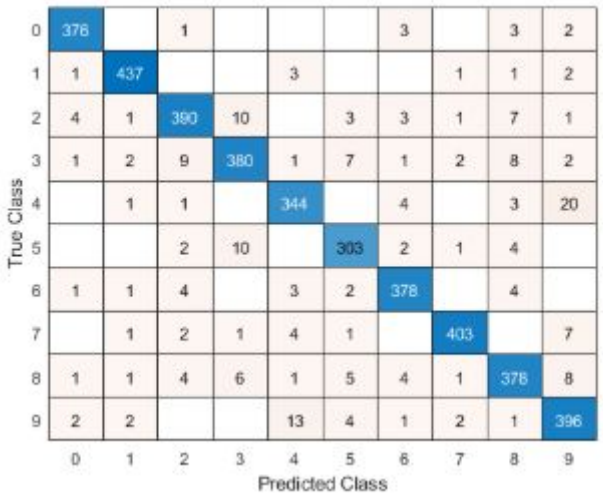


Nos ha dado un resultado muy por debajo de lo esperado. Con un 68% de acierto, hemos decidido que para las pruebas combinadas, descartamos estos predictores.
Se puede observar en la matriz de confusión que el número 1 lo detecta correctamente la mayoría de veces. Eso es debido a la forma tan diferenciada de este número respecto a los otros.

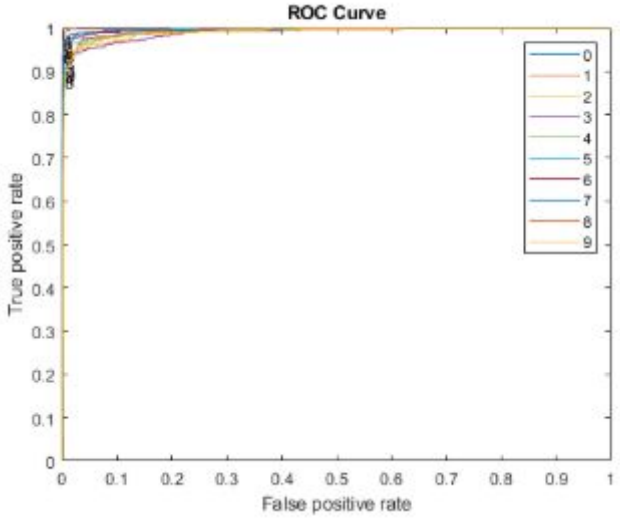
Experimento Combinado (Wavelet y K-means)

Para este experimento hemos decidido hacer pruebas con los predictores al completo y con los predictores después de aplicar la técnica PCA

Without PCA



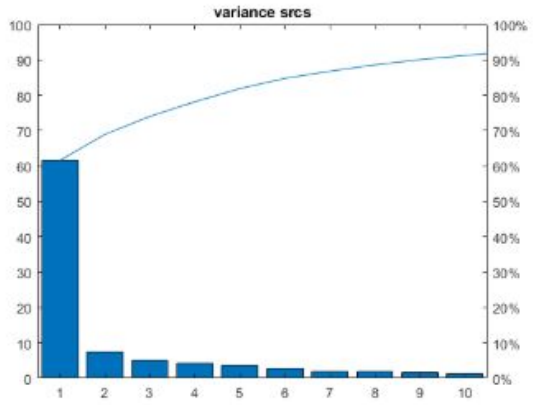

```
accuracy = 0.9416
"CubicSVM"
```

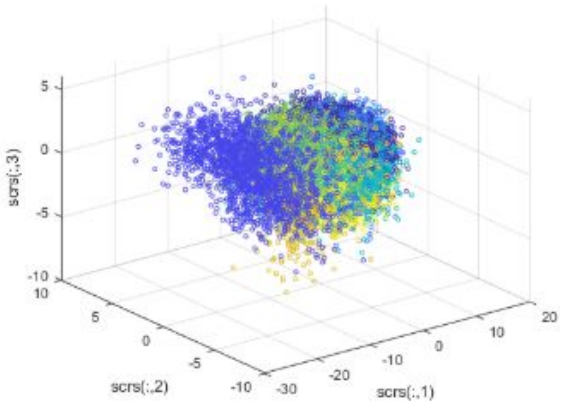


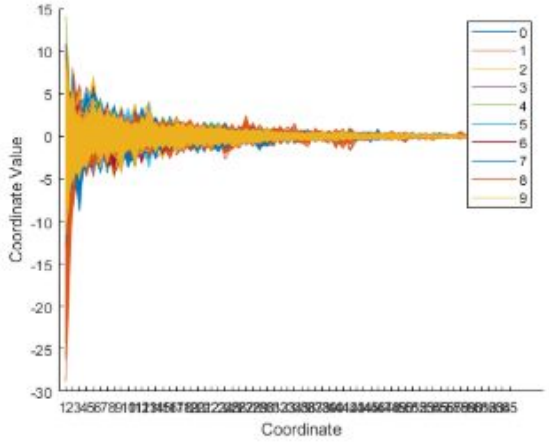
En los resultados volvemos a observar el mejor resultado por parte de los clasificadores SVM, llegando a alcanzar una precisión cercana al 95%. Eso significa que los descriptores de Wavelet son buenos

With PCA

Estudio de los datos



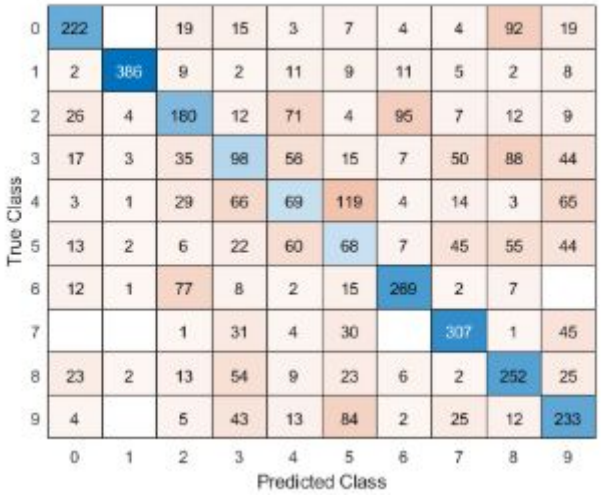




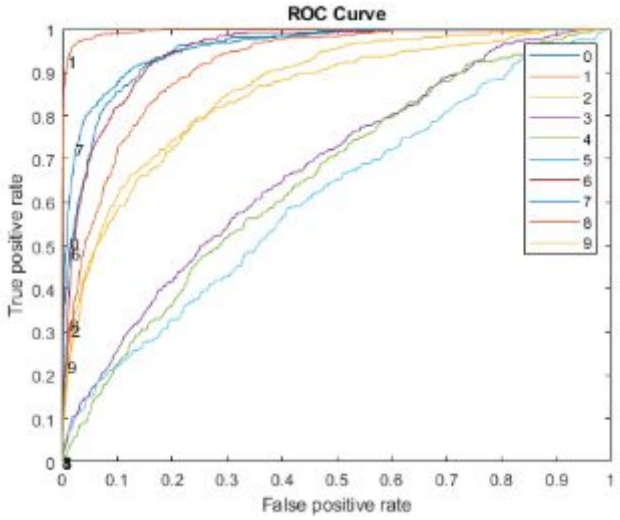

```
accuracy = 0.9391
"CubicSVM"
```

Al ser una combinación de los descriptores de Wavelet Scattering y K-means, el estudio realizado sobre los datos resulta en una combinación de ambos. Sabiendo esto, es posible que los resultados después de realizar la técnica PCA sean muy similares a los resultados de los experimentos anteriores.

El problema de la técnica PCA es que intenta buscar correlaciones que en esta caso son inexistentes, pudiendo causar resultados no óptimos




```
accuracy = 0.9391
"CubicSVM"
```



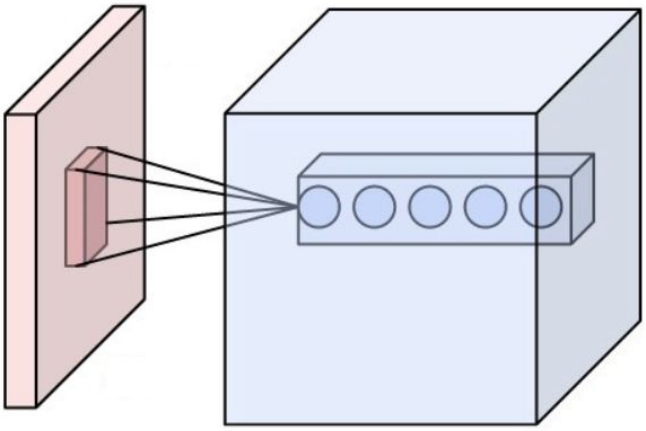
Podemos observar que los resultados al aplicar PCA no son muy buenos. Esto es debido a la importancia de cada descriptor, principalmente los descriptores de K-means, los cuales no pueden omitirse. Tambien, como explicamos anteriormente, puede ser debido a la poca correlación entre descriptores

Neural Networks

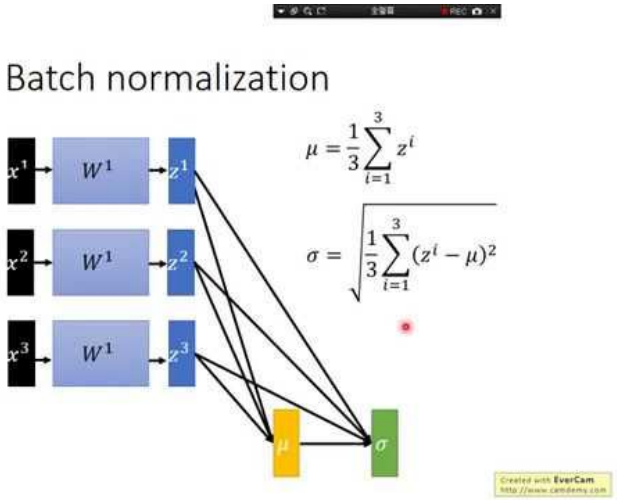
Convolutional Neural Network

Las Convolutional Neural Network son un tipo de redes neuronales cuyas características hacen que sea muy utilizadas en tareas de visión por computador y clasificación y segmentación de imágenes. En este tipo de red se le introduce una imagen y a partir de un conjunto de layers permite hacer classificacion multi classes. Los layers que caracterizan nuestra red neuronal son los siguientes:

- **Image Input Layer**: es un tipo de layer que se encarga simplemente de hacer una lectura de la imagen y normalizar los valores.
- **Convolutional Layer**: layer principal del CNN. Una convolutional layer consiste en un conjunto de neuronas que conecta cada subregión de la imagen o de los datos que imprime el layer anterior. Por cada región se le suele aplicar el producto de un peso más la suma de una base. Tras esas operaciones se le aplica una función sigmoide(Neuron Function).



- **Batch Normalization Layer**: las batch normalization layer normaliza cada canal de entrada a través de un mini-batch. Se suele colocar este tipo de layer entre el Convolutional layer y el ReLu layer para poder reducir la sensibilidad de la red inicial y aumentar la velocidad. El layer primero normaliza los datos y después calcula la desviación estándar de los valores.

- 

- **ReLU Layer:** ReLu Layer es un conjunto de neuronas cuya función de activación es definida de la siguiente manera**:**

  $$f(x) = x^+ = \max(0, x),$$

  Aun así se puede utilizar otro tipo de funciones.
- **Pool Layer:** es un tipo de layer no lineal que se encarga de comprimir los datos o la imagen. Pasa un kernel por toda la imagen y se le aplica una función. Las funciones que se suelen utilizar más son el máximo o la media de los valores del kernel.
- **Fully Connected Layer:** el fully connected layer multiplica el input por una matriz de pesos y le añade al resultado un vector base.
  La característica principal es que cada neurona del layer se conecta directamente con cada neurona del siguiente layer.
- **Output layers.**
  - **Softmax layer:** un softmax layer aplica la función softmax a la entrada. Para cada dato de entrada del layer se le aplica una función logística tal que así:

    $$p_j = \frac{e^{q_j}}{\sum_{k=1}^{N} e^{q_k}}$$

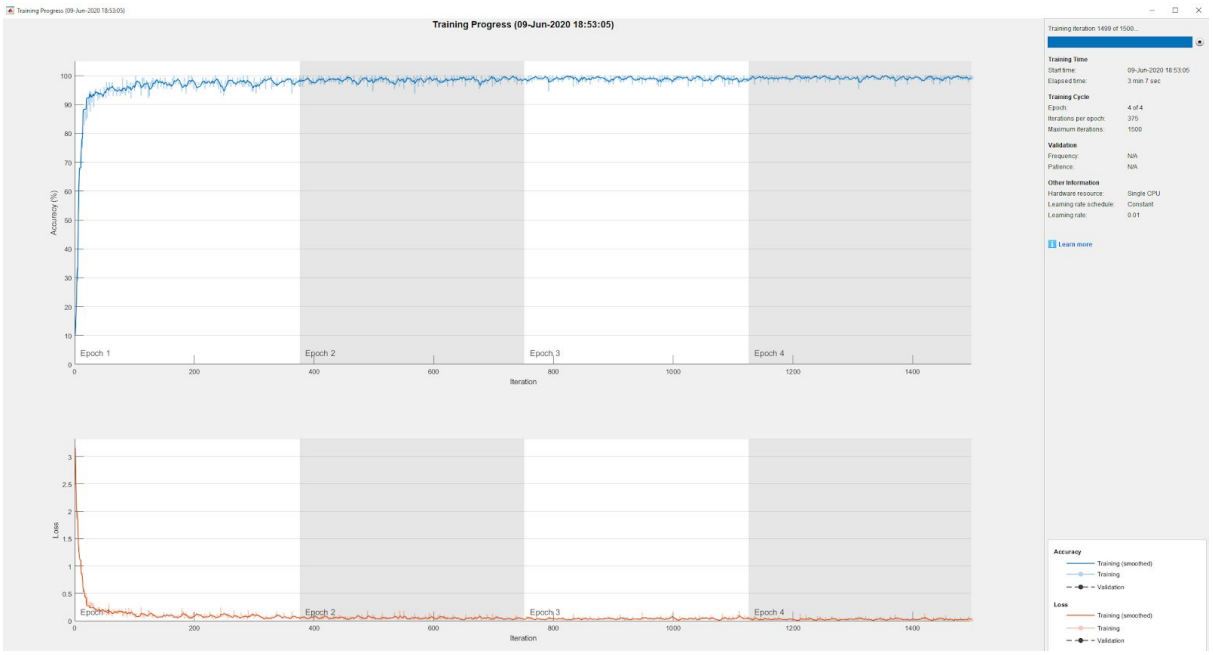
  - **Classification Layer:** Es el encargado de realizar la clasificación según los valores de que se les ha introducido la softmax layer.

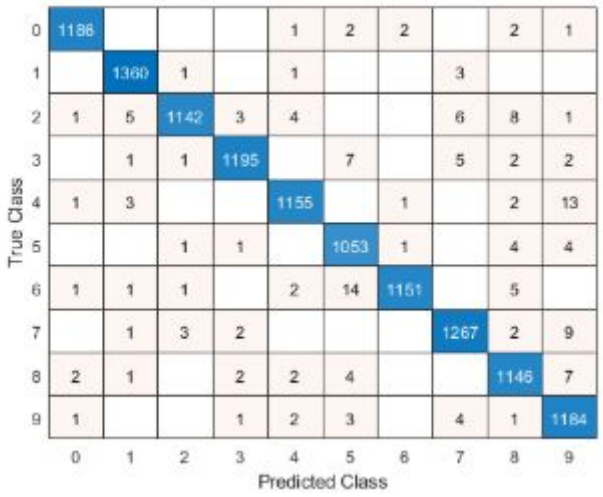
Para la construcción de nuestra red neuronal comenzábamos con un imageinput. Después vendría una tripleta de 4 layers (Convolutional Layer, Batch Normalization Layer, reluLayer y Max Pool Layer). Finalmente acabamos la red con 2 layers (Convolutional Layers y Batch Normalization Layer), 1 Fully Connected Layer y con los output layers. Todas las convolutional layers tienen 32 filtros, excepto el segundo convolutional layer que tiene 16.

Entrenamiento Red Neuronal

Para el entrenamiento de la red neuronal configuramos diferentes parámetros. El algoritmo que utilizamos para optimizar la función objetivo de la red neuronal y así configurar los parámetros fue el Stochastic gradient descent (SGD). Para la función objetivo consideramos que el Initial Learn Rate fuese de 0.01. Respecto a las iteración consideramos realizar 4 epochs como máximo y con una frecuencia de validación de los datos de la red neuronal de 30 iteraciones, donde cada epochs se modifica los datos de configuración del entrenamiento.

Resultados Red Neuronal






```
accuracy = 0.9866
```

Como podemos ver la arquitectura de la red neuronal es bastante correcta. Tan solo con unas simples iteraciones de entrenamiento de la red neuronal podemos observar como el valor Accuracy aumenta logarítmicamente y el valor de Los disminuye de manera inversa, lo que muestra un buen resultado.

La Matriz de confusión junto al valor de precisión de realizar el test de la red junto a las imágenes de testeo, nos corrobora que la red neuronal funciona correctamente.

Funciones Utilizadas

Clasification Learner

```
function [maxModel] = classification_learner(features,
labels,k_fold)
```

Entrena los modelos de cada clasificador y nos devuelve el de mayor acierto.

Distribución de datos

```
function [trainimages trainlabels testimages testlabels]  =
splitdata(images,labels, k)
```

Se encarga de dividir nuestros grupo de datos en función del porcentaje “k” para utilizar el grupo de mayor porcentaje para entrenar los modelos y el resto para probar su precisión.

Estudios de modelos y datos

```
function [] = study_of_model(model, features, labels)
```

Función que a partir de un modelo y los features realiza un estudio detallado de cómo se comporta el modelo.

```
function [featuresNorm,  pcs, scrs] = study_of_data(features,
labels)
```

Método encargado de hacer un estudio de los datos. Principalmente realiza una reducción de la dimensionalidad de los datos y hace un estudio respecto a los valores.
Tanto la función study_of_model como la study_of_data, nos ha servido para poder obtener resultados experimentales respecto al problema.

Descriptores

All points

```
function [features] = extractfeaturesallpoints(images)
```

Función encargada de extraer la lista de puntos de la imagen como descriptores.

Shape Context

```
function [features] =
extractfeatureswithShapeContext_multicenters(images, centers,
radi, nbins_r, nbins_theta)
```

Función encargada de obtener los shape context a partir de varios puntos que son introducidos en la función. Los histogramas shape context tendrán un radio en concreto, un número de bins determinados para el radio y un número de bins en concreto para los angulos.

```
function [feature] = ShapeContext(im, center, radi, nbins_r,
nbins_theta)
```

```
function [contour, points] = gecontour(im)
```

Fast-Fourier

```
function [features] =
extractfeatureswithfourier(images,Ndescriptors)
```

Extrae la transformada de fourier del contorno de la imagen. Le puedes introducir el número de descriptores que quieres utilizar. Extractfeatureswithfourier acaba devolviendo una matriz donde cada línea es la transformada de fourier de cada imagen.

```
function [features] =
extractfeatureswithfourier_with_holes(images,Ndescriptors)
```

Extra la transformada de fourier del contorno de la imagen contando los agujeros que tiene la figura. Como la funcion anterior le puedes introducir el numero de descriptores que quieres utilizar. Extractfeatureswithfourier_with_holes acaba

devolviendo una matriz donde cada línea es la transformada de fourier de cada imagen.

```
function [descriptor] = descriptor_fourier(im, Ndescriptors, forats)
```

Extrae la transformada de fourier del contorno de la imagen. Le puedes introducir el número de descriptores que quieres utilizar. Tambien le puedes introducir un parámetro booleano indicando si la transformada de fourier será con agujeros o no.

HOG

```
function [features] = extractfeatureswithHOG(images, cellSize)
```

Devuelve una matriz donde cada fila corresponde con un histograma de gradientes de las imágenes que le introduces. También se le pasa un parámetro que corresponde al tamaño de celdas que divide la imagen para realizar el histograma de gradientes.

Wavelet Scattering

```
function [features] = extractfeatureswithWaveletScattering(images)
```

Devuelve una matriz donde cada fila corresponde con la transformada de wavelet de las imágenes que le introduces.

```
function features = helperScatImages(sf,x)
```

Función donde se realiza la transformada de wavelet a partir de una imagen.

Number of Corners

```
function [features] = extractfeatureswithHarrisCorner(images)
```

Dado un conjunto de imágenes devuelve una matriz con el numero de vertices de cada imagen.

Geometric Properties

```
function [features] = geometricpropieties(images)
```

Dado un conjunto de imágenes devuelve una matriz con el área de cada imagen, el área de la concavidades de la figura de cada imagen y el numero de agujeros de la figura de cada imagen.

K-Means

```
function [features] = extractfeatureswithKMeans(images)
```

Función encargada de conseguir los puntos más representativos de la imagen obtenidos con k-means y ordenandolos por el eje X.

```
function [features] =
extractfeatureswithKMeansOrdered(images,trainlabels)
```

Función encargada de conseguir los puntos más representativos de la imagen obtenidos con k-means y ordenándolos en función de unos centros calculados previamente.

Conclusiones

Tras finalizar el proyecto hemos obtenido diferentes conclusiones al respecto. Para resolver un problema de visión de computador de reconocimiento es necesario realizar una gran cantidad de experimentos.

Para cada experimento el hecho de modificar un solo parámetro en la experimentación hace que los resultados de modelo de reconocimiento varíen. Una de las conclusiones que podemos obtener es que podemos conseguir una gran cantidad de descriptores a partir de una imagen.

Según el tipo de imágenes que tengamos, funcionará mejor un tipo de descriptor u otro, aunque hay algunos que por general funcionan correctamente en la mayoría de casos, como en el caso de wavelet scattering. Tras realizar un exhaustivo estudio de los datos que tenemos es importante probar diferentes clasificadores.

Igual que con los descriptores, a partir de unos datos podemos clasificarlos de distintas formas, lo que hace que el estudio sea muy complicado. Parametrizar correctamente los experimentos que se hacen es muy importante, es por ello que hemos intentado hacer una gran variedad de experimentos.

Los resultados que obtenemos, después de los experimentos, es que hay dos modelos que han sobresalido: Wavelet Scattering + K-means y Convolutional Neural Network. Aparte de los experimentos mostrados en este documento, hemos realizado pequeñas pruebas con la intención de testear el funcionamiento de los modelos a menor escala, esto nos ha llevado a descartar HOG debido a resultados

muy poco estables e imprecisos con distintos datos de test aunque a primera vista tuviera un resultado prometedor.

Para concluir podríamos decir que la obtención de un buen resultado no provoca que se deje de probar nuevas técnicas y algoritmos de reconocimiento de dígitos. Es importante aclarar que en está práctica se ha hecho una breve introducción en el mundo del machine learning, por lo que sería interesante para proyectos futuros profundizar más sobre el tema, en específico sobre la fascinante materia de los deep-learning.

Bibliografía

The MathWorks, Inc [En línea] Create Simple Deep Learning Network for Classification - MATLAB & Simulink Example Disponible en:

<https://www.mathworks.com/help/deeplearning/ug/create-simple-deep-learning-network-for-classification.html#responsive_offcanvas >

The MathWorks, Inc [En línea] Digit Classification with Wavelet Scattering - MATLAB & Simulink Disponible en:
<https://www.mathworks.com/help/wavelet/examples/digit-classification-with-wavelet-scattering.html >

The MathWorks, Inc [En línea] Digit Classification Using HOG Features on MNIST Database - MATLAB & Simulink Disponible en:

<https://es.mathworks.com/help/supportpkg/android/ref/digit-classification-using-hog-features-on-mnist-database.html >

ABHISHEK PAUDEL [En línea] Pen Stroke Sequence Feature Extraction from MNIST Digits.[6 junio 2020] Disponible en:

<https://abpaudel.com/blog/mnist-sequence-feature-extraction/>

Codigo

Reconeximent Automàtic de Digits Manuscrits

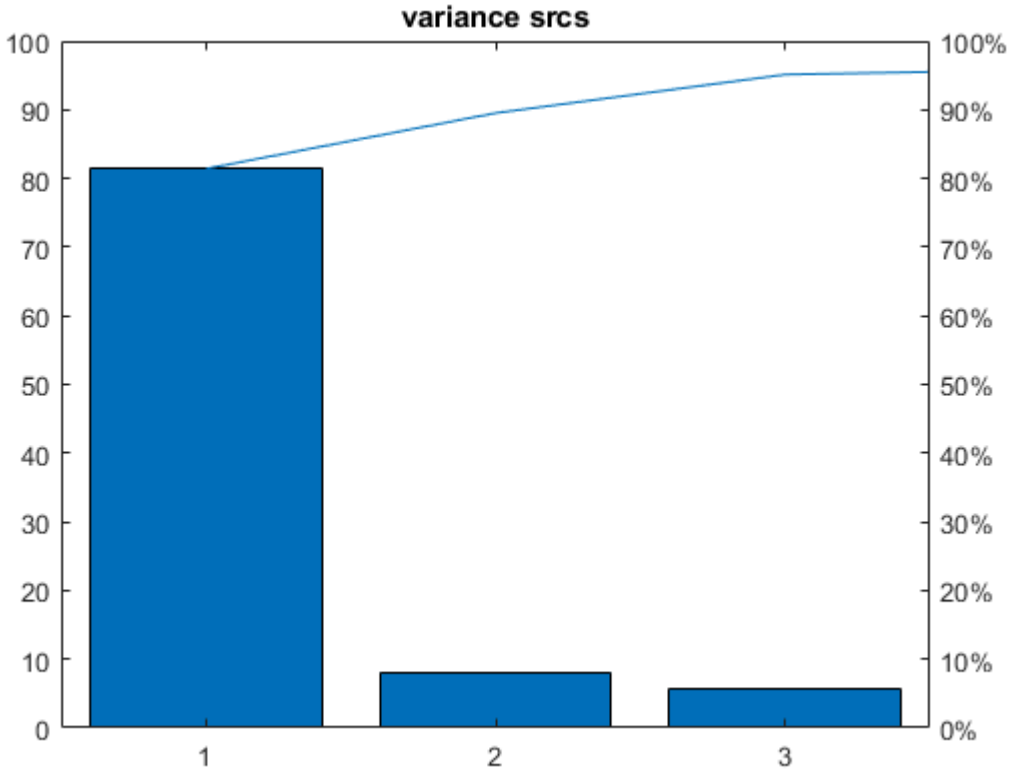
```
[images labels] = readMNIST('train-images.idx3-ubyte', 'train-labels.idx1-ubyte',20000 ,0);
[trainimages trainlabels testimages testlabels]  = splitdata(images,labels, 0.8);
```

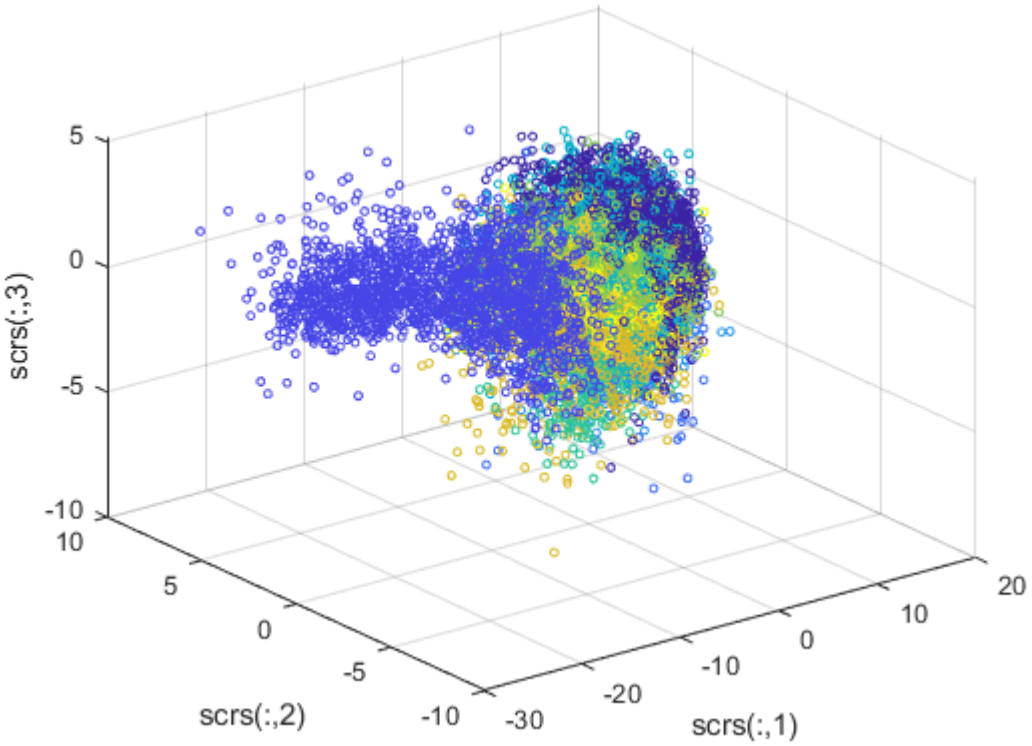
Wavelet Study

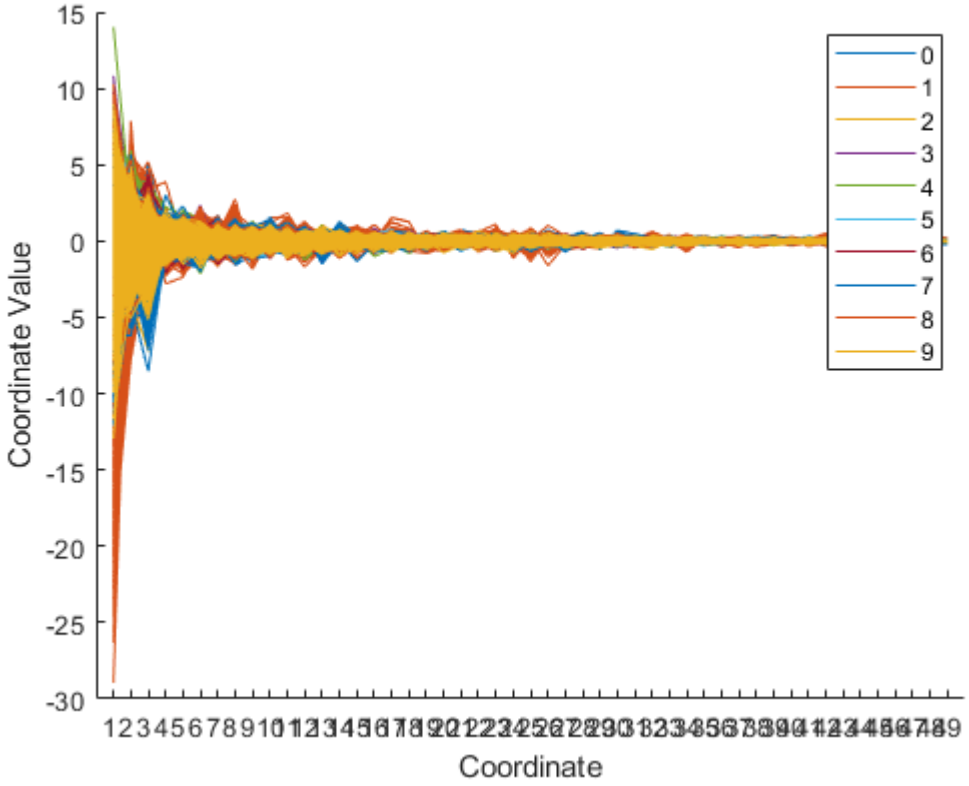
```
%[trainimages trainlabels testimages testlabels]  = splitdata(images,labels, 0.8);
[trainfeatures] = extractfeatureswithWaveletScattering(trainimages);
```

Dimensionality Reduction

```
[trainfeaturesNorm, trainpcs, trainscrs] = study_of_data(trainfeatures, trainlabels);
```




scrs(:,3)
5
0
-5
-10
10
5
0
-5
-10
scrs(:,2)
-30
-20
-10
0
10
20
scrs(:,1)


Coordinate Value
15
10
5
0
-5
-10
-15
-20
-25
-30
Coordinate
0
1
2
3
4
5
6
7
8
9

```
%for i = [2 5 10 50]
```

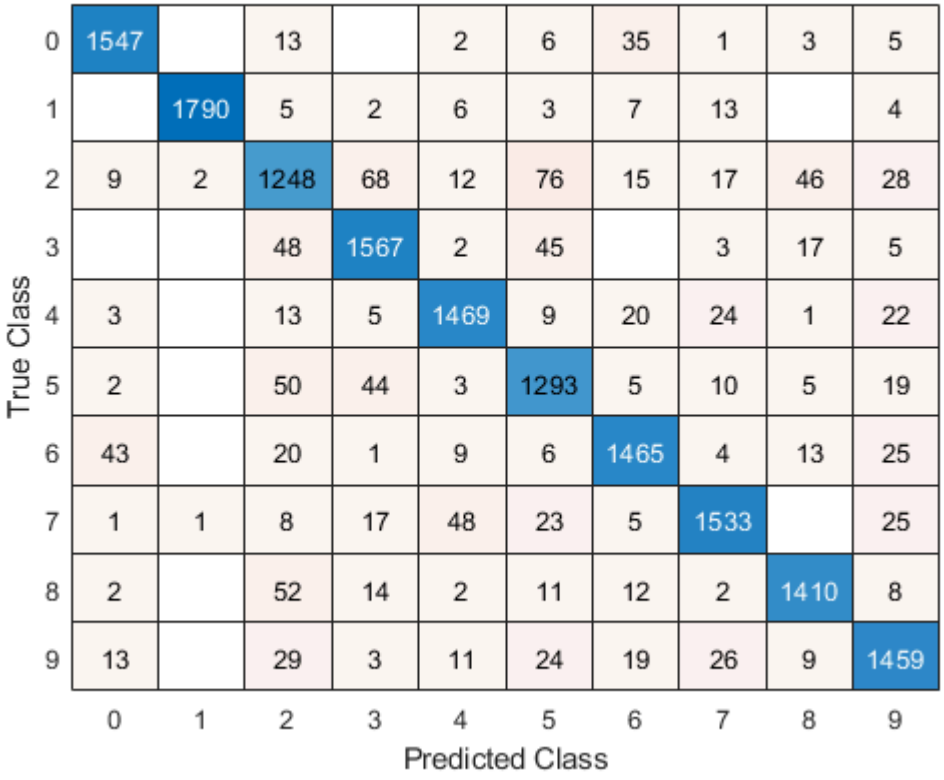
```
i=5
```

```
i = 5
```

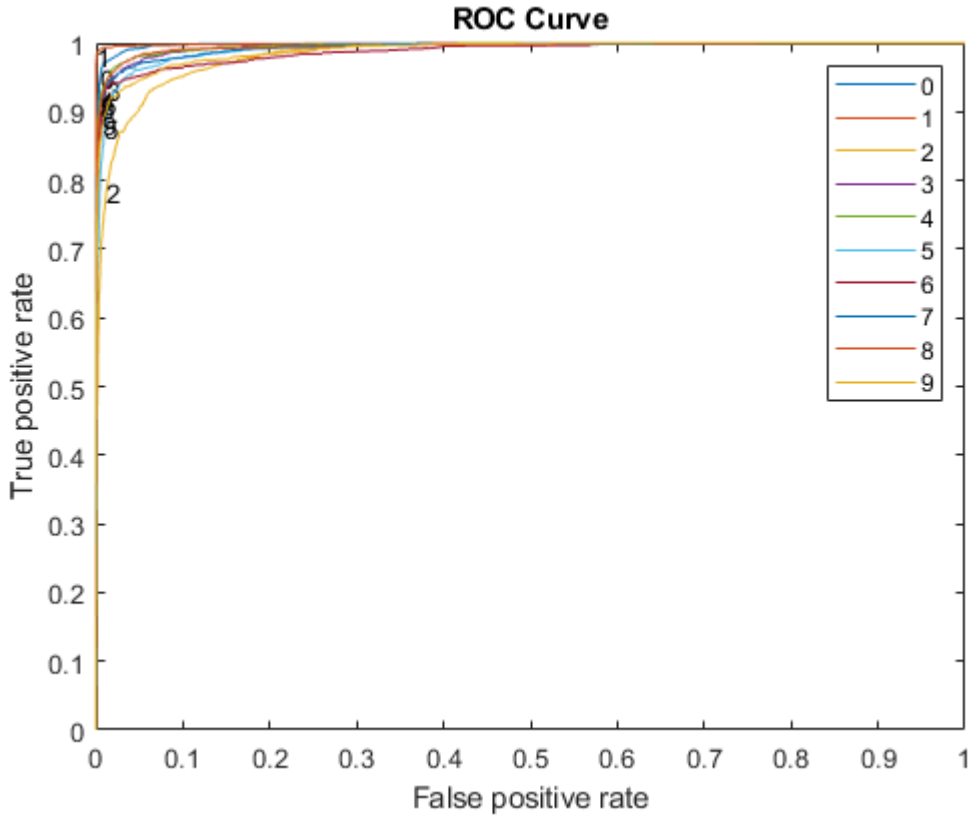
```
[maxModel] = classification_learner(trainscrs, trainlabels,i);
```

```
accuracy = 0.4807
    "FinetreeModel"
accuracy = 0.4285
    "MediumtreeModel"
accuracy = 0.3182
    "CoarsetreeModel"
accuracy = 0.7234
    "LinearDiscriminant"
accuracy = 0.7863
    "QuadraticDiscriminant"
accuracy = 0.6674
    "KernelNaiveBayes"
accuracy = 0.6822
    "GaussianNaiveBayes"
accuracy = 0.7632
    "LinearSVM"
accuracy = 0.8091
    "QuadraticSVM"
accuracy = 0.8039
    "CubicSVM"
accuracy = 0.1144
    "FineGuassianSVM"
accuracy = 0.8023
    "MediumGuassianSVM"
accuracy = 0.7479
    "CoarseGaussianSVM"
accuracy = 0.6361
    "MediumKNN"
accuracy = 0.6661
    "CosineKNN"
accuracy = 0.6356
    "CubicKNN"
accuracy = 0.6437
    "WeightedKNN"
accuracy = 0.4400
    "BoostedTrees"
accuracy = 0.6234
    "BaggedTrees"
accuracy = 0.7044
    "SubspaceDiscriminant"
accuracy = 0.6617
    "SubspaceKNN"
accuracy = 0.4385
    "RUSBoostedTrees"
maxAcc = 0.8091
```

```
study_of_model(maxModel,trainscrs,trainlabels)
```




```
accuracy = 0.0762
```




```
%end
```

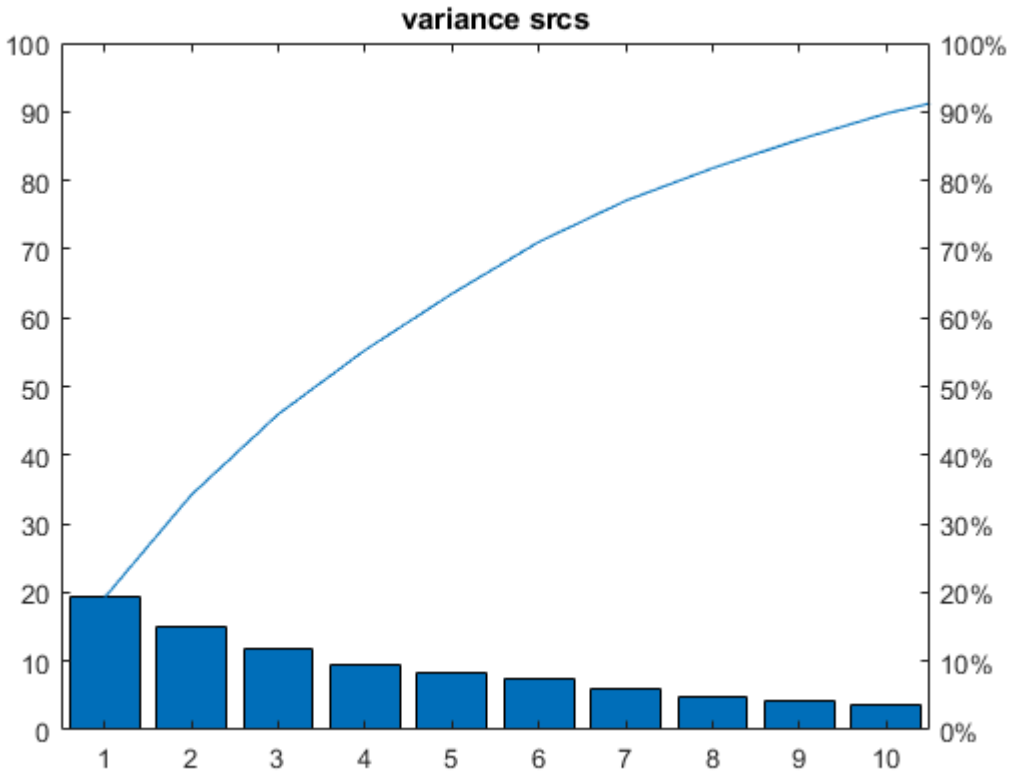
```
study_of_model(maxModel,trainscrs,trainlabels)
```

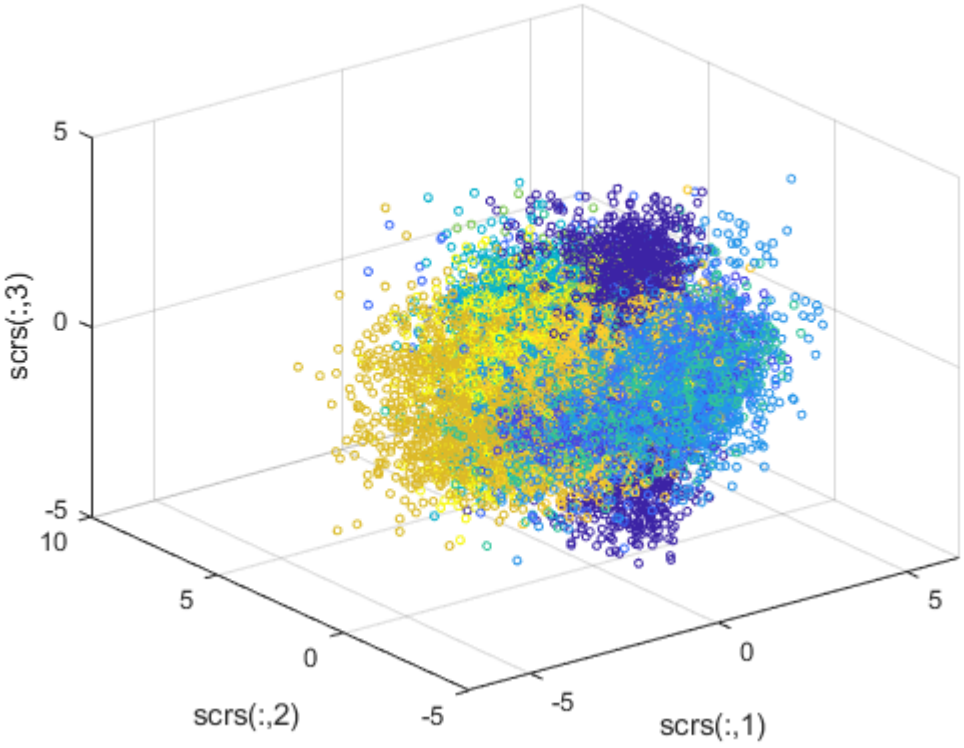
K-Means

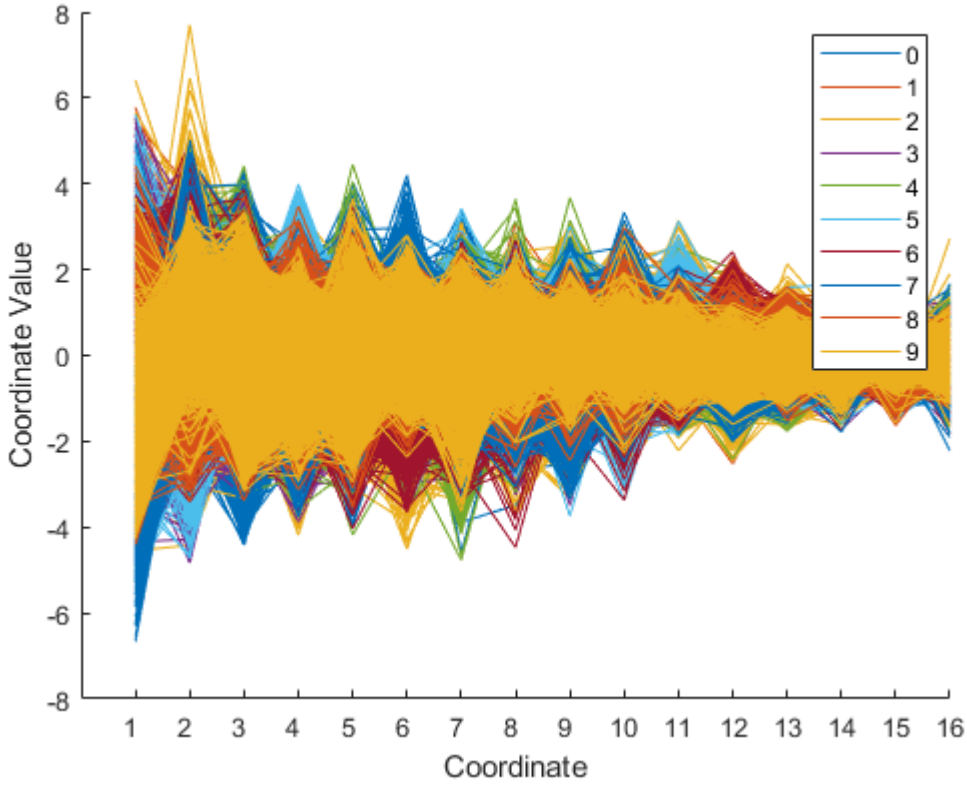
```
%[trainimages trainlabels testimages testlabels]  = splitdata(images,labels, 0.8);
[trainfeatures] = extractfeatureswithKMeans(trainimages);
```

Dimensionality Reduction

```
[trainfeaturesNorm, trainpcs, trainscrs] = study_of_data(trainfeatures, trainlabels);
```




scrs(:,3)
scrs(:,2)
scrs(:,1)


Coordinate Value
Coordinate
0
1
2
3
4
5
6
7
8
9

```
%for i = [2 5 10 50]
```

```
i=5
```

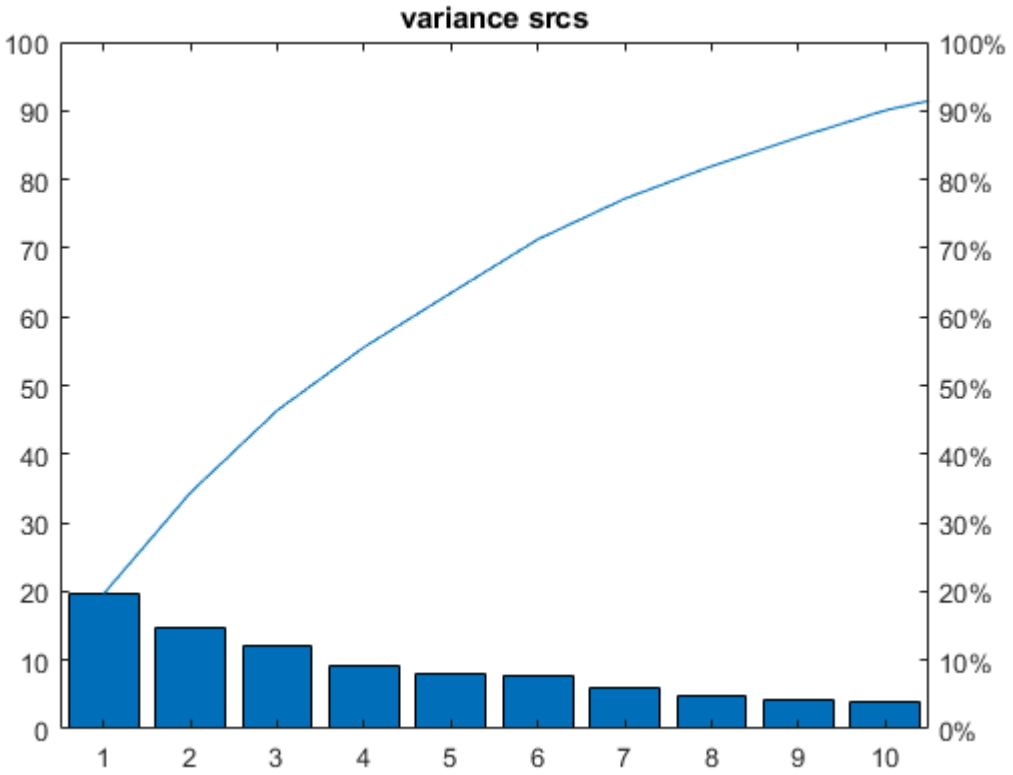
```
i = 5
```

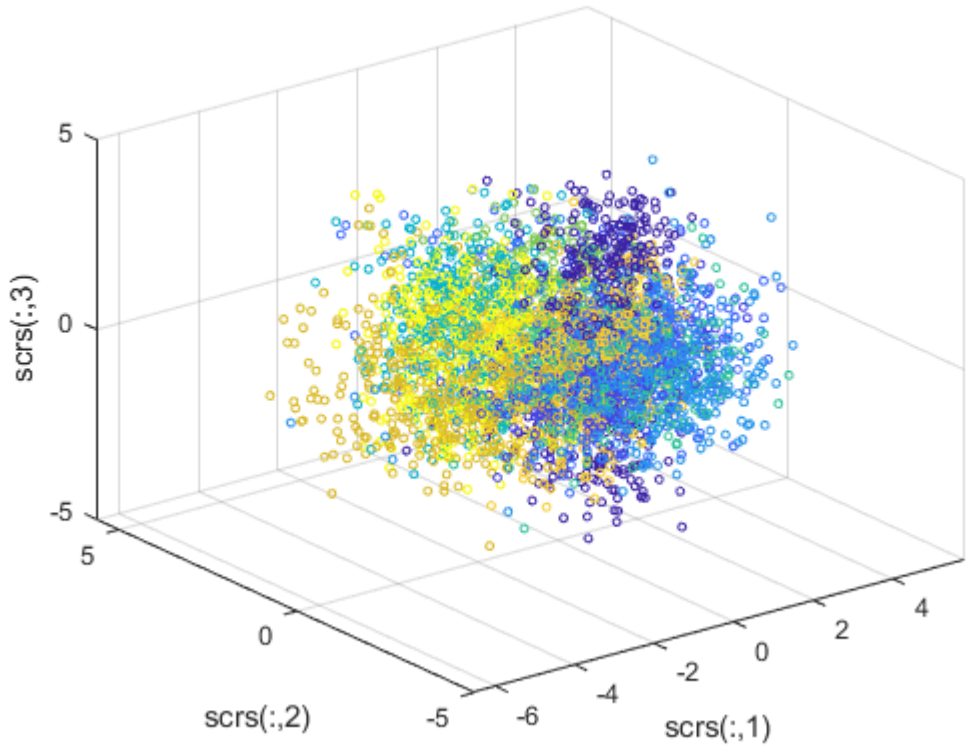
```
[maxModel] = classification_learner(trainscrs, trainlabels,i);
```

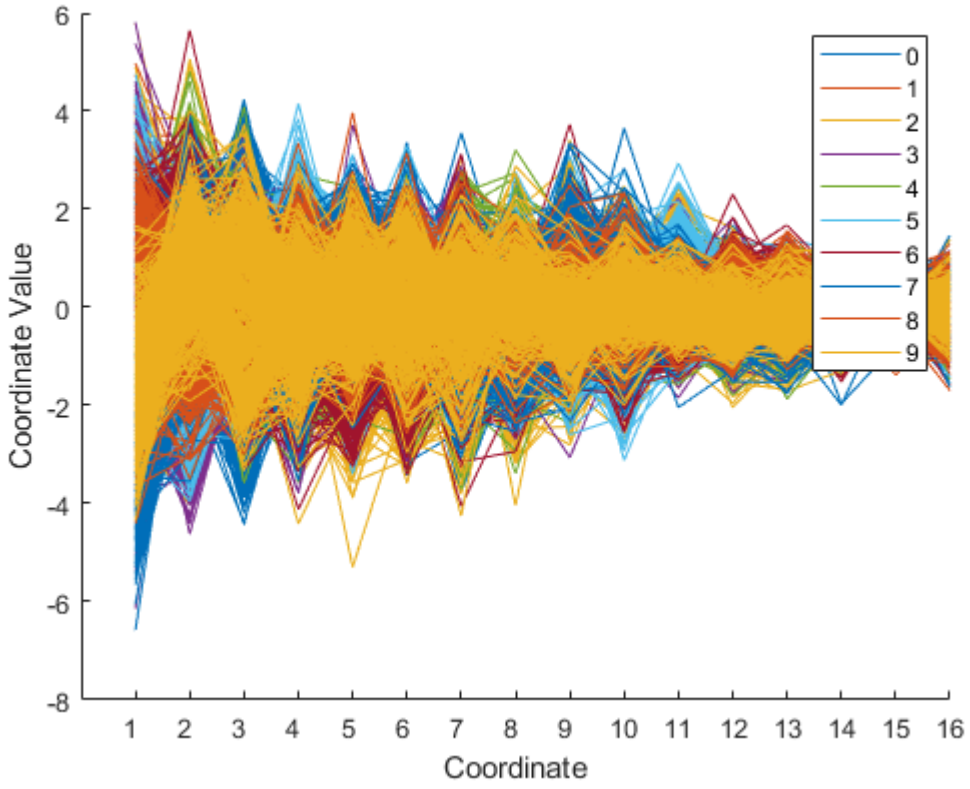
```
accuracy = 0.5372
    "FinetreeModel"
accuracy = 0.4392
    "MediumtreeModel"
accuracy = 0.3119
    "CoarsetreeModel"
accuracy = 0.6363
    "LinearDiscriminant"
accuracy = 0.8440
    "QuadraticDiscriminant"
accuracy = 0.7408
    "KernelNaiveBayes"
accuracy = 0.7373
    "GaussianNaiveBayes"
accuracy = 0.6955
    "LinearSVM"
accuracy = 0.8979
    "QuadraticSVM"
accuracy = 0.9070
    "CubicSVM"
accuracy = 0.8414
    "FineGuassianSVM"
accuracy = 0.8562
    "MediumGuassianSVM"
accuracy = 0.6904
    "CoarseGaussianSVM"
accuracy = 0.8435
    "MediumKNN"
accuracy = 0.8561
    "CosineKNN"
accuracy = 0.8451
    "CubicKNN"
accuracy = 0.8485
    "WeightedKNN"
accuracy = 0.5119
    "BoostedTrees"
accuracy = 0.8117
    "BaggedTrees"

    "Fallo"

    "Fallo"
accuracy = 0.4466
    "RUSBoostedTrees"
maxAcc = 0.9070
```

```
[testfeatures] = extractfeatureswithKMeans(testimages);
```

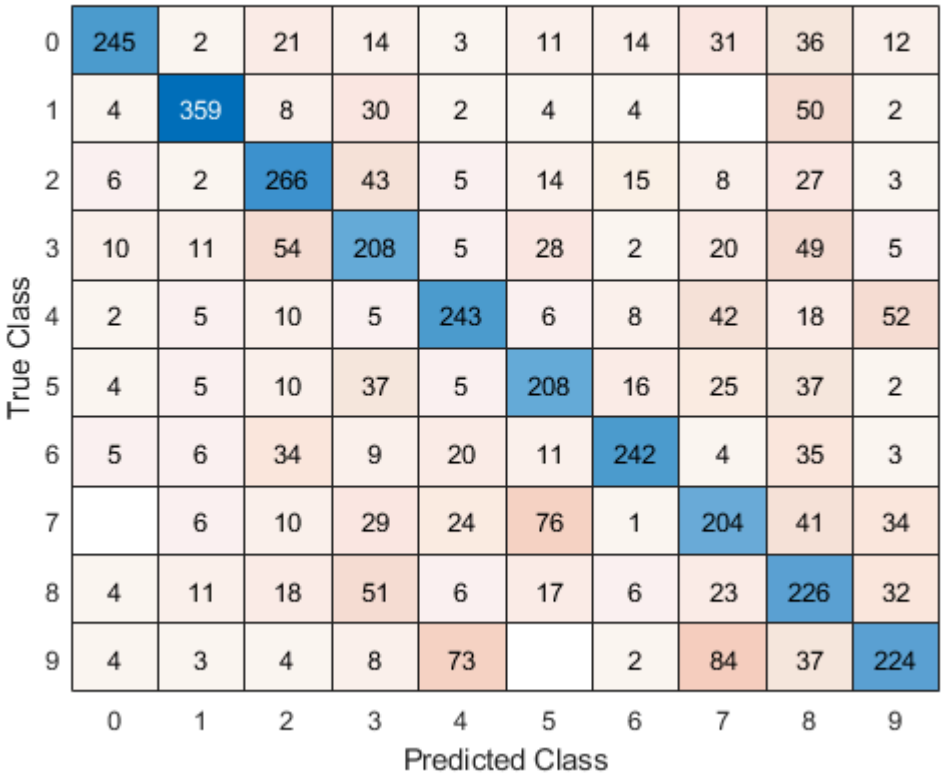
```
[testfeaturesNorm, testpcs, testscrs] = study_of_data(testfeatures, testlabels);
```


variance srcs
100
90
80
70
60
50
40
30
20
10
0
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
1
2
3
4
5
6
7
8
9
10

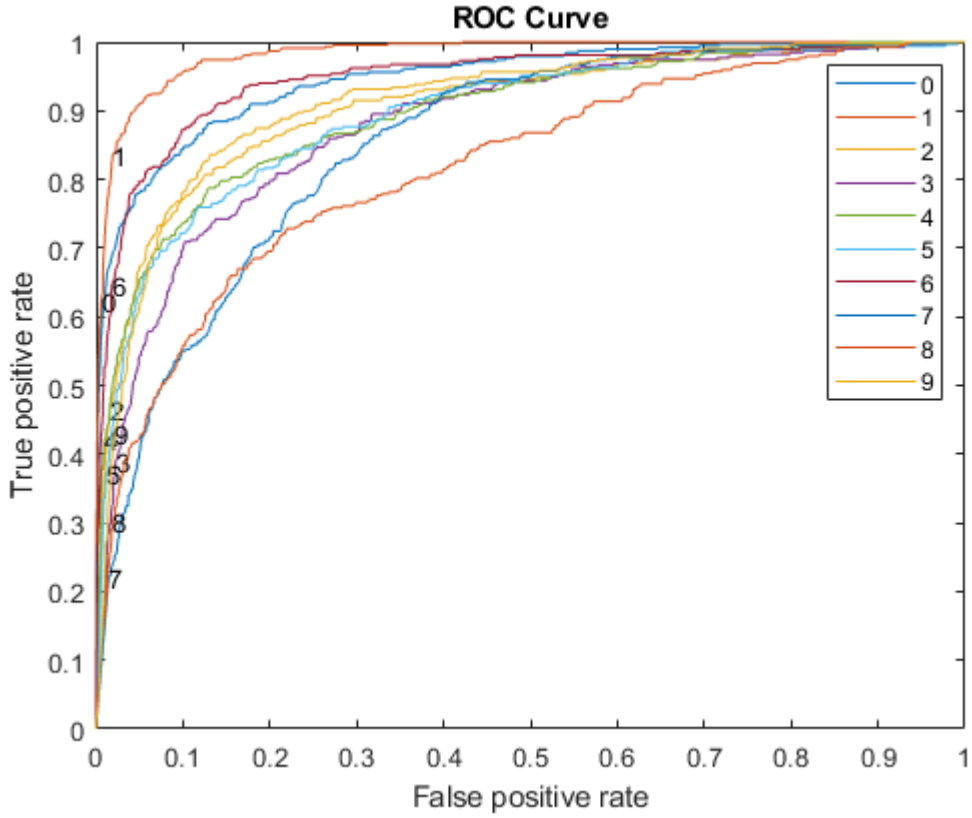

scrs(:,3)
5
0
-5
5
0
-5
scrs(:,2)
-6
-4
-2
0
2
4
scrs(:,1)




```
study_of_model(maxModel,testscrs,testlabels)
```




```
accuracy = 0.3938
```




```
%end
```

HOG

```
%[trainimages trainlabels testimages testlabels]  = splitdata(images,labels, 0.8);
[trainfeatures] = extractfeatureswithHOG(trainimages, [4 4]);
```

Dimensionality Reduction

```
%[trainfeaturesNorm, trainpcs, trainscrs] = study_of_data(trainfeatures, trainlabels);
%trainfeaturesAux = [trainfeatures trainlabels];
```

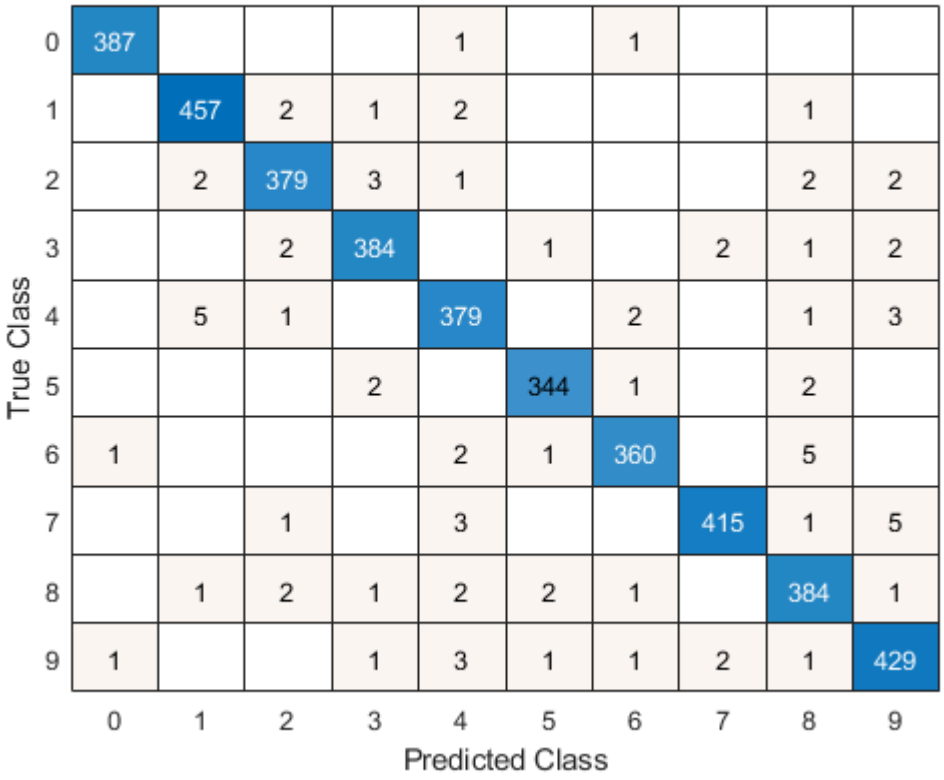
```
%for i = [2 5 10 50]
    i=5
```

```
i = 5
```

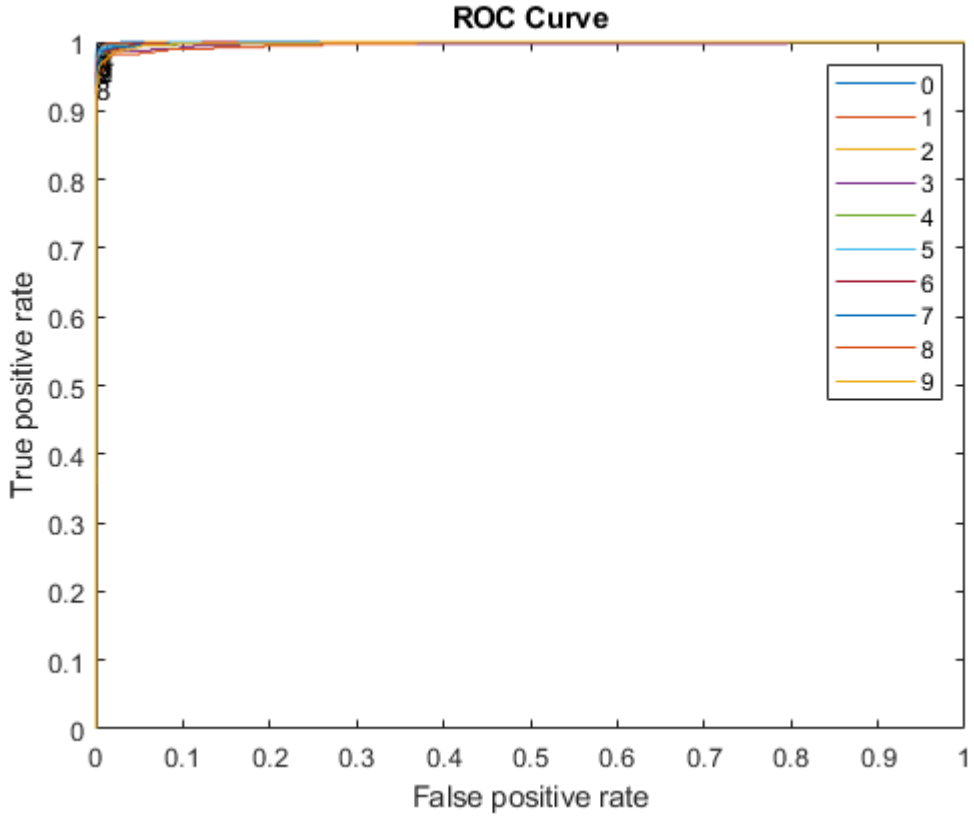
```
    %[maxModel] = classification_learner(trainscrs, trainlabels,i);
    [maxModel] = classification_learner(trainfeatures, trainlabels,i);
```

```
accuracy = 0.9814
    "QuadraticSVM"
maxAcc = 0.9814
```

```
[testfeatures] = extractfeatureswithHOG(testimages, [4 4]);
study_of_model(maxModel,testfeatures,testlabels)
```




```
accuracy = 0.0205
```




```
%end
```

Fourier

```
%[trainimages trainlabels testimages testlabels]  = splitdata(images,labels, 0.8);
[trainfeatures] = extractfeatureswithfourier(trainimages, 44);
```

Dimensionality Reduction

```
[trainfeaturesNorm, trainpcs, trainscrs] = study_of_data(trainfeatures, trainlabels);
```

```
%for i = [2 5 10 50]
    i=5
```

```
i = 5
```

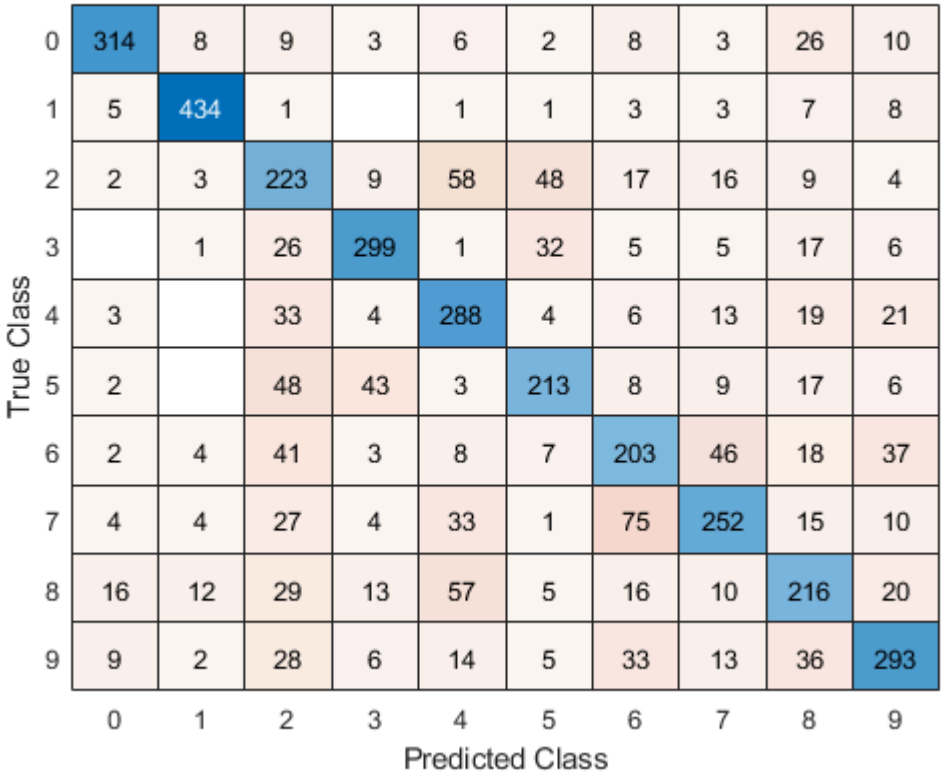
```
    %[maxModel] = classification_learner(trainscrs, trainlabels,i);
    [maxModel] = classification_learner(trainfeatures, trainlabels,i);
```

```
accuracy = 0.6561
    "FinetreeModel"
accuracy = 0.6044
    "MediumtreeModel"
accuracy = 0.4128
    "CoarsetreeModel"

    "Fallo"
```

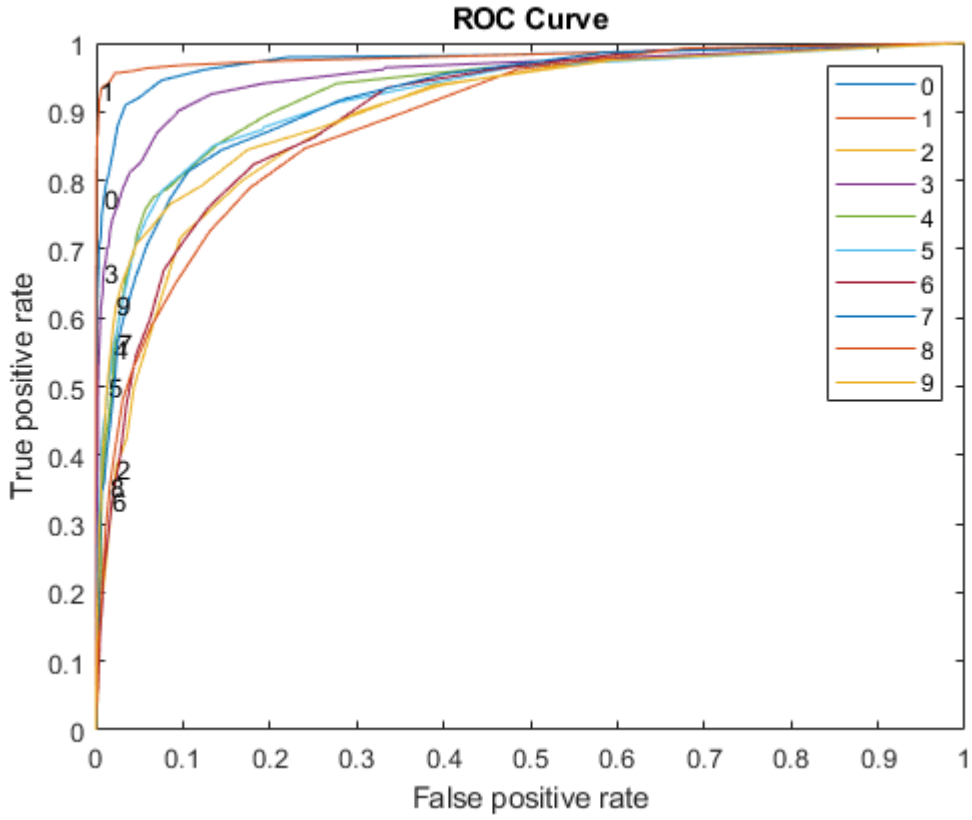
```
    "Fallo"
accuracy = 0.5230
    "KernelNaiveBayes"
accuracy = 0.4302
    "GaussianNaiveBayes"
accuracy = 0.1003
    "LinearSVM"
accuracy = 0.1003
    "QuadraticSVM"
accuracy = 0.1003
    "CubicSVM"
accuracy = 0.1003
    "FineGuassianSVM"
accuracy = 0.1003
    "MediumGuassianSVM"
accuracy = 0.1003
    "CoarseGaussianSVM"
accuracy = NaN
    "MediumKNN"
accuracy = NaN
    "CosineKNN"
accuracy = NaN
    "CubicKNN"
accuracy = NaN
    "WeightedKNN"
accuracy = 0.6304
    "BoostedTrees"
accuracy = 0.6851
    "BaggedTrees"
accuracy = 0.4998
    "SubspaceDiscriminant"
accuracy = 0.5515
    "SubspaceKNN"
accuracy = 0.6141
    "RUSBoostedTrees"
maxAcc = 0.6851
```

```
%end
```

```
[testfeatures] = extractfeatureswithfourier(testimages, 44);
study_of_model(maxModel,testfeatures,testlabels)
```




```
accuracy = 0.3163
```



Wavelet + Kmeans

Without PCA

```
%[trainimages trainlabels testimages testlabels]  = splitdata(images,labels, 0.8);
[trainfeatures1] = extractfeatureswithWaveletScattering(trainimages);
[trainfeatures2] = extractfeatureswithKMeans(trainimages);
trainfeatures = [trainfeatures1 trainfeatures2];
```

```
i=5
```

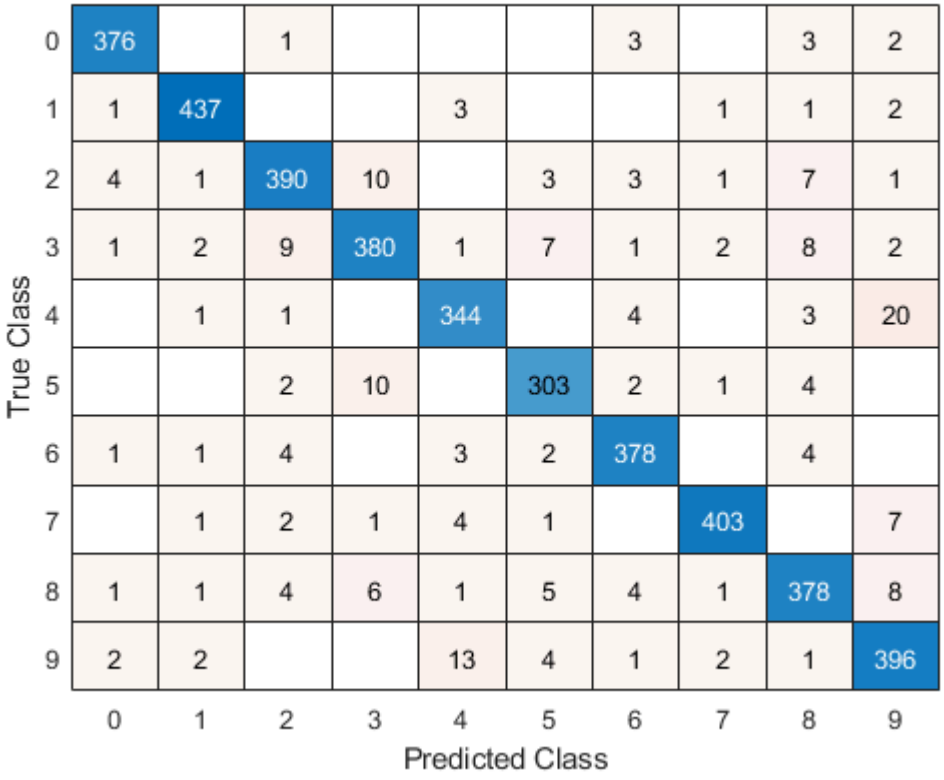
```
i = 5
```

```
[maxModel] = classification_learner(trainfeatures, trainlabels,i);
```

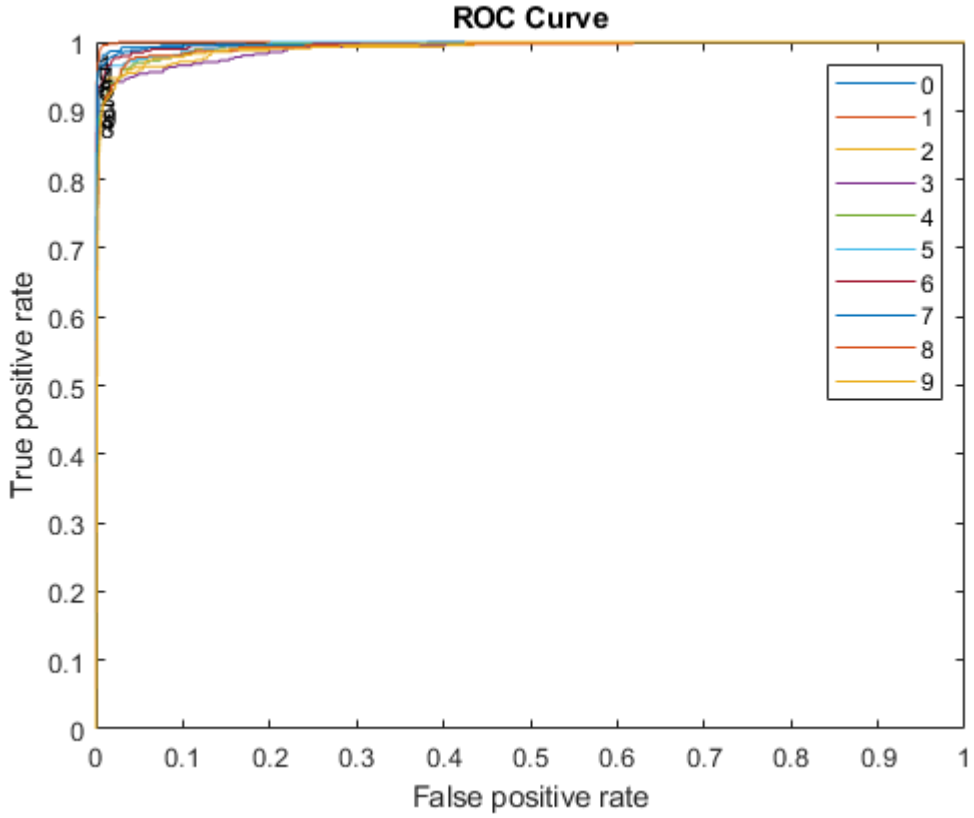
```
accuracy = 0.6541
    "FinetreeModel"
accuracy = 0.5312
    "MediumtreeModel"
accuracy = 0.3479
    "CoarsetreeModel"
accuracy = 0.8622
    "LinearDiscriminant"
accuracy = 0.9327
    "QuadraticDiscriminant"
accuracy = 0.5891
    "KernelNaiveBayes"
accuracy = 0.5599
    "GaussianNaiveBayes"
accuracy = 0.8978
    "LinearSVM"
accuracy = 0.9414
    "QuadraticSVM"
accuracy = 0.9416
    "CubicSVM"
accuracy = 0.6979
    "FineGuassianSVM"
accuracy = 0.9227
    "MediumGuassianSVM"
accuracy = 0.8732
    "CoarseGaussianSVM"
accuracy = 0.8628
    "MediumKNN"
accuracy = 0.8492
    "CosineKNN"
accuracy = 0.8621
    "CubicKNN"
accuracy = 0.8674
    "WeightedKNN"
accuracy = 0.6403
    "BoostedTrees"
accuracy = 0.8496
    "BaggedTrees"
accuracy = 0.8445
    "SubspaceDiscriminant"
accuracy = 0.8992
    "SubspaceKNN"
accuracy = 0.5883
    "RUSBoostedTrees"
```

```
maxAcc = 0.9416
```

```
[testfeatures1] = extractfeatureswithWaveletScattering(testimages);
[testfeatures2] = extractfeatureswithKMeans(testimages);
testfeatures = [testfeatures1 testfeatures2];

study_of_model(maxModel,testfeatures,testlabels)
```

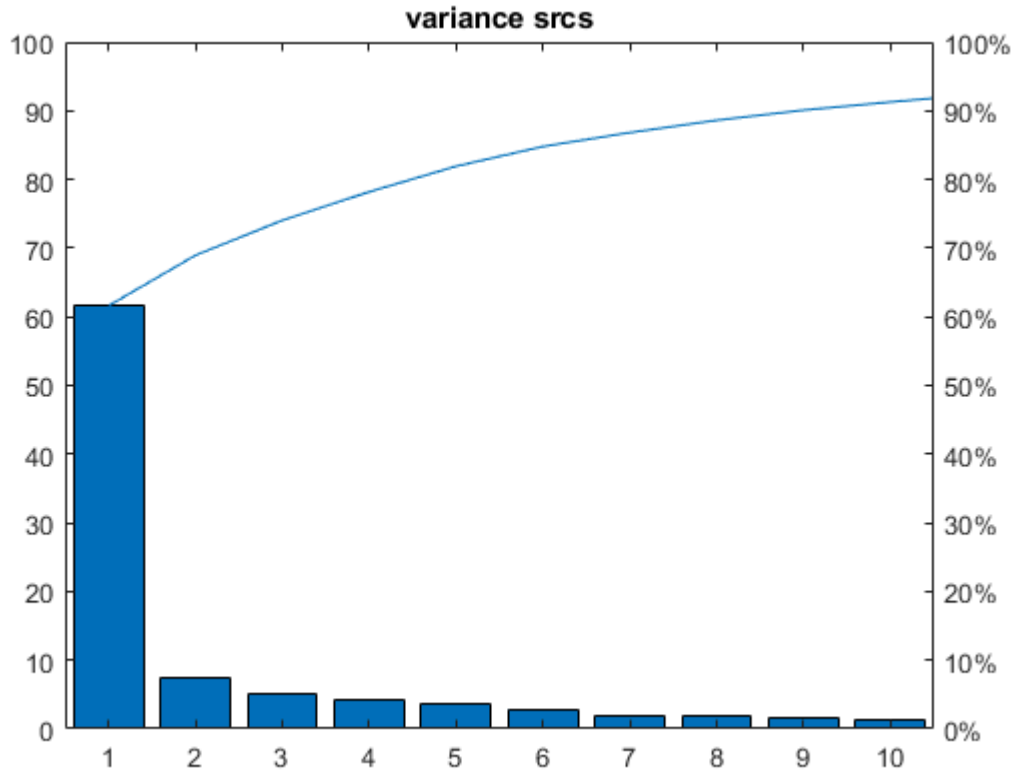


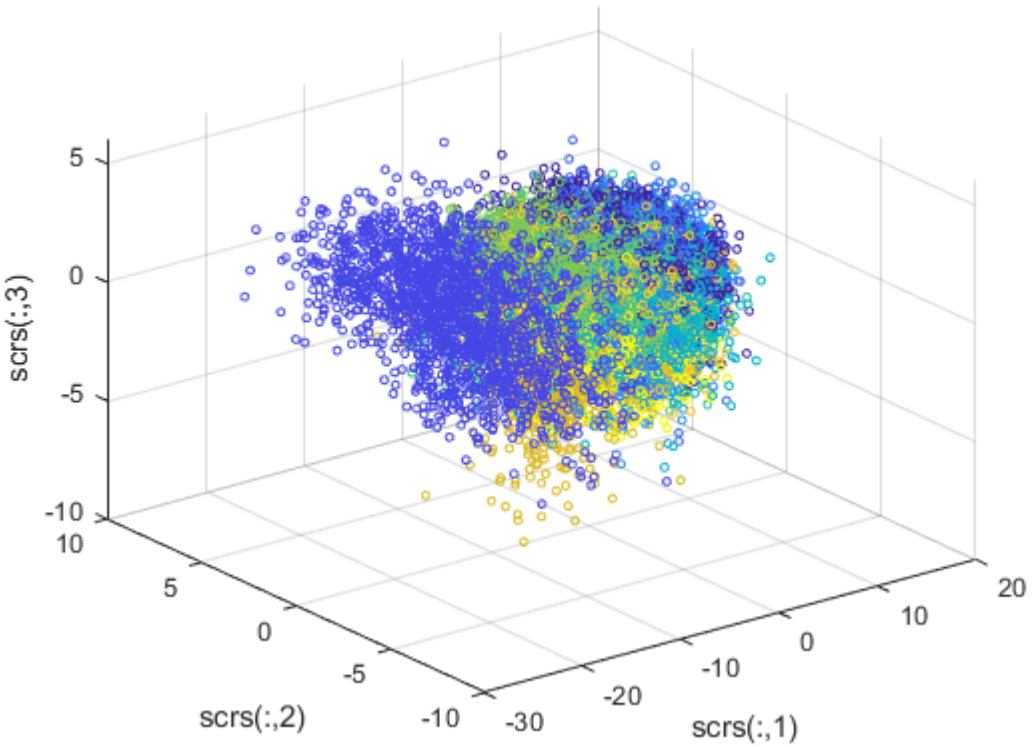

```
accuracy = 0.0538
```

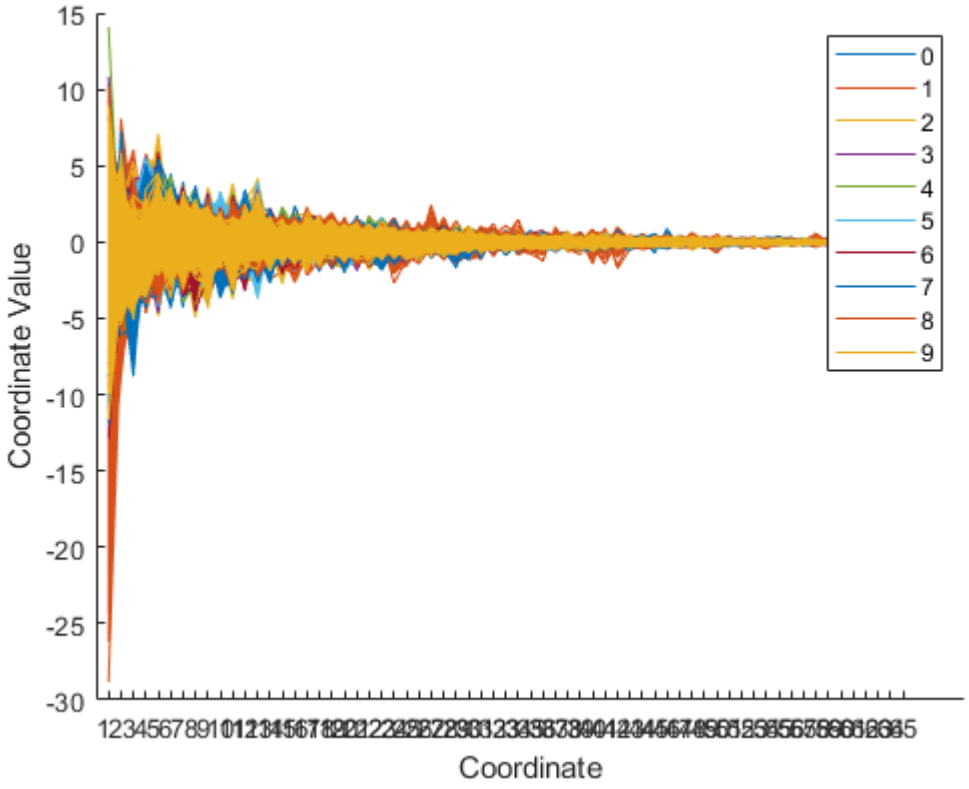


With PCA

```
[trainfeaturesNorm, trainpcs, trainscrs] = study_of_data(trainfeatures, trainlabels);
```


variance srcs
100
90
80
70
60
50
40
30
20
10
0
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
1
2
3
4
5
6
7
8
9
10


scrs(:,3)
5
0
-5
-10
10
5
0
-5
-10
scrs(:,2)
-30
-20
-10
0
10
20
scrs(:,1)




```
%for i = [2 5 10 50]
    i=5
```

```
i = 5
```

```
    [maxModel] = classification_learner(trainscrs, trainlabels,i);
```

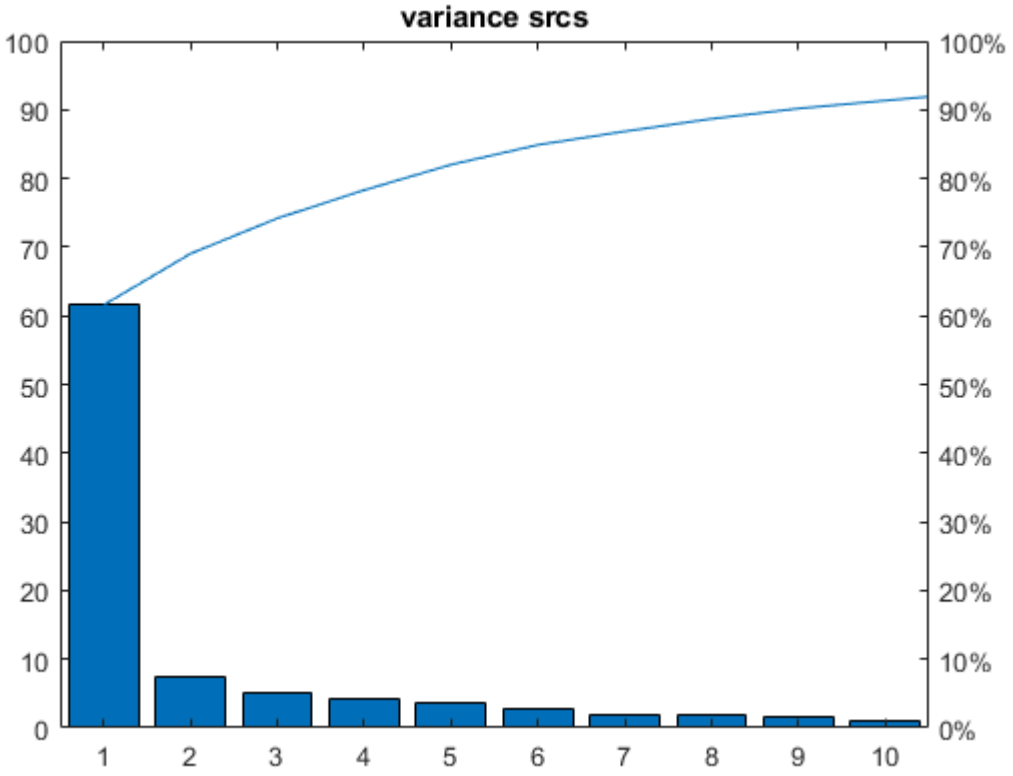
```
accuracy = 0.6549
    "FinetreeModel"
accuracy = 0.5501
    "MediumtreeModel"
accuracy = 0.3417
    "CoarsetreeModel"
accuracy = 0.8626
    "LinearDiscriminant"
accuracy = 0.9334
    "QuadraticDiscriminant"
accuracy = 0.8445
    "KernelNaiveBayes"
accuracy = 0.8515
    "GaussianNaiveBayes"
accuracy = 0.9046
    "LinearSVM"
accuracy = 0.9361
    "QuadraticSVM"
accuracy = 0.9391
    "CubicSVM"
accuracy = 0.1147
    "FineGuassianSVM"
accuracy = 0.9303
    "MediumGuassianSVM"
accuracy = 0.8990
```

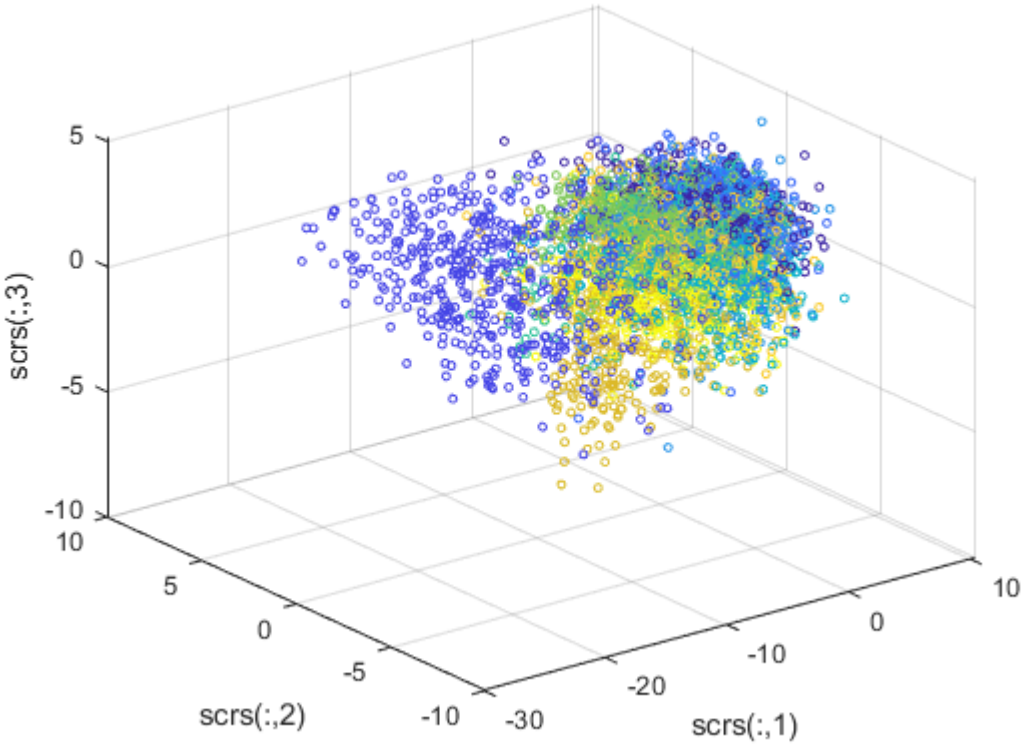
```
    "CoarseGaussianSVM"
accuracy = 0.8481
    "MediumKNN"
accuracy = 0.8847
    "CosineKNN"
accuracy = 0.8457
    "CubicKNN"
accuracy = 0.8513
    "WeightedKNN"
accuracy = 0.5911
    "BoostedTrees"
accuracy = 0.8195
    "BaggedTrees"
accuracy = 0.8437
    "SubspaceDiscriminant"
accuracy = 0.8921
    "SubspaceKNN"
accuracy = 0.5681
    "RUSBoostedTrees"
maxAcc = 0.9391
```

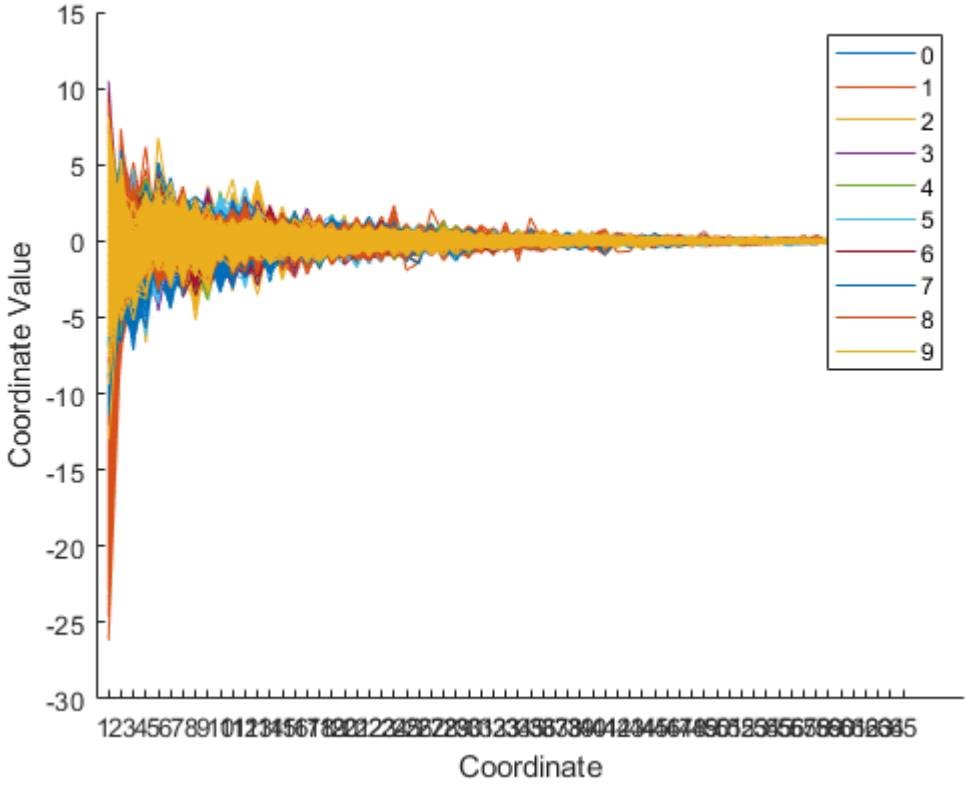
```
    %[maxModel] = classification_learner(trainfeatures, trainlabels,i);
%end
```

```
[testfeatures1] = extractfeatureswithWaveletScattering(testimages);
[testfeatures2] = extractfeatureswithKMeans(testimages);
testfeatures = [testfeatures1 testfeatures2];
```

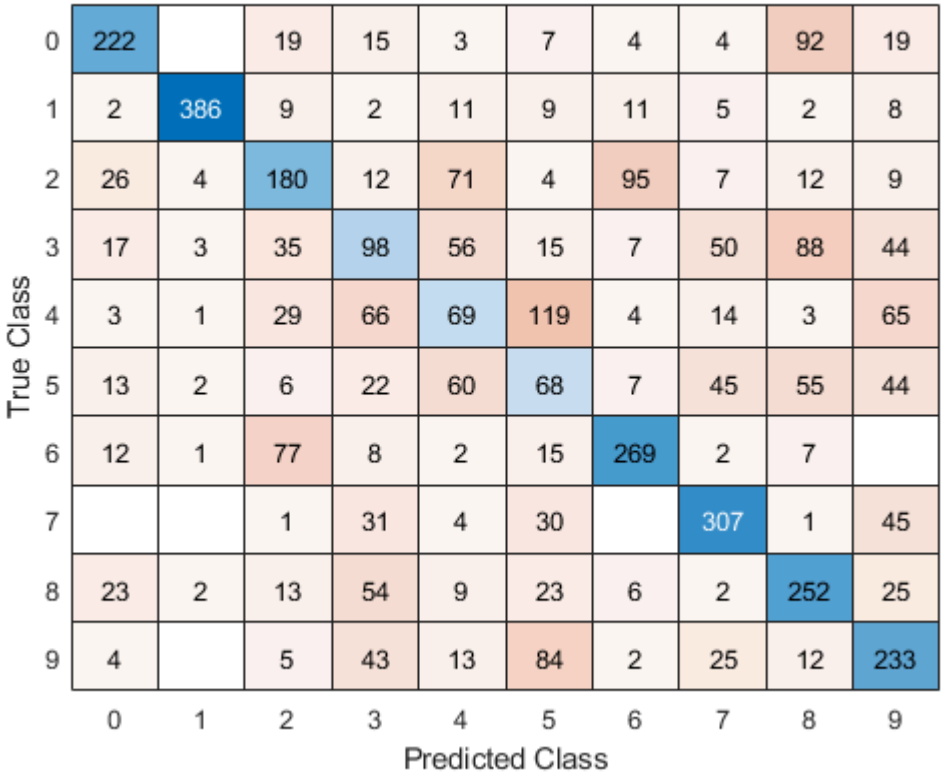
```
[testfeaturesNorm, testpcs, testscrs] = study_of_data(testfeatures, testlabels);
```



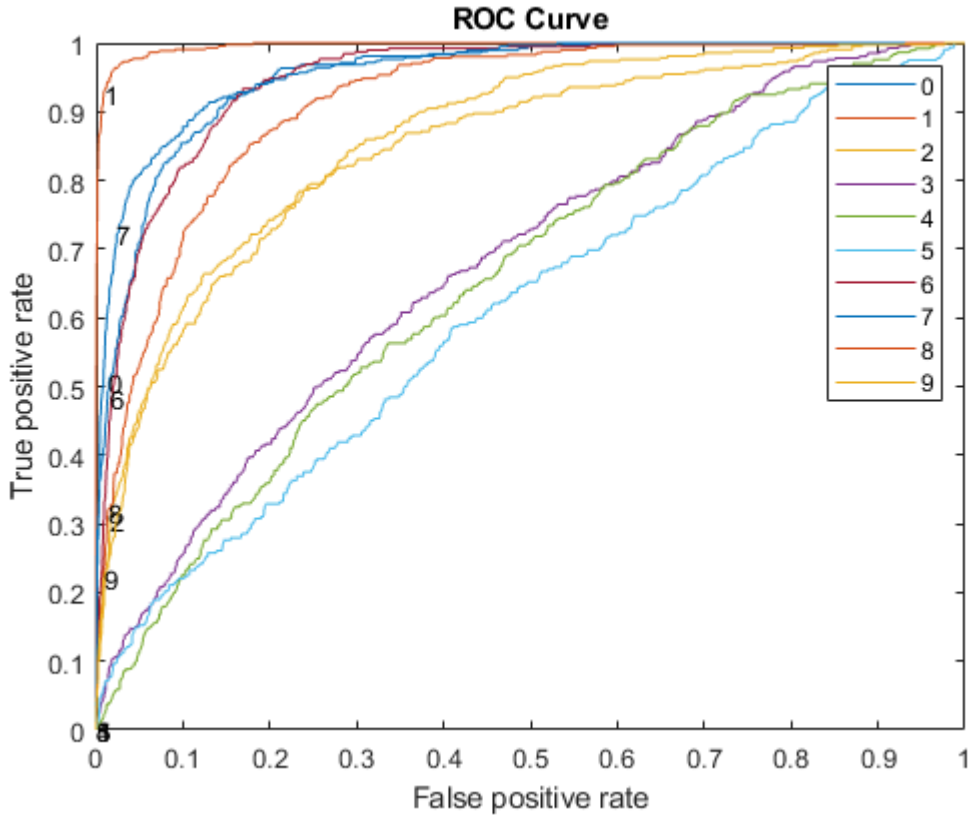





```
study_of_model(maxModel,testscrs,testlabels)
```




```
accuracy = 0.4790
```



Estudio Neural Network

Convolutional Network Practica 2

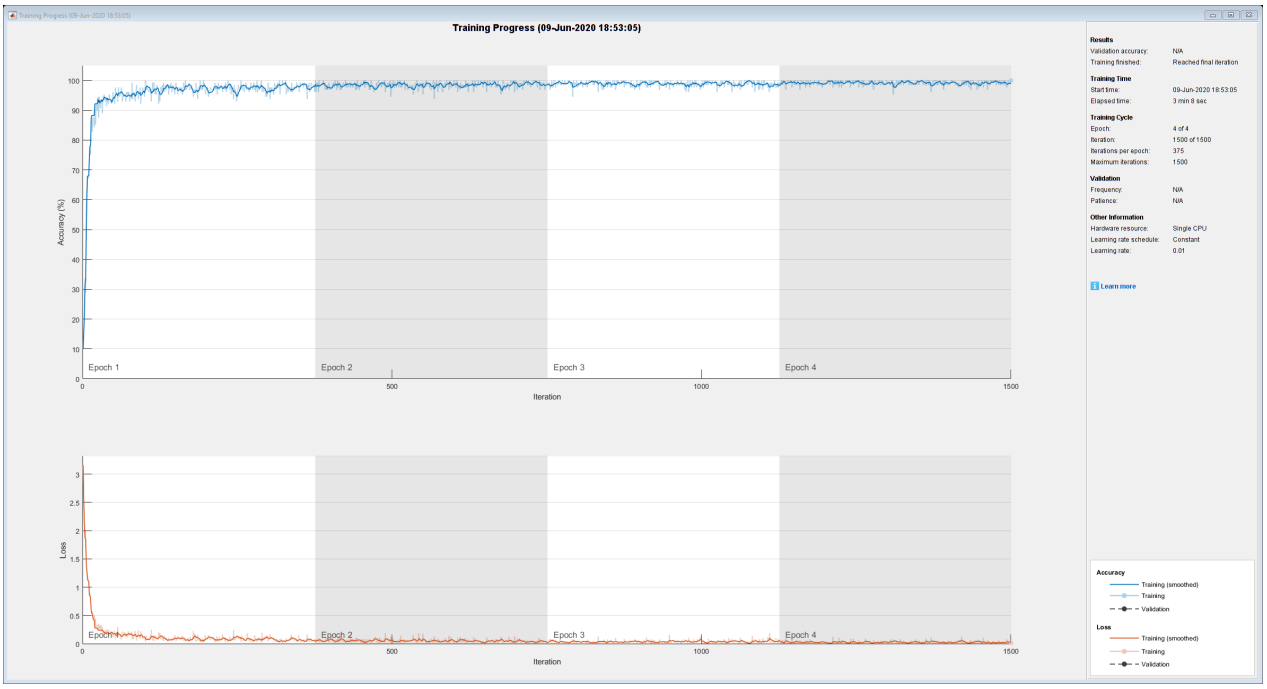
```
[images labels] = readMNIST('train-images.idx3-ubyte', 'train-labels.idx1-ubyte',60000 ,0);
[trainimages2 trainlabels2 images2 labels2]  = splitdata(images,labels, 0.8);
trainimages = zeros(20,20,1,length(trainimages2));
for i = 1:length(trainimages)
    trainimages(:,:,:,i) = trainimages2(:,:,i);
end
images = zeros(20,20,1,length(images2));
for i = 1:length(images)
    images(:,:,:,i) = images2(:,:,i);
end
trainlabels = categorical(trainlabels2);
labels = categorical(labels2);
layers = [
    imageInputLayer([20 20 1],"Name","imageinput")
    convolution2dLayer([3 8],32,"Name","conv_1","Padding","same")
    batchNormalizationLayer("Name","batchnorm_1")
    reluLayer("Name","relu_1")
    maxPooling2dLayer([2 2],"Name","maxpool_1","Padding","same","Stride",[2 2])
    convolution2dLayer([3 3],16,"Name","conv_2","Padding","same")
    batchNormalizationLayer("Name","batchnorm_2")
    reluLayer("Name","relu_2")
    maxPooling2dLayer([2 2],"Name","maxpool_2_1","Padding","same","Stride",[2 2])
    convolution2dLayer([3 3],32,"Name","conv_3_1","Padding","same")
    batchNormalizationLayer("Name","batchnorm_3_1")
    reluLayer("Name","relu_3_1")
    maxPooling2dLayer([2 2],"Name","maxpool_2_2","Padding","same","Stride",[2 2])
    convolution2dLayer([3 3],32,"Name","conv_3_2","Padding","same")
    batchNormalizationLayer("Name","batchnorm_3_2")
    reluLayer("Name","relu_3_2")
    maxPooling2dLayer([2 2],"Name","maxpool_2_3","Padding","same","Stride",[2 2])
    convolution2dLayer([3 3],32,"Name","conv_3_3","Padding","same")
    batchNormalizationLayer("Name","batchnorm_3_3")
    fullyConnectedLayer(10,"Name","fc")
    softmaxLayer("Name","softmax")
    classificationLayer("Name","classoutput")];
```

Training options

```
options = trainingOptions('sgdm', ...
    'InitialLearnRate',0.01, ...
    'MaxEpochs',4, ...
    'Shuffle','every-epoch', ...
    'ValidationFrequency',30, ...
    'Verbose',false, ...
    'Plots','training-progress');
```

Train Netowrk Using Training Data

```
net2 = trainNetwork(trainimages, trainlabels,layers,options);
```

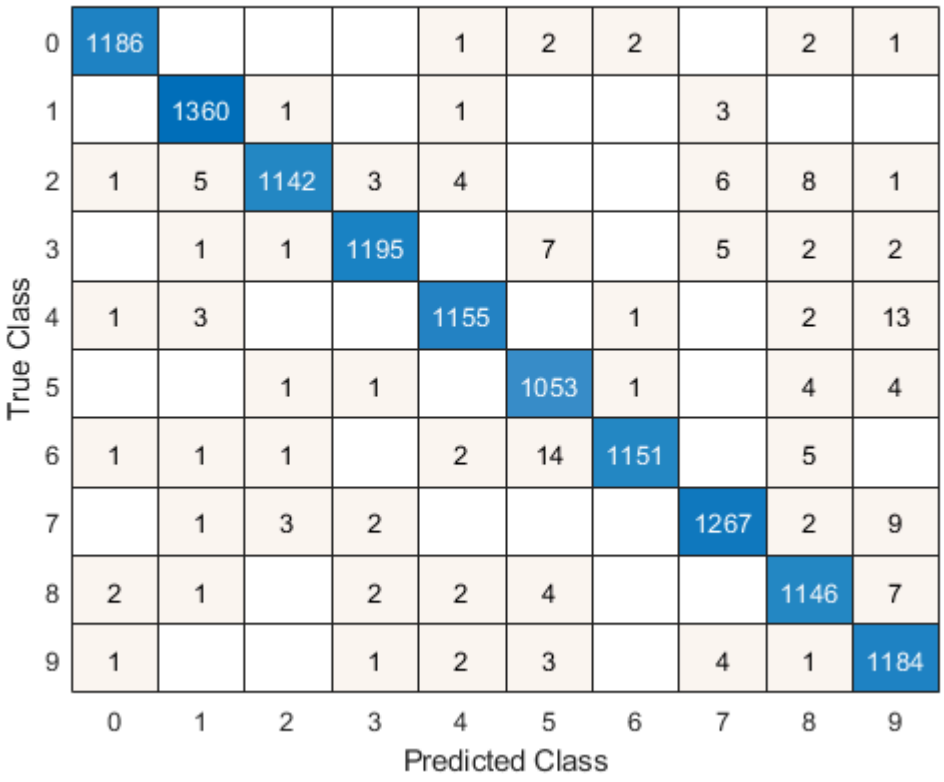


Classify Validation Images and Compute Accuracy

```
labelsPredict = classify(net2,images);
accuracy = sum(labelsPredict == labels)/length(labels)
```

```
accuracy = 0.9866
```

```
confusionchart(labels,labelsPredict)
```




```
ans =
  ConfusionMatrixChart with properties:

    NormalizedValues: [10×10 double]
         ClassLabels: [10×1 categorical]

  Show all properties
```

Multiclass classifications

```
function [maxModel] = classification_learner(features, labels,k_fold)
    maxAcc = 0;

    try
        FinetreeModel = fitctree(features,labels,"SplitCriterion","gdi","MaxNumSplits",100, 'Su
        [accuracy] = accuracy_result(FinetreeModel, features, labels,k_fold)
        display("FinetreeModel")
        if accuracy > maxAcc
            maxModel = FinetreeModel;
            maxAcc = accuracy;
        end
    catch ME
        display("Fallo")
    end
```

```
    try
        MediumtreeModel = fitctree(features,labels,"SplitCriterion","gdi","MaxNumSplits",20, 'S
        [accuracy] = accuracy_result(MediumtreeModel, features, labels,k_fold)
        display("MediumtreeModel")
        if accuracy > maxAcc
            maxModel = FinetreeModel;
            maxAcc = accuracy;
        end
    catch ME
        display("Fallo")
    end

    try
        CoarsetreeModel = fitctree(features,labels,"SplitCriterion","gdi","MaxNumSplits",4, 'Su
        [accuracy] = accuracy_result(CoarsetreeModel, features, labels,k_fold)
        display("CoarsetreeModel")
        if accuracy > maxAcc
            maxModel = FinetreeModel;
            maxAcc = accuracy;
        end
    catch ME
        display("Fallo")
    end

    try
        LinearDiscriminant =  fitcdiscr(features,labels,"DiscrimType","linear" , 'Gamma', 0, 'F
        [accuracy] = accuracy_result(LinearDiscriminant, features, labels,k_fold)
        display("LinearDiscriminant")
        if accuracy > maxAcc
            maxModel = LinearDiscriminant;
            maxAcc = accuracy;
        end
    catch ME
        display("Fallo")
    end

    try
        QuadraticDiscriminant = fitcdiscr(features,labels,"DiscrimType","quadratic", 'FillCoeff
        [accuracy] = accuracy_result(QuadraticDiscriminant, features, labels,k_fold)
        display("QuadraticDiscriminant")
        if accuracy > maxAcc
            maxModel = QuadraticDiscriminant;
            maxAcc = accuracy;
        end
    catch ME
        display("Fallo")
    end

    try
        KernelNaiveBayes = fitcnb(features,labels,"DistributionNames",repmat({'Kernel'}, 1, siz
        [accuracy] = accuracy_result(KernelNaiveBayes, features, labels,k_fold)
        display("KernelNaiveBayes")
        if accuracy > maxAcc
            maxModel = KernelNaiveBayes;
```

```
            maxAcc = accuracy;
        end
    catch ME
        display("Fallo")
    end

    try
        GaussianNaiveBayes = fitcnb(features,labels,"DistributionNames",repmat({'Normal'}, 1, s
        [accuracy] = accuracy_result(GaussianNaiveBayes, features, labels,k_fold)
        display("GaussianNaiveBayes")
        if accuracy > maxAcc
            maxModel = GaussianNaiveBayes;
            maxAcc = accuracy;
        end
    catch ME
        display("Fallo")
    end

    try
        t = templateSVM("KernelFunction","polynomial","PolynomialOrder",1,"KernelScale","auto",
        LinearSVM = fitcecoc(features,labels,"Learners",t, 'Coding', 'onevsone');
        [accuracy] = accuracy_result(LinearSVM, features, labels,k_fold)
        display("LinearSVM")
        if accuracy > maxAcc
            maxModel = LinearSVM;
            maxAcc = accuracy;
        end
    catch ME
        display("Fallo")
    end

    try
        t = templateSVM("KernelFunction","polynomial","PolynomialOrder",2,"KernelScale","auto",
        QuadraticSVM = fitcecoc(features,labels,"Learners",t, 'Coding', 'onevsone');
        [accuracy] = accuracy_result(QuadraticSVM, features, labels,k_fold)
        display("QuadraticSVM")
        if accuracy > maxAcc
            maxModel = QuadraticSVM;
            maxAcc = accuracy;
        end
    catch ME
        display("Fallo")
    end

    try
        t = templateSVM("KernelFunction","polynomial","PolynomialOrder",3,"KernelScale","auto",
        CubicSVM = fitcecoc(features,labels,"Learners",t, 'Coding', 'onevsone');
        [accuracy] = accuracy_result(CubicSVM, features, labels,k_fold)
        display("CubicSVM")
        if accuracy > maxAcc
            maxModel = CubicSVM;
            maxAcc = accuracy;
        end
    catch ME
```

```
        display("Fallo")
    end

    try
        t = templateSVM("KernelFunction","gaussian", "KernelScale",1.8,"BoxConstraint",1,"Stand
        FineGuassianSVM = fitcecoc(features,labels,"Learners",t, 'Coding', 'onevsone');
        [accuracy] = accuracy_result(FineGuassianSVM, features, labels,k_fold)
        display("FineGuassianSVM")
        if accuracy > maxAcc
            maxModel = FineGuassianSVM;
            maxAcc = accuracy;
        end
    catch ME
        display("Fallo")
    end

    try
        t = templateSVM("KernelFunction","gaussian", "KernelScale",7,"BoxConstraint",1,"Standar
        MediumGuassianSVM = fitcecoc(features,labels,"Learners",t,'Coding', 'onevsone');
        [accuracy] = accuracy_result(MediumGuassianSVM, features, labels,k_fold)
        display("MediumGuassianSVM")
        if accuracy > maxAcc
            maxModel = MediumGuassianSVM;
            maxAcc = accuracy;
        end
    catch ME
        display("Fallo")
    end

    try
        t = templateSVM("KernelFunction","gaussian", "KernelScale",28,"BoxConstraint",1,"Standa
        CoarseGaussianSVM = fitcecoc(features,labels,"Learners",t,'Coding', 'onevsone');
        [accuracy] = accuracy_result(CoarseGaussianSVM, features, labels,k_fold)
        display("CoarseGaussianSVM")
        if accuracy > maxAcc
            maxModel = CoarseGaussianSVM;
            maxAcc = accuracy;
        end
    catch ME
        display("Fallo")
    end

    try
        MediumKNN = fitcknn(features, labels,"NumNeighbors",10,"Distance","euclidean","Distance
        [accuracy] = accuracy_result(MediumKNN, features, labels,k_fold)
        display("MediumKNN")
        if accuracy > maxAcc
            maxModel = MediumKNN;
            maxAcc = accuracy;
        end
    catch ME
        display("Fallo")
    end
```

```
try
    CosineKNN = fitcknn(features, labels,"NumNeighbors",10,"Distance","cosine","DistanceWei
    [accuracy] = accuracy_result(CosineKNN, features, labels,k_fold)
    display("CosineKNN")
    if accuracy > maxAcc
        maxModel = CosineKNN;
        maxAcc = accuracy;
    end
catch ME
    display("Fallo")
end

try
    CubicKNN = fitcknn(features, labels,"NumNeighbors",10,"Distance","minkowski","DistanceW
    [accuracy] = accuracy_result(CubicKNN, features, labels,k_fold)
    display("CubicKNN")
    if accuracy > maxAcc
        maxModel = CubicKNN;
        maxAcc = accuracy;
    end
catch ME
    display("Fallo")
end

try
    WeightedKNN = fitcknn(features, labels,"NumNeighbors",10,"Distance","euclidean","Distan
    [accuracy] = accuracy_result(WeightedKNN, features, labels,k_fold)
    display("WeightedKNN")
    if accuracy > maxAcc
        maxModel = WeightedKNN;
        maxAcc = accuracy;
    end
catch ME
    display("Fallo")
end

try
    t = templateTree("MaxNumSplits" ,20);
    BoostedTrees = fitcensemble(features, labels, "Method","AdaBoostM2","Learners",t, "NumL
    [accuracy] = accuracy_result(BoostedTrees, features, labels,k_fold)
    display("BoostedTrees")
    if accuracy > maxAcc
        maxModel = BoostedTrees;
        maxAcc = accuracy;
    end
catch ME
    display("Fallo")
end

try
    t = templateTree("MaxNumSplits" ,15999);
    BaggedTrees = fitcensemble(features, labels, "Method","Bag","Learners",t, "NumLearningC
    [accuracy] = accuracy_result(BaggedTrees, features, labels,k_fold)
    display("BaggedTrees")
```

```
        if accuracy > maxAcc
            maxModel = BaggedTrees;
            maxAcc = accuracy;
        end
    catch ME
        display("Fallo")
    end

    try
        SubspaceDiscriminant= fitcensemble(features, labels, "Method","Subspace", "Learners","d
        [accuracy] = accuracy_result(SubspaceDiscriminant, features, labels,k_fold)
        display("SubspaceDiscriminant")
        if accuracy > maxAcc
            maxModel = SubspaceDiscriminant;
            maxAcc = accuracy;
        end
    catch ME
        display("Fallo")
    end

    try
        SubspaceKNN = fitcensemble(features, labels, "Method","Subspace", "Learners","knn", "Nu
        [accuracy] = accuracy_result(SubspaceKNN, features, labels,k_fold)
        display("SubspaceKNN")
        if accuracy > maxAcc
            maxModel = SubspaceKNN;
            maxAcc = accuracy;
        end
    catch ME
        display("Fallo")
    end

    try
        t = templateTree("MaxNumSplits" ,20);
        RUSBoostedTrees = fitcensemble(features, labels, "Method","RUSBoost","Learners",t, "Num
        [accuracy] = accuracy_result(RUSBoostedTrees, features, labels,k_fold)
        display("RUSBoostedTrees")
        if accuracy > maxAcc
            maxModel = RUSBoostedTrees;
            maxAcc = accuracy;
        end
    catch ME
        display("Fallo")
    end

    maxAcc
end
```

Split data

```
function [trainimages trainlabels testimages testlabels]  = splitdata(images,labels, k)
```

```
    cv = cvpartition(size(images,3),'HoldOut',k);
    idx = cv.test;
    trainimages = images(:,:,idx);
    trainlabels = labels(idx);
    testimages = images(:,:,~idx);
    testlabels = labels(~idx);
end
```

Study of model

```
function [] = study_of_model(model, features, labels)
    [predictlabels,score] = predict(model,features);
    confusionchart(labels,predictlabels);
    accuracy = nnz(labels ~= predictlabels)/length(labels)
    ClassNames = int2str(model.ClassNames);
    figure;
    for i = 1:10
        [X,Y,T,AUC,OPTROCPT] = perfcurve(labels,score(:,i),ClassNames(i));
        if (i ==2)
            hold on;
        end
        plot(X,Y);
        text(max(OPTROCPT(1)),max(OPTROCPT(2)),ClassNames(i),'FontSize',10);
    end
    xlabel('False positive rate');
    ylabel('True positive rate');
    title('ROC Curve');
    legend(ClassNames);
    hold off;
end
function [accuracy] = accuracy_result(model, features, labels, k_fold)
    partitionedModel = crossval(model, 'KFold', k_fold);
    accuracy = 1 - kfoldLoss(partitionedModel, 'LossFun', 'ClassifError');
end
```

Study of data

```
function [featuresNorm,  pcs, scrs] = study_of_data(features, labels)
    %normalize data in A, center de data and scale to have standard
    %deviation 1
    featuresNorm = normalize(features);
    %PCA pcs = principal components coefficients (Matrix changes of basis),
    %scrs = principapl scores (featuresNorm*pcs)
    %matrix, pexp = percentage of total variance explained
    [pcs,scrs,~,~,pexp] = pca(featuresNorm);
    %Pareto
    figure
    H = pareto(pexp);
    title('variance srcs')
    %PCA scatter 3 first columns
    figure
    scatter3(scrs(:,1),scrs(:,2),scrs(:,3),9,labels);
    xlabel('scrs(:,1)');
```

```
    ylabel('scrs(:,2)');
    zlabel('scrs(:,3)');
    figure
    parallelcoords(scrs,"Group",labels);
end
```

All points feature

```
function [features] = extractfeaturesallpoints(images)
    [D1, D2, nimages] = size(images);
    features = zeros(nimages, D1*D2);
    for i = 1:nimages
        im = images(:,:,i);
        features(i,:) = im(:);
    end
end
```

Shape Context feature

```
function [features] = extractfeatureswithShapeContext_multicenters(images, centers, radi, nbins
    [n_centers ~] = size(centers);
    [~, ~, nimages] = size(images);
    features = zeros(nimages, nbins_r*nbins_theta*n_centers);
    for i = 1:nimages
        im = images(:,:,i);
        feature = zeros(1,nbins_r*nbins_theta*n_centers);
        for j = 1:n_centers
           feature_aux =  ShapeContext(im, centers(j,:), radi, nbins_r, nbins_theta);
           feature(((j-1)*nbins_r*nbins_theta)+1:(j*nbins_r*nbins_theta)) = feature_aux;
        end
        features(i,:) = feature;
    end
 end

function [feature] = ShapeContext(im, center, radi, nbins_r, nbins_theta)
    %Hago un resize de la imagen [200 200]
    [~, points] = gecontour(im);
    [D1 D2] = size(points);
    %Calculate euclidean distance
    distances = points-center;
    distances = sqrt((distances(:,1).^2)+(distances(:,2).^2));
    %calculate r_bin
    r_bin_edges = logspace(log10(1), log10(radi), nbins_r);
    r_bin = double(zeros(D1,1));
    for i=1:length(r_bin_edges)
        s = (distances<r_bin_edges(1,i));
        r_bin = r_bin + double(s);
    end

    %calculate angle between two points
    distance = points-center;
    angle = atan(distance(:,2)./distance(:,1));
```

```
    min_angle = min(angle);
    if (min_angle < 0)
        angle = (angle+abs(min(angle)));
    end
    angle = (angle*2*pi)./max(angle);

    %calculate theta_bins
    theta_bin_edge = linspace(0,2*pi,nbins_theta);
    theta_bin = double(zeros(D1,1));
    for i=1:length(theta_bin_edge)
        s = (angle<theta_bin_edge(1,i));
        theta_bin = theta_bin + double(s);
    end

    feature = double(zeros(1,nbins_r*nbins_theta));
    k = 1;
    for i = 1:nbins_r
       for j = 1:nbins_theta
           features_aux = (r_bin == i) & (theta_bin == j);
           feature(k)= sum(features_aux);
           k = k+1;
       end
    end

end
function [contour, points] = gecontour(im)
    %lo mismo que im = im -imclose(im) o cany; Hago un resize de la imagen [200
    %200]
    im = imbinarize(im);
    im = imresize(im, [200 200]);
    [D1 D2] = size(im);
    contour = logical(zeros(D1,D2));
    points = [];
    for i = 2:(D1-1)
        for j = 2:(D2-1)
            if ((~im(i,j-1) || ~im(i,j+1) || ~im(i+1,j) || ~im(i-1,j)) && im(i,j))
               contour(i,j) = true;
               points = [points; [i j]];
            end
        end
    end
end
```

Fast-Fourier(Características de contorn)

```
function [features] = extractfeatureswithfourier(images,Ndescriptors)
    [~, ~, nimages] = size(images);
    features = zeros(nimages, Ndescriptors);
    for i = 1:nimages
        im = images(:,:,i);
        features(i,:) = descriptor_fourier(im, Ndescriptors,false);
    end
end
```

```
function [features] = extractfeatureswithfourier_with_holes(images,Ndescriptors)
    [~, ~, nimages] = size(images);
    features = zeros(nimages, Ndescriptors);
    for i = 1:nimages
        im = images(:,:,i);
        [D1 D2] = size(features);
        descriptor = descriptor_fourier(im, Ndescriptors,true);
        [d1 d2] = size(descriptor);
        if D2 < d2
           features_aux = zeros(D1,d2);
           features_aux(:,1:D2) = features;
           features = features_aux;
        elseif (d2<D2)
           descriptor_aux = zeros(1,D2);
           descriptor_aux(1,1:d2) = descriptor;
           descriptor = descriptor_aux;
        end
        features(i,:) = descriptor;
    end
end

function [descriptor] = descriptor_fourier(im, Ndescriptors, forats)
    [d1 d2 d3] = size(im);
    im = imbinarize(im);
    im = imresize(im, [300 300]);
    % 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
    % 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) + 1i*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 els descriptros

    descriptor = descriptor(1:Ndescriptors);
    %log del resultat es opcional, absoluto porque la classificacion en
    %matlab no puede ser imaginario
    descriptor = log(abs(descriptor));
    descriptor = descriptor';
    if (~forats)
           return
    end
```

```
    %descriptor fourier forats
    im = imcomplement(im);
    mark=true(size(im));
    mark(2:end-1,2:end-1) = 0;
    dilc = imreconstruct(mark,im);
    res = ~dilc;
    im = xor(res,im);
    im = imcomplement(im);
    [BW, numberOfObject] = bwlabel(im);
    for i = 1:numberOfObject
        descriptors_aux = descriptor_fourier(double(BW == i), Ndescriptors, false);
        descriptor = [descriptor descriptors_aux];
    end
end
```

HOG Features(Histogram of oriented gradients)

https://es.mathworks.com/help/supportpkg/android/ref/digit-classification-using-hog-features-on-mnist-database.html

```
function [features] = extractfeatureswithHOG(images, cellSize)
    im = images(:,:,1);
    im = imbinarize(im);
    im = imresize(im, [28 28]);
    hogFeatureSize = length(extractHOGFeatures(im, 'CellSize', cellSize));
    [~, ~, nimages] = size(images);
    features = zeros(nimages, hogFeatureSize);
    for i = 1:nimages
        im = images(:,:,i);
        im = imbinarize(im);
        im = imresize(im, [28 28]);
        [feature , image] = extractHOGFeatures(im, 'CellSize', cellSize);
        features(i,:) = feature;
    end
end
```

Wavelet Scattering

https://es.mathworks.com/help/wavelet/examples/digit-classification-with-wavelet-scattering.html

```
function [features] = extractfeatureswithWaveletScattering(images)
    sf = waveletScattering2('ImageSize', [28 28]);
    [~, ~, nimages] = size(images);
    for i = 1:nimages
        im = imresize(images(:,:,i),[28 28]);
        if (i == 1)
           feature  = helperScatImages(sf,im);
           l = length(feature);
           features = zeros(nimages, l);
           features(1,:) = feature';
        else
```

```
            features(i,:)  = helperScatImages(sf,im)';
        end
    end
end
function features = helperScatImages(sf,x)
% This function is only to support examples in the Wavelet Toolbox.
% It may change or be removed in a future release.
% Copyright 2018 MathWorks
    smat = featureMatrix(sf,x,'transform','log');
    features = mean(mean(smat,2),3);
end
```

Number of Corners (Harris Corner Detector)

```
function [features] = extractfeatureswithHarrisCorner(images)
    [~, ~, nimages] = size(images);
    features = zeros(nimages, 1);
    for i = 1:nimages
        im = images(:,:,i);
        im = imbinarize(im);
        corners =  detectHarrisFeatures(im);
        features(i,:) = length(corners);
    end
end
```

Propietats geometricas

```
function [features] = geometricpropieties(images)
    [~, ~, nimages] = size(images);
    features = zeros(nimages,2);
    %Area y Area de les concavitats de la figura
    for i = 1:nimages
        im = images(:,:,i);
        im = imbinarize(im);
        features(i,1) = sum(im(:));
        CH = bwconvhull(im);
        feature = sum(CH(:));
        features(i,2) = sum(feature)- features(i,1);
    end
    %Numero de forats
    for i = 1:nimages
        im = images(:,:,i);
        im = imbinarize(im);
        im = imcomplement(im);
        mark=true(size(im));
        mark(2:end-1,2:end-1) = 0;
        dilc = imreconstruct(mark,im);
        res = ~dilc;
        im = xor(res,im);
        im = imcomplement(im);
        [~, numberOfObject] = bwlabel(im);
        features(i,3) = numberOfObject;
    end
```

```
end
```

K-Means

```
function [features] = extractfeatureswithKMeans(images)
    NClusters = 8;
    [MAXFILA, MAXCOL, nimages] = size(images);
    features = zeros(nimages, NClusters*2);
    [A,B] = meshgrid((1:20),(1:20));
    c=cat(2,A',B');
    d=reshape(c,[],2);
    for i = 1:nimages
        im = images(:,:,i);
        im = imbinarize(im);
        %im
        %figure,imshow(im)
        %label = double(reshape(im,MAXFILA*MAXCOL,1));
        label = reshape(im,MAXFILA*MAXCOL,1);

        d1 = d(:,1);
        d2 = d(:,2);
        d1 = d1(label == 1);
        d2 = d2(label == 1);
        aux = [d1 d2];
        [~, clusterC] = kmeans(aux,NClusters);
        %[cluster_idx, clusterC] = kmeans(aux,NClusters);
        %label(label == 1) = cluster_idx;
        %eti=reshape(label,MAXFILA,MAXCOL);
        %figure,imshow(eti,[]),colormap(colorcube), title('imatge etiquetada')
        centreAux = sortrows(clusterC,1);
        %feature = zeros(1,length(centreAux));
        %for cont = 1:centreAux
        %    feature(cont) = norm(centreAux(cont,:));
        %end
        %features(i,:) = sort(feature);
        features(i,:) = reshape(centreAux,[],1);
    end
end


function [features] = extractfeatureswithKMeansOrdered(images,trainlabels)
    NClusters = 8;
    [MAXFILA, MAXCOL, nimages] = size(images);
    features = zeros(nimages, NClusters*2);
    [A,B] = meshgrid((1:20),(1:20));
    c=cat(2,A',B');
    d=reshape(c,[],2);
    ordenCero = [6.16666666666667,9.33333333333333;10.6153846153846,5.23076923076923;16.3333333
    ordenUno = [2.87500000000000,16.1250000000000;5.25000000000000,14.7500000000000;7.2857142857
    ordenDos = [5.88888888888889,8.66666666666667;3.68750000000000,13.7500000000000;8.0909090909
    ordenTres = [3.88235294117647,8.94117647058824;3.33333333333333,14.1904761904762;6.4000000000
    ordenCuatro = [5,1.50000000000000;8.28571428571429,1.42857142857143;12.3636363636364,2.8181
    ordenCinco = [2.50000000000000,18;3.70588235294118,13;4.41176470588235,7.70588235294118;7.8
```

```
    ordenSeis = [2.23076923076923,11.5384615384615;6.16666666666667,8.41666666666667;10,6.84615
    ordenSiete = [8.47619047619048,5.61904761904762;5.75000000000000,10.9166666666667;5.5000000
    ordenOcho = [5.84210526315790,9.47368421052632;10.7500000000000,9.62500000000000;14.5000000
    ordenNueve = [5.72727272727273,15.2727272727273;4.50000000000000,11.5000000000000;6.6666666

    for i = 1:nimages
        switch trainlabels(i)
            case 0
                orden = sqrt(sum(ordenCero.^2,2));
            case 1
                orden = sqrt(sum(ordenUno.^2,2));
            case 2
                orden = sqrt(sum(ordenDos.^2,2));
            case 3
                orden = sqrt(sum(ordenTres.^2,2));
            case 4
                orden = sqrt(sum(ordenCuatro.^2,2));
            case 5
                orden = sqrt(sum(ordenCinco.^2,2));
            case 6
                orden = sqrt(sum(ordenSeis.^2,2));
            case 7
                orden = sqrt(sum(ordenSiete.^2,2));
            case 8
                orden = sqrt(sum(ordenOcho.^2,2));
            case 9
                orden = sqrt(sum(ordenNueve.^2,2));
        end
        im = images(:,:,i);
        im = imbinarize(im);
        %im
        %figure,imshow(im)
        label = double(reshape(im,MAXFILA*MAXCOL,1));
        %label = reshape(im,MAXFILA*MAXCOL,1);

        d1 = d(:,1);
        d2 = d(:,2);
        d1 = d1(label == 1);
        d2 = d2(label == 1);
        aux = [d1 d2];
        [~, clusterC] = kmeans(aux,NClusters);
        %[cluster_idx, clusterC] = kmeans(aux,NClusters);
        %label(label == 1) = cluster_idx;
        %eti=reshape(label,MAXFILA,MAXCOL);
        %figure,imshow(eti,[]),colormap(colorcube), title('imatge etiquetada')

        pos = perms(1:NClusters);
        minPos = 1;
        minValor = Inf;
        for j=1:length(pos)
            clusterOrd = clusterC(pos(j,:),:);
            euclDist = sqrt(sum(clusterOrd.^2,2));
            mse = mean((orden-euclDist).^2);
            if minValor > mse
```

```
                minValor = mse;
                minPos = j;
            end
        end
        clusterC = clusterC(pos(minPos,:),:);
        %{
        s = (1:NClusters)';
        t = (1:.05:NClusters)';
        x = clusterC(:,1);
        y = clusterC(:,2);
        u = pchiptx(s,x,t);
        v = pchiptx(s,y,t);
        %clf reset
        plot(x,y,'.',u,v,'-');
        %}
        %features = clusterC;
        features(i,:) = reshape(clusterC,16,1);

    end
end
```