Reconeximent Automàtic de Digits Manuscrits

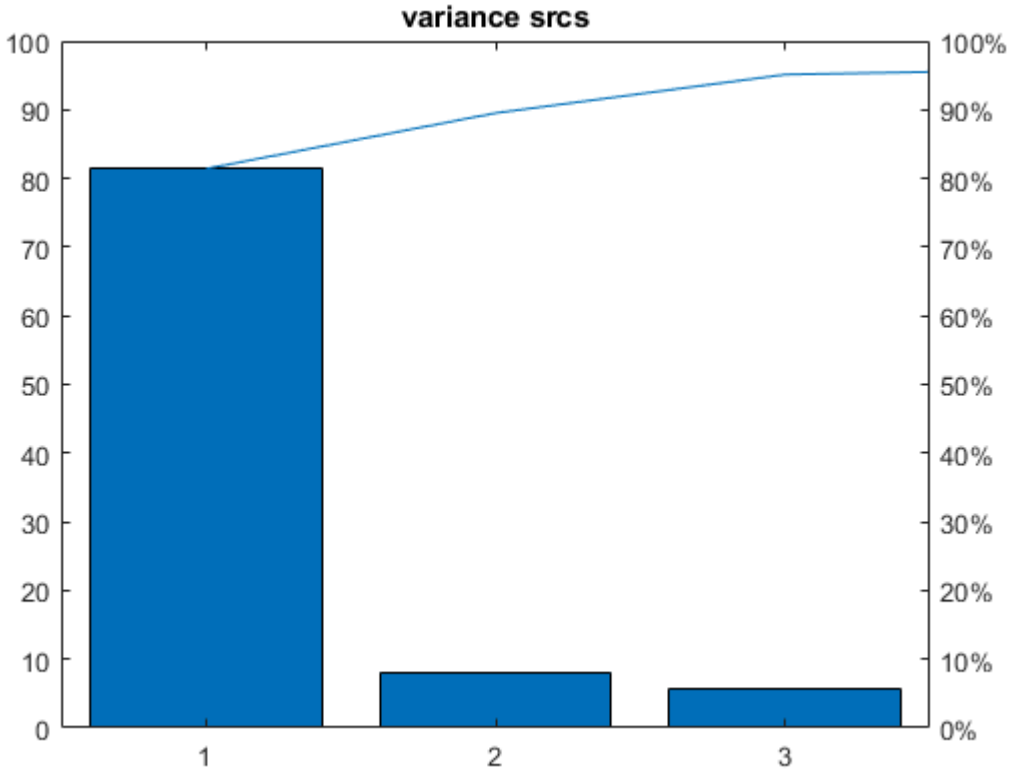
```
[images labels] = readMNIST('train-images.idx3-ubyte', 'train-labels.idx1-ubyte',20000 ,0);
[trainimages trainlabels testimages testlabels]  = splitdata(images,labels, 0.8);
```

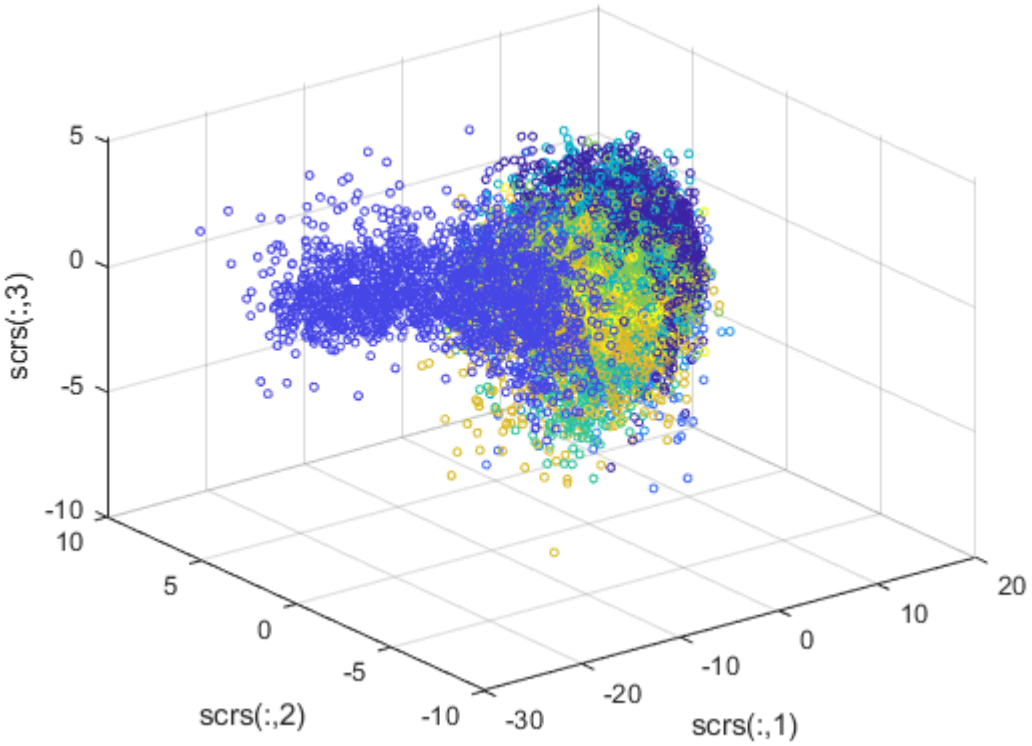
Wavelet Study

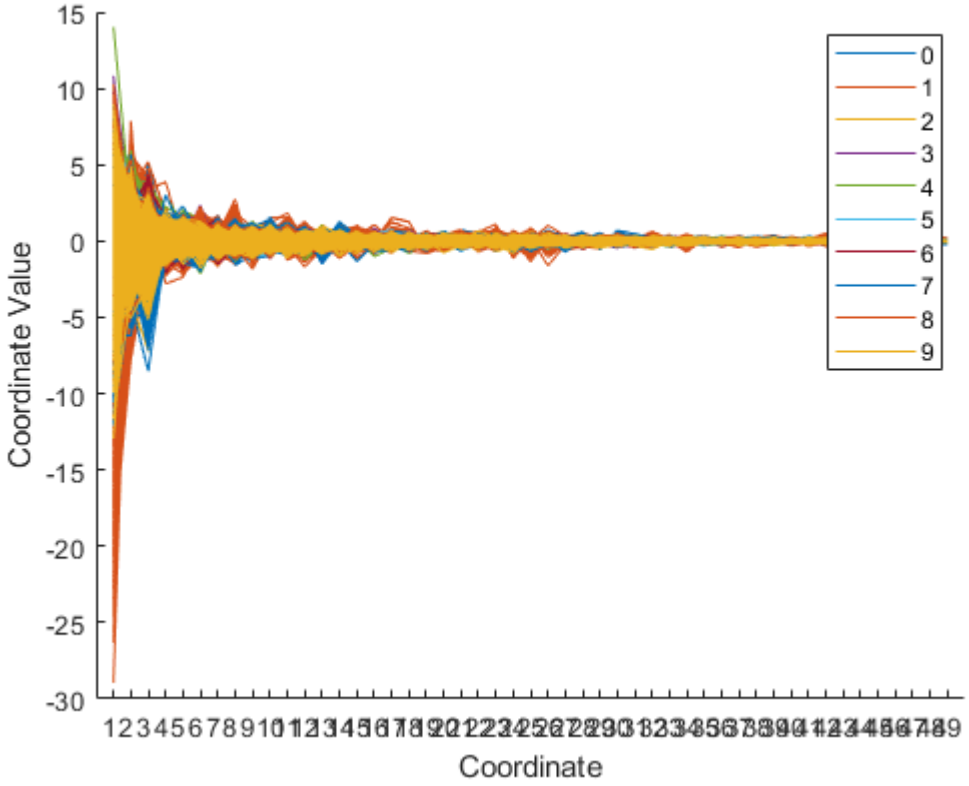
```
%[trainimages trainlabels testimages testlabels]  = splitdata(images,labels, 0.8);
[trainfeatures] = extractfeatureswithWaveletScattering(trainimages);
```

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]
```

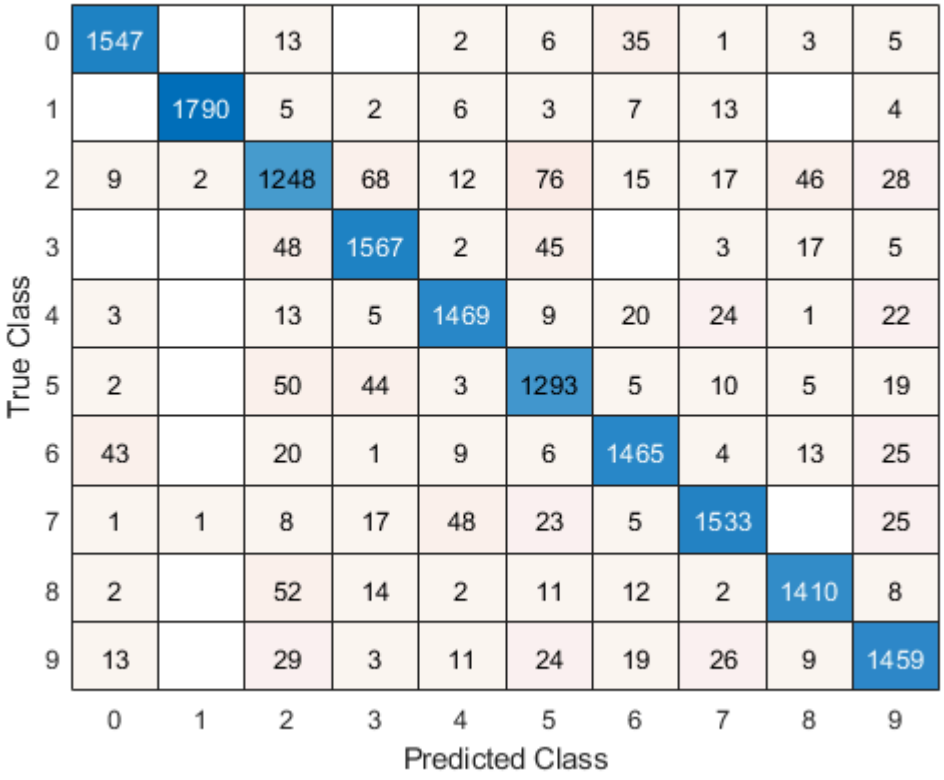
```
i=5
```

```
i = 5
```

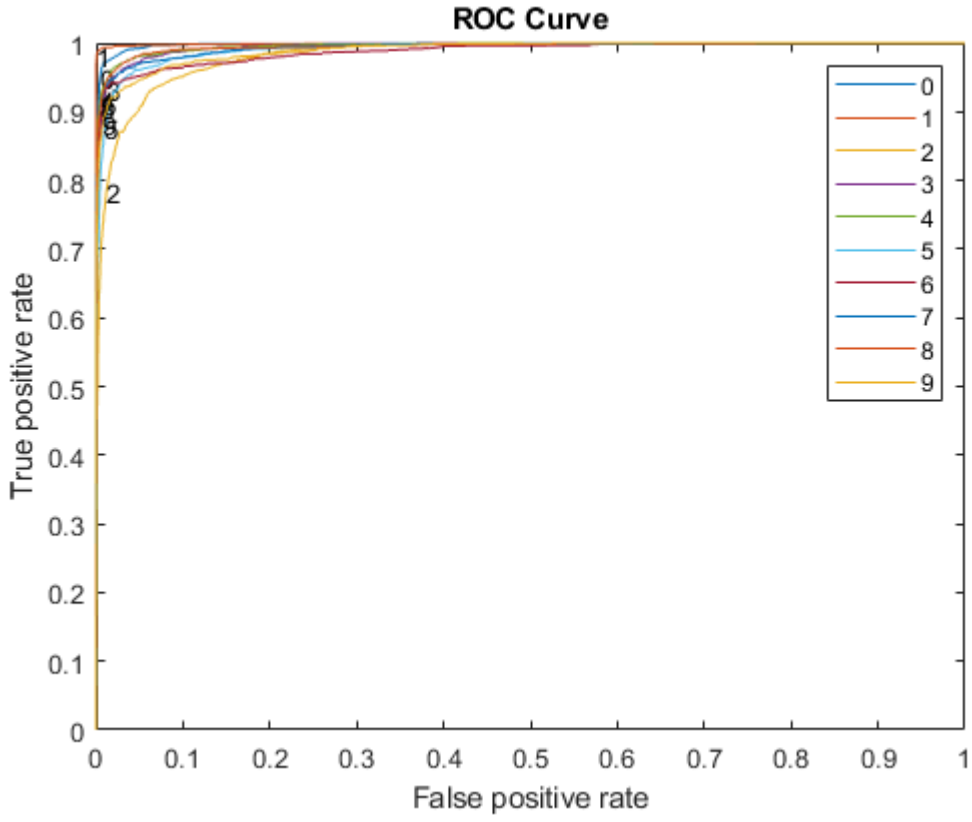
```
[maxModel] = classification_learner(trainscrs, trainlabels,i);
```

```
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
```

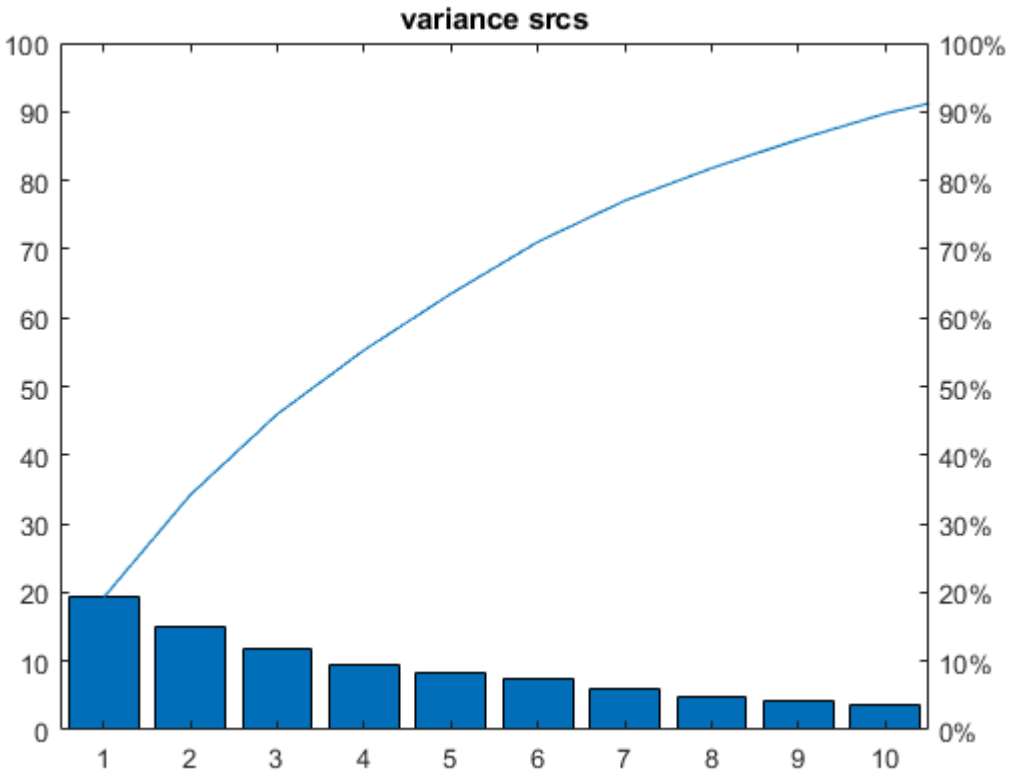
```
study_of_model(maxModel,trainscrs,trainlabels)
```

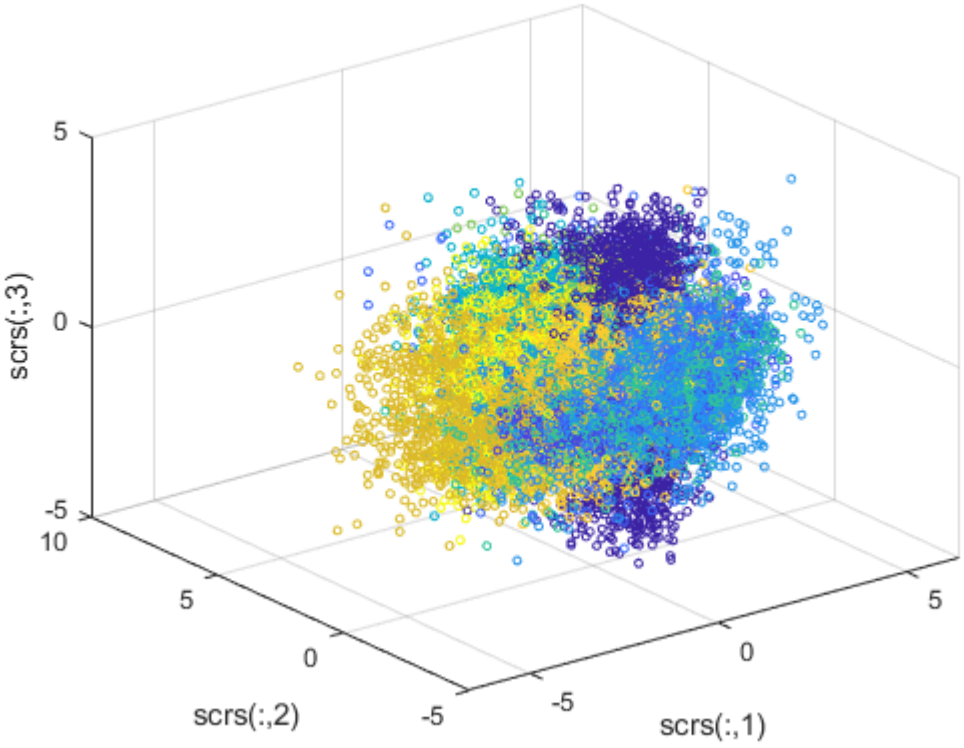
K-Means

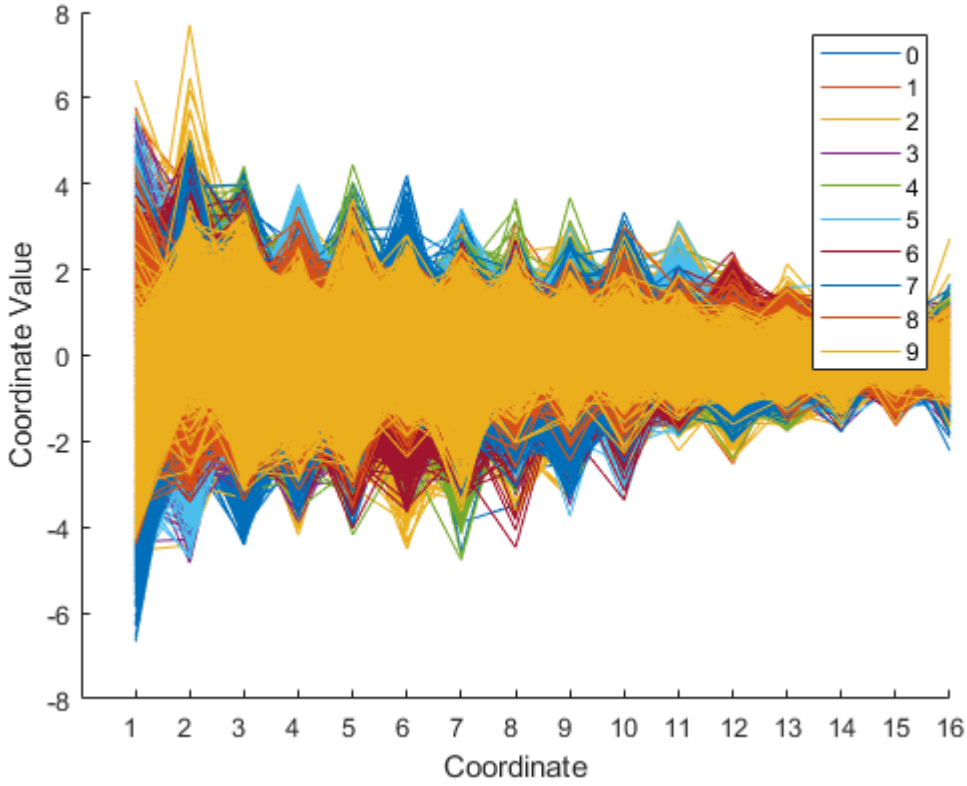
```
%[trainimages trainlabels testimages testlabels]  = splitdata(images,labels, 0.8);
[trainfeatures] = extractfeatureswithKMeans(trainimages);
```

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
```

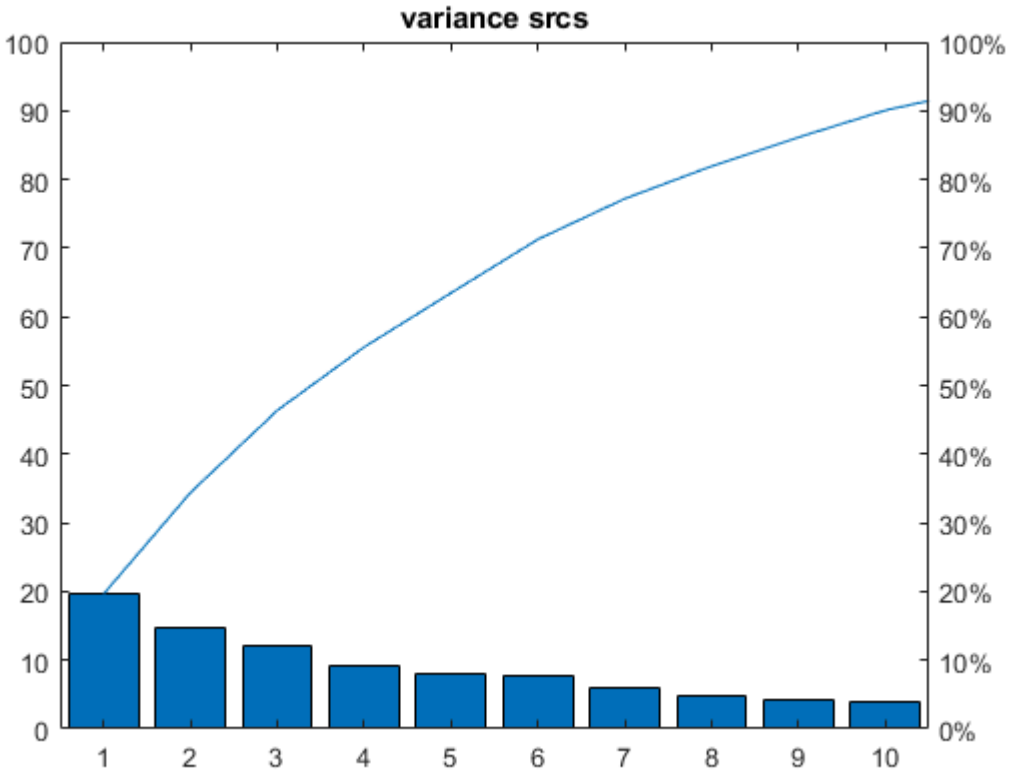
```
i = 5
```

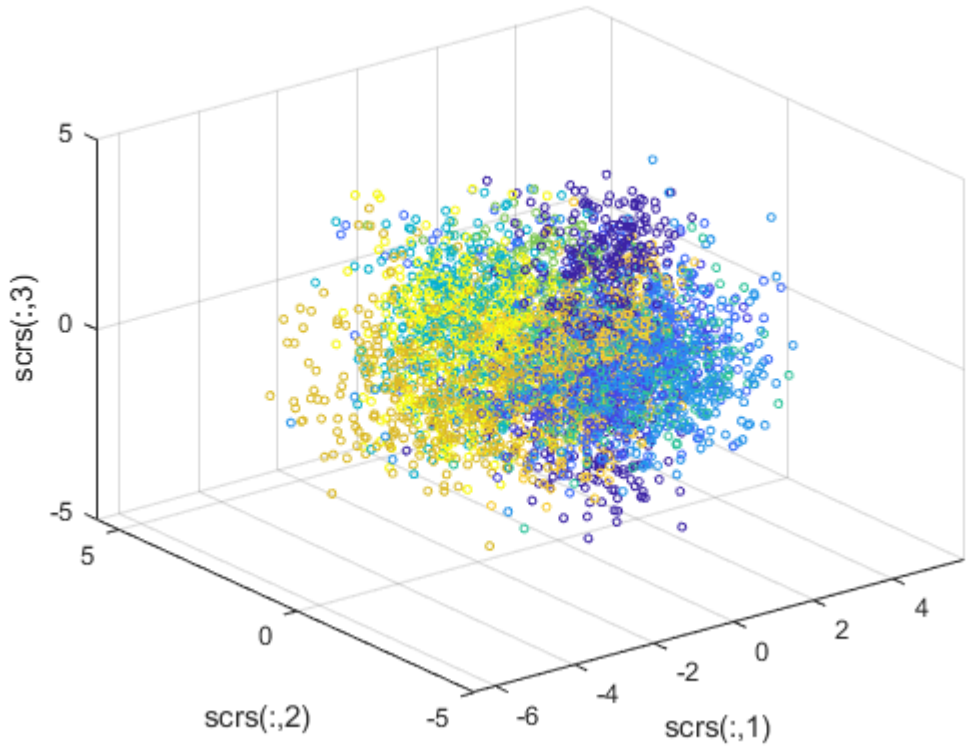
```
[maxModel] = classification_learner(trainscrs, trainlabels,i);
```

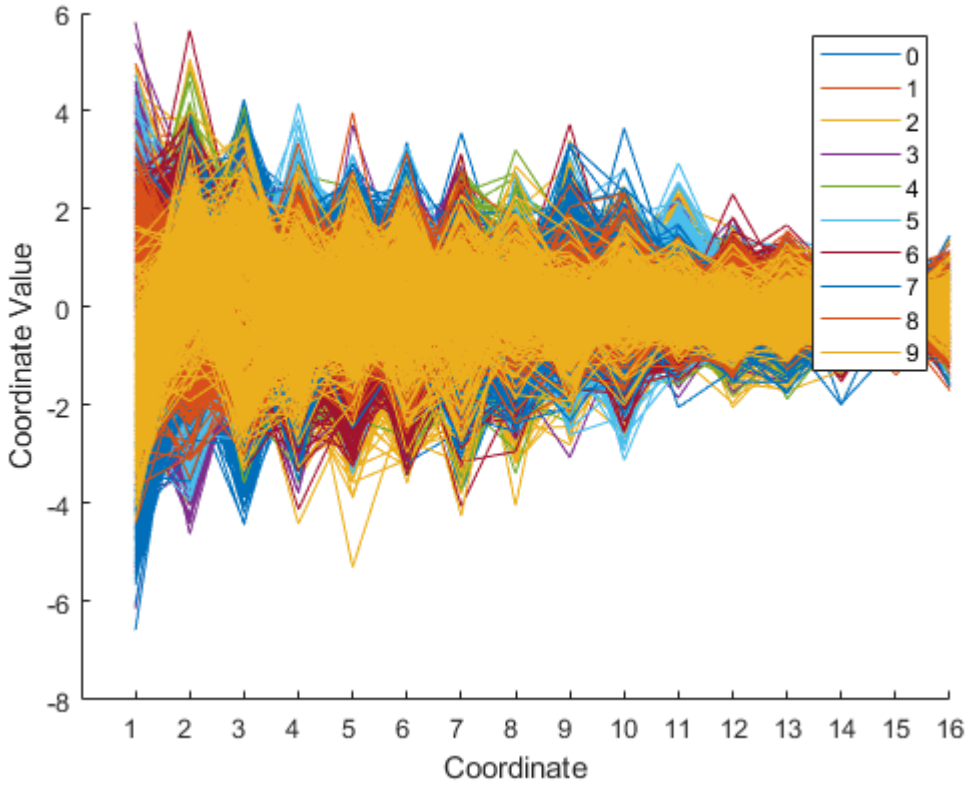
```
accuracy = 0.5372
    "FinetreeModel"
accuracy = 0.4392
    "MediumtreeModel"
accuracy = 0.3119
    "CoarsetreeModel"
accuracy = 0.6363
    "LinearDiscriminant"
accuracy = 0.8440
    "QuadraticDiscriminant"
accuracy = 0.7408
    "KernelNaiveBayes"
accuracy = 0.7373
    "GaussianNaiveBayes"
accuracy = 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);
```

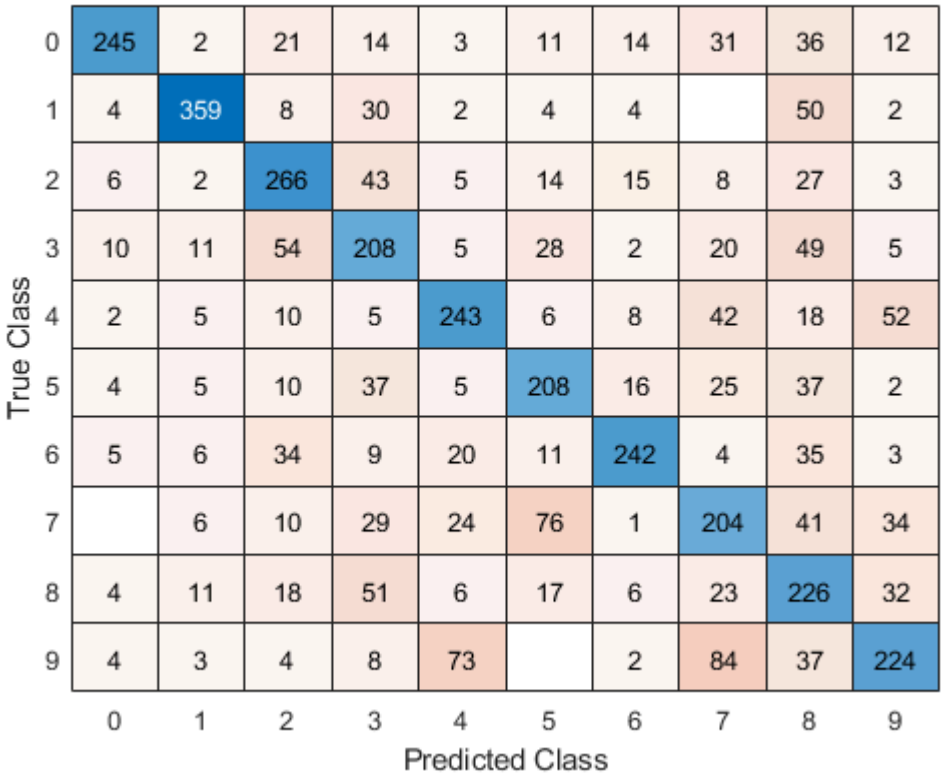
```
[testfeaturesNorm, testpcs, testscrs] = study_of_data(testfeatures, testlabels);
```


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

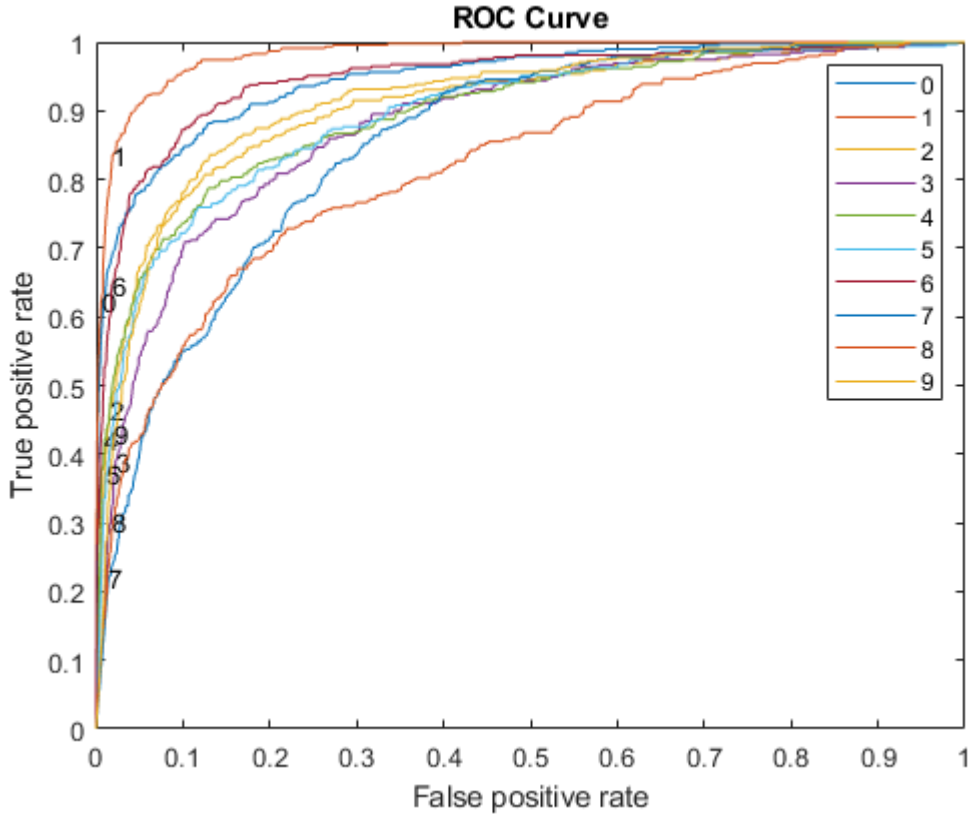

scrs(:,3)
5
0
-5
5
0
-5
scrs(:,2)
-6
-4
-2
0
2
4
scrs(:,1)




```
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];
```

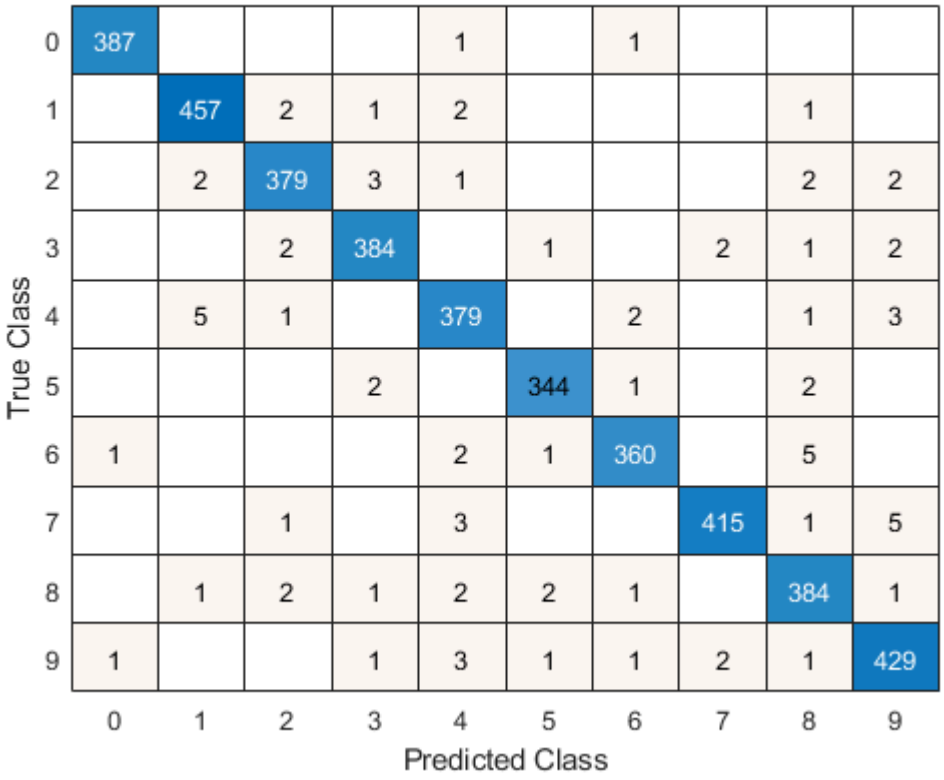
```
%for i = [2 5 10 50]
    i=5
```

```
i = 5
```

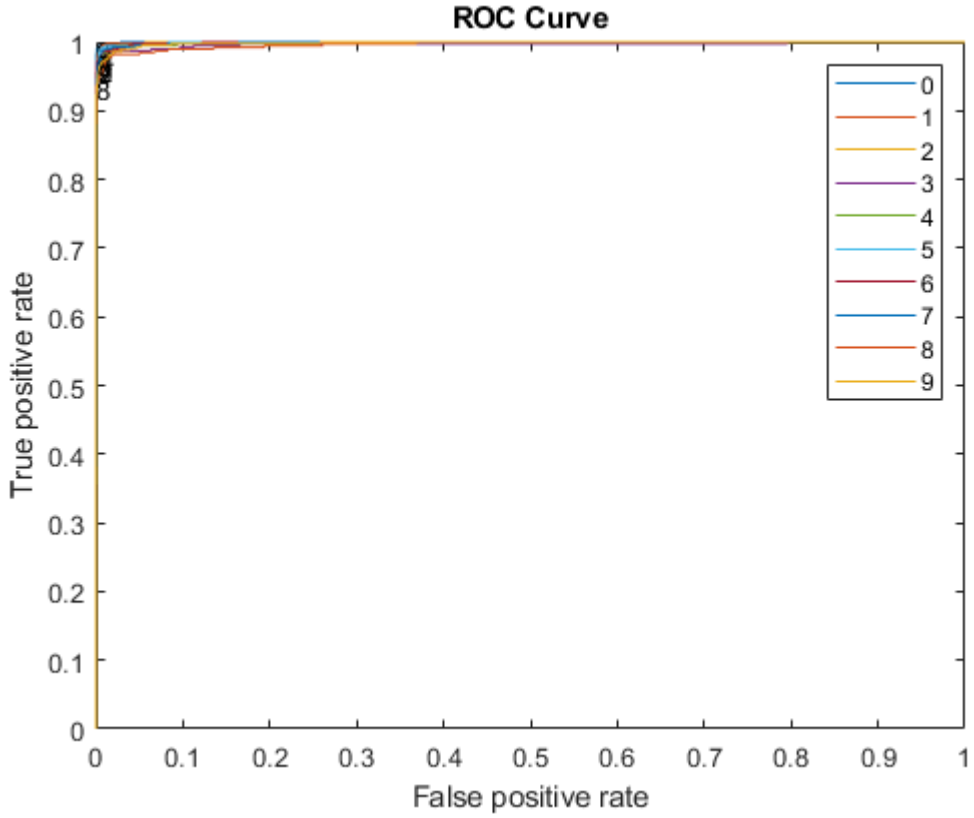
```
    %[maxModel] = classification_learner(trainscrs, trainlabels,i);
    [maxModel] = classification_learner(trainfeatures, trainlabels,i);
```

```
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
```

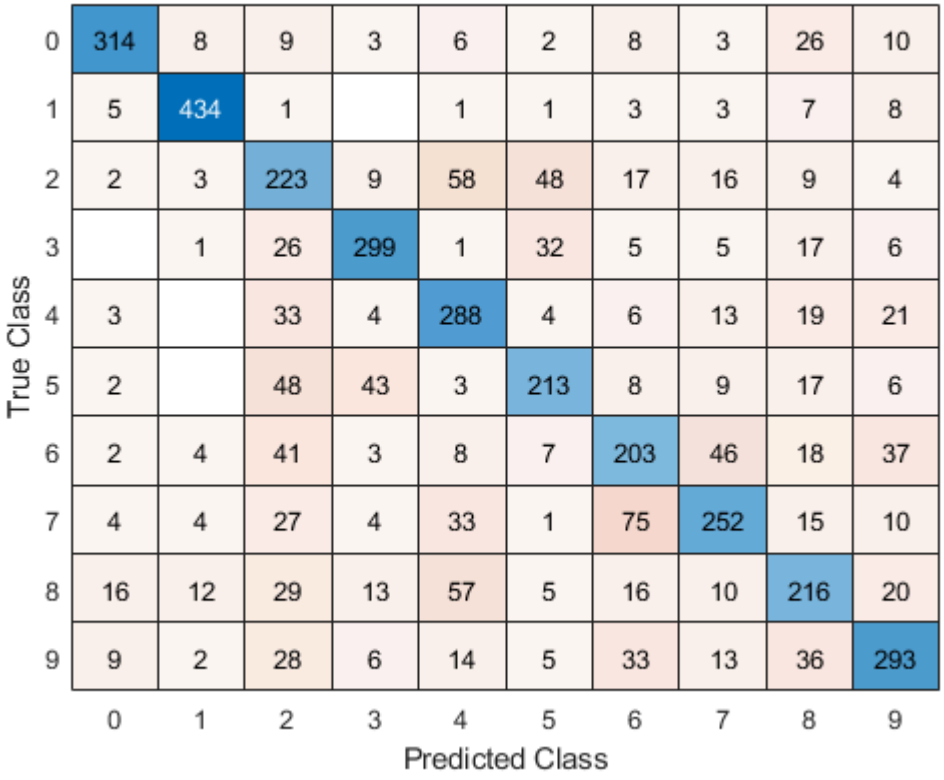
```
    %[maxModel] = classification_learner(trainscrs, trainlabels,i);
    [maxModel] = classification_learner(trainfeatures, trainlabels,i);
```

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

    "Fallo"
```

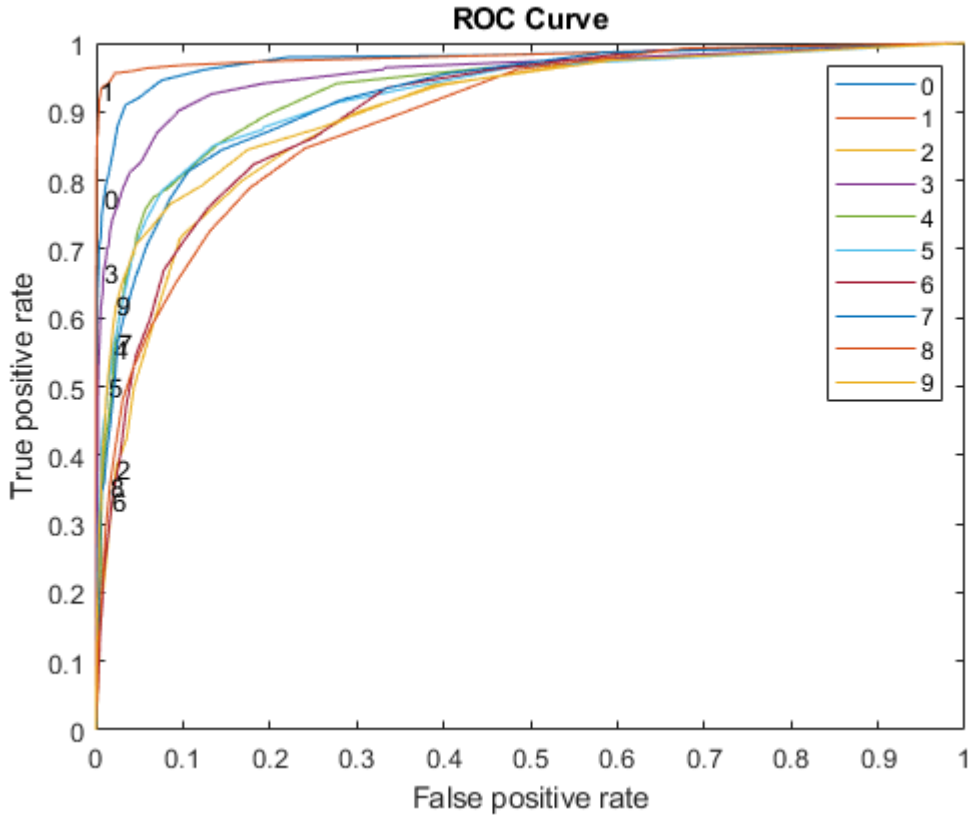
```
    "Fallo"
accuracy = 0.5230
    "KernelNaiveBayes"
accuracy = 0.4302
    "GaussianNaiveBayes"
accuracy = 0.1003
    "LinearSVM"
accuracy = 0.1003
    "QuadraticSVM"
accuracy = 0.1003
    "CubicSVM"
accuracy = 0.1003
    "FineGuassianSVM"
accuracy = 0.1003
    "MediumGuassianSVM"
accuracy = 0.1003
    "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
```

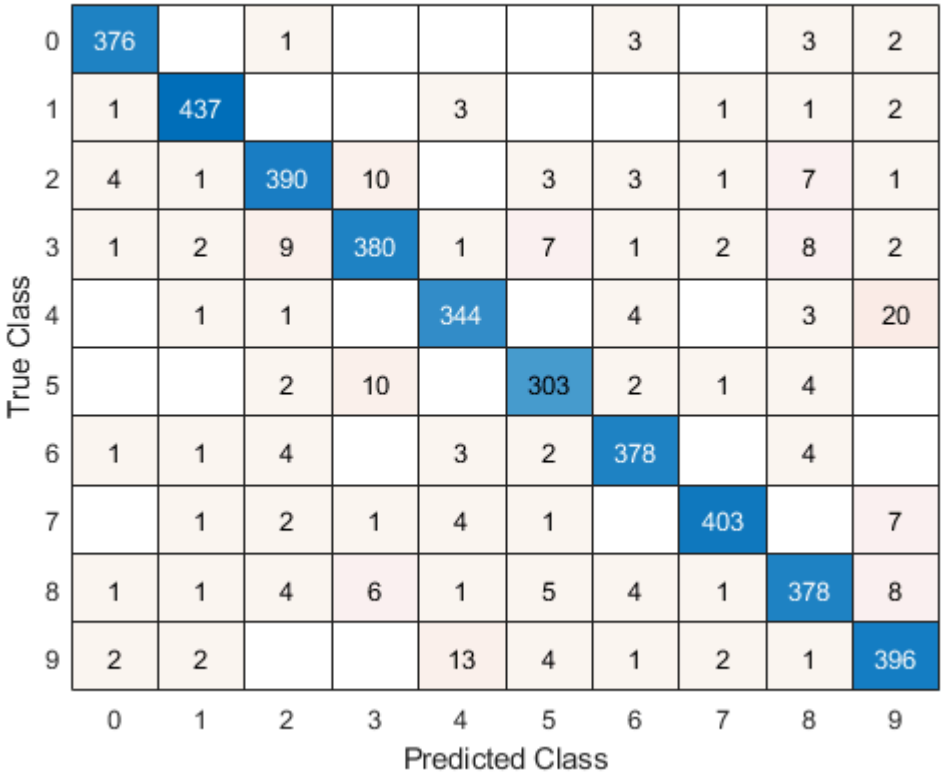
```
i = 5
```

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

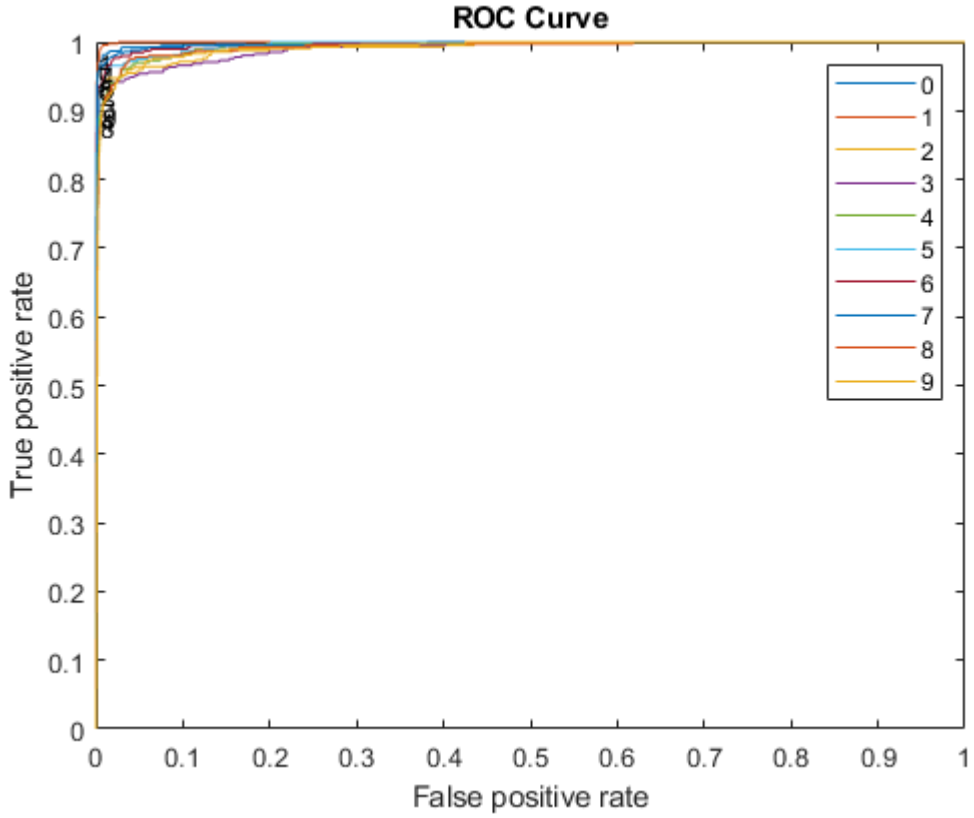
```
accuracy = 0.6541
    "FinetreeModel"
accuracy = 0.5312
    "MediumtreeModel"
accuracy = 0.3479
    "CoarsetreeModel"
accuracy = 0.8622
    "LinearDiscriminant"
accuracy = 0.9327
    "QuadraticDiscriminant"
accuracy = 0.5891
    "KernelNaiveBayes"
accuracy = 0.5599
    "GaussianNaiveBayes"
accuracy = 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)
```

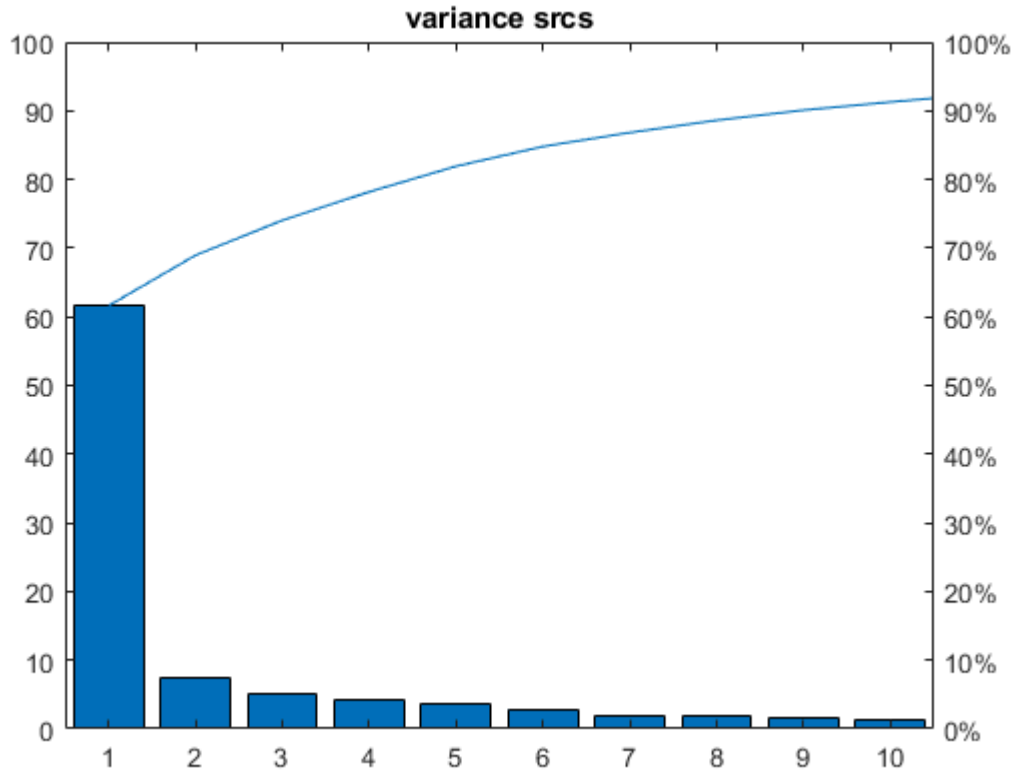


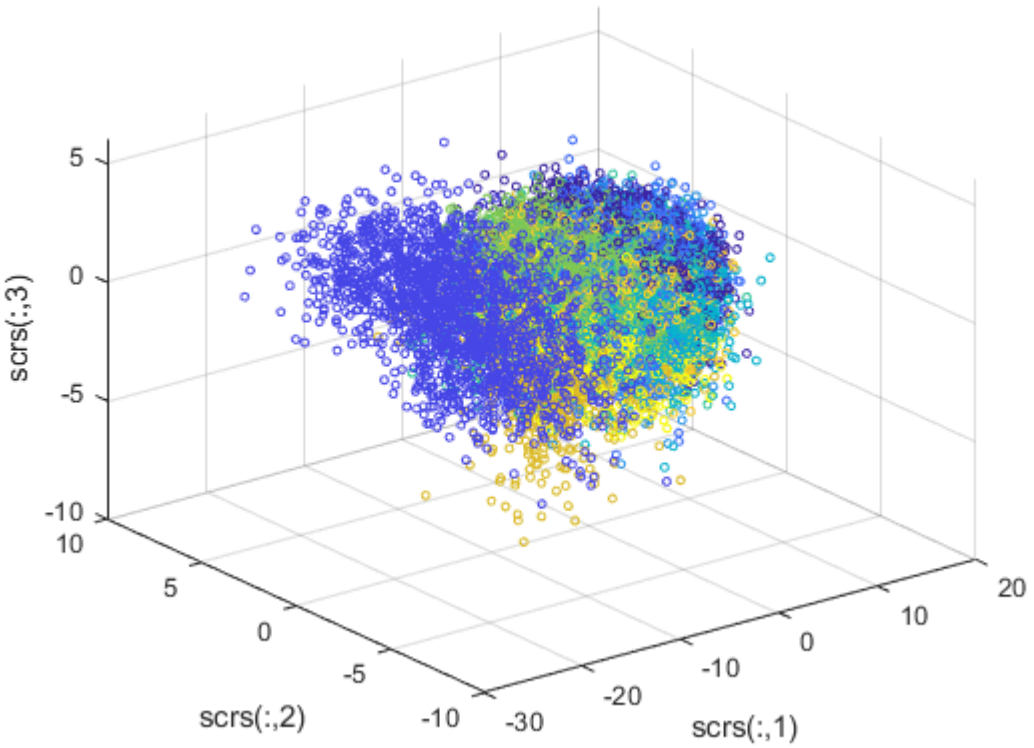

```
accuracy = 0.0538
```

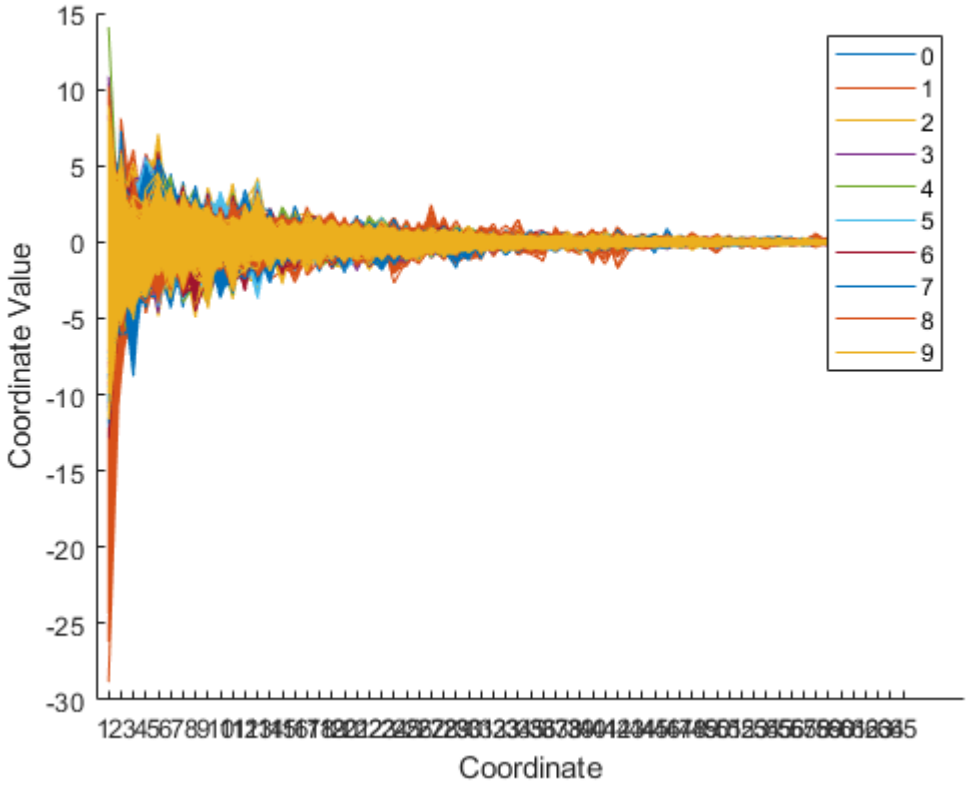


With PCA

```
[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);
```

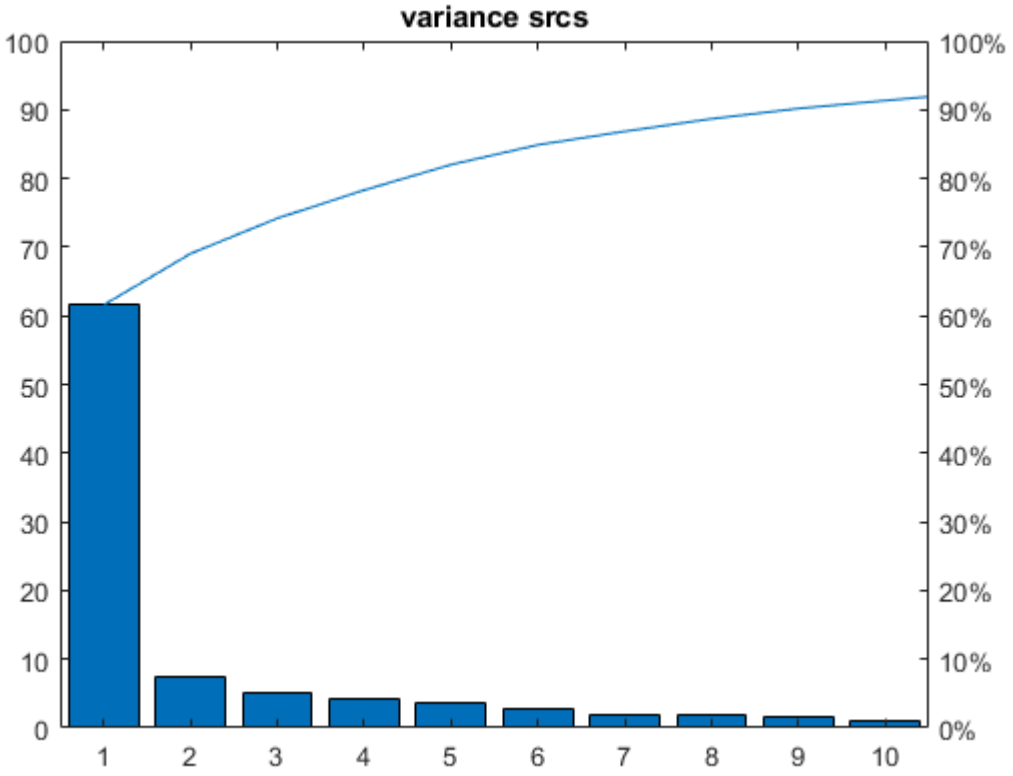
```
accuracy = 0.6549
    "FinetreeModel"
accuracy = 0.5501
    "MediumtreeModel"
accuracy = 0.3417
    "CoarsetreeModel"
accuracy = 0.8626
    "LinearDiscriminant"
accuracy = 0.9334
    "QuadraticDiscriminant"
accuracy = 0.8445
    "KernelNaiveBayes"
accuracy = 0.8515
    "GaussianNaiveBayes"
accuracy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

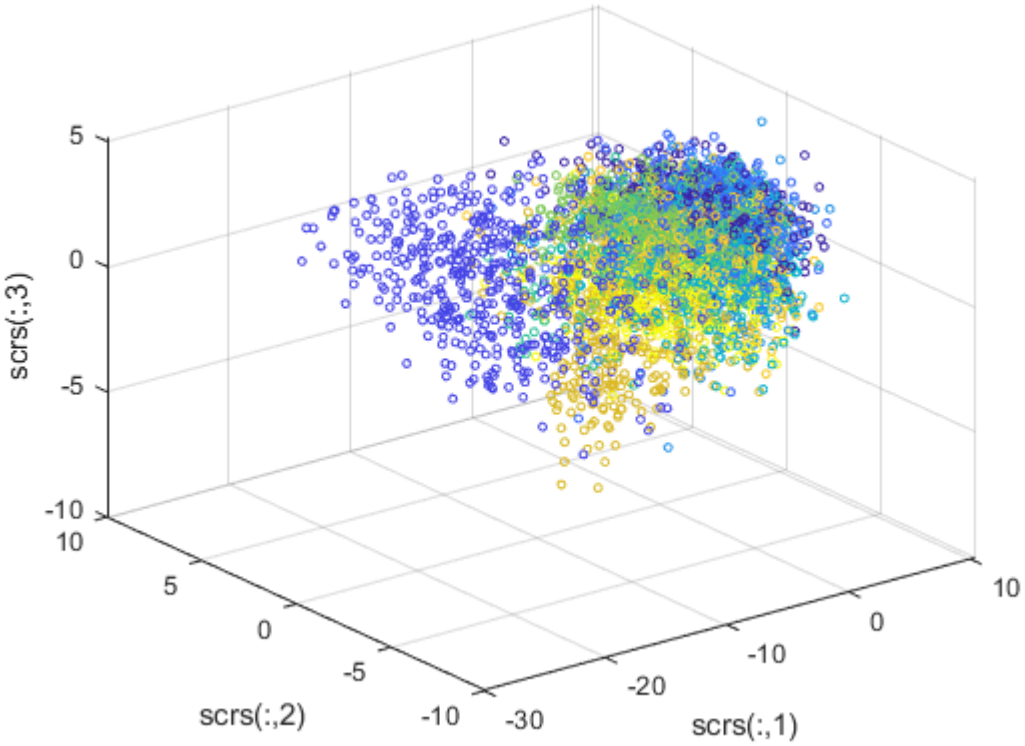
```
    "CoarseGaussianSVM"
accuracy = 0.8481
    "MediumKNN"
accuracy = 0.8847
    "CosineKNN"
accuracy = 0.8457
    "CubicKNN"
accuracy = 0.8513
    "WeightedKNN"
accuracy = 0.5911
    "BoostedTrees"
accuracy = 0.8195
    "BaggedTrees"
accuracy = 0.8437
    "SubspaceDiscriminant"
accuracy = 0.8921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

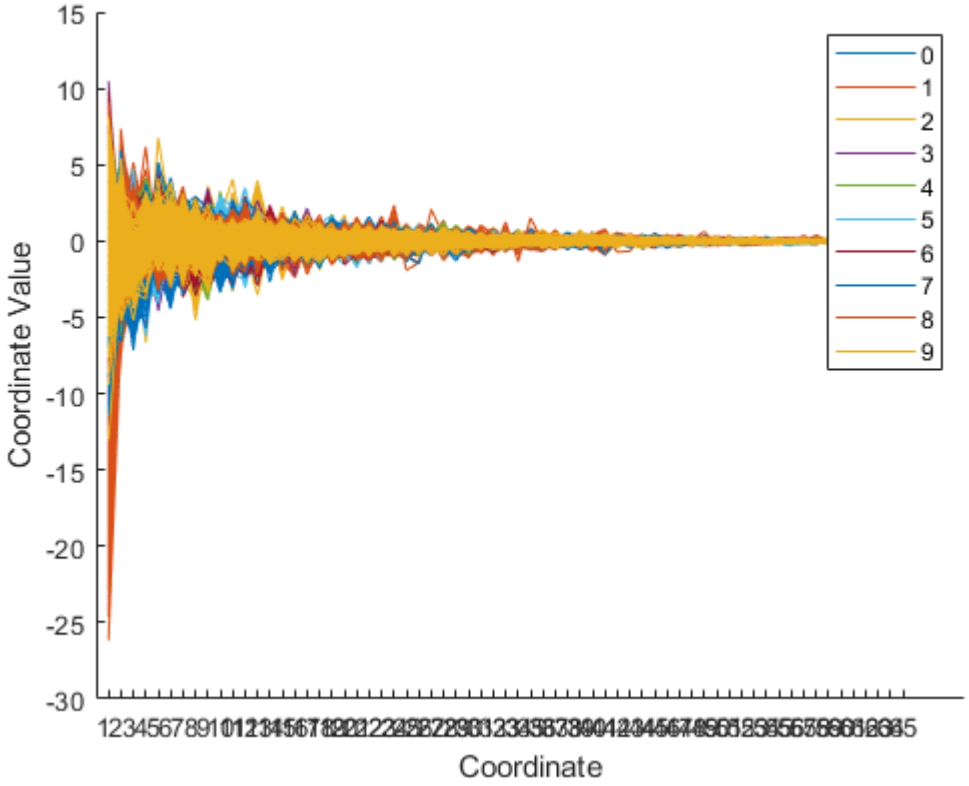
```
    %[maxModel] = classification_learner(trainfeatures, trainlabels,i);
%end
```

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

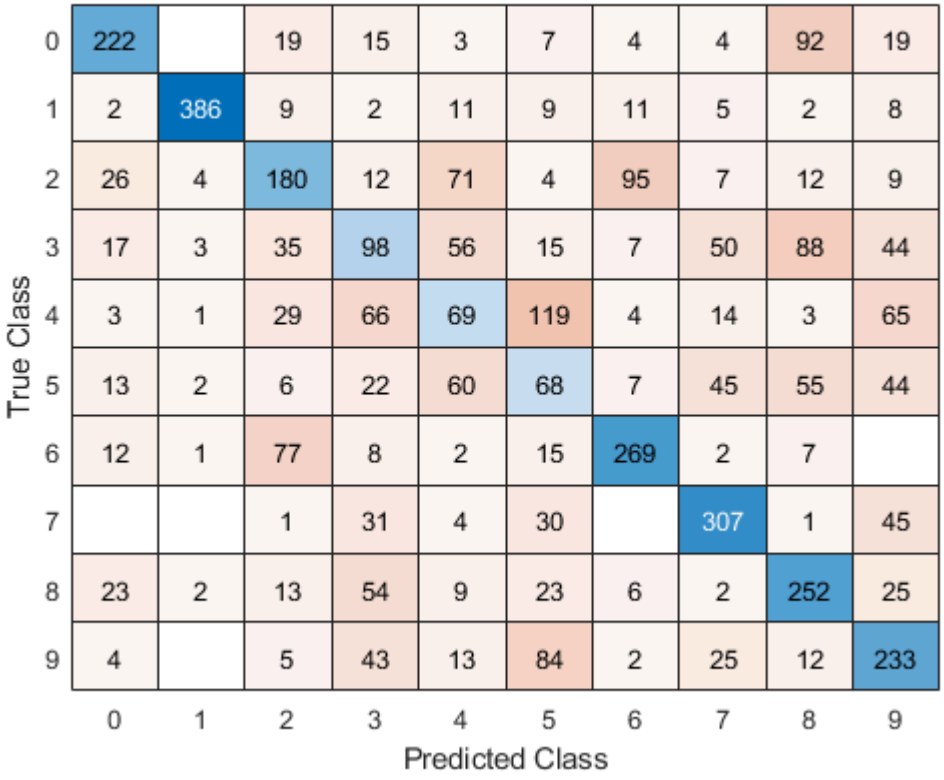
```
[testfeaturesNorm, testpcs, testscrs] = study_of_data(testfeatures, testlabels);
```



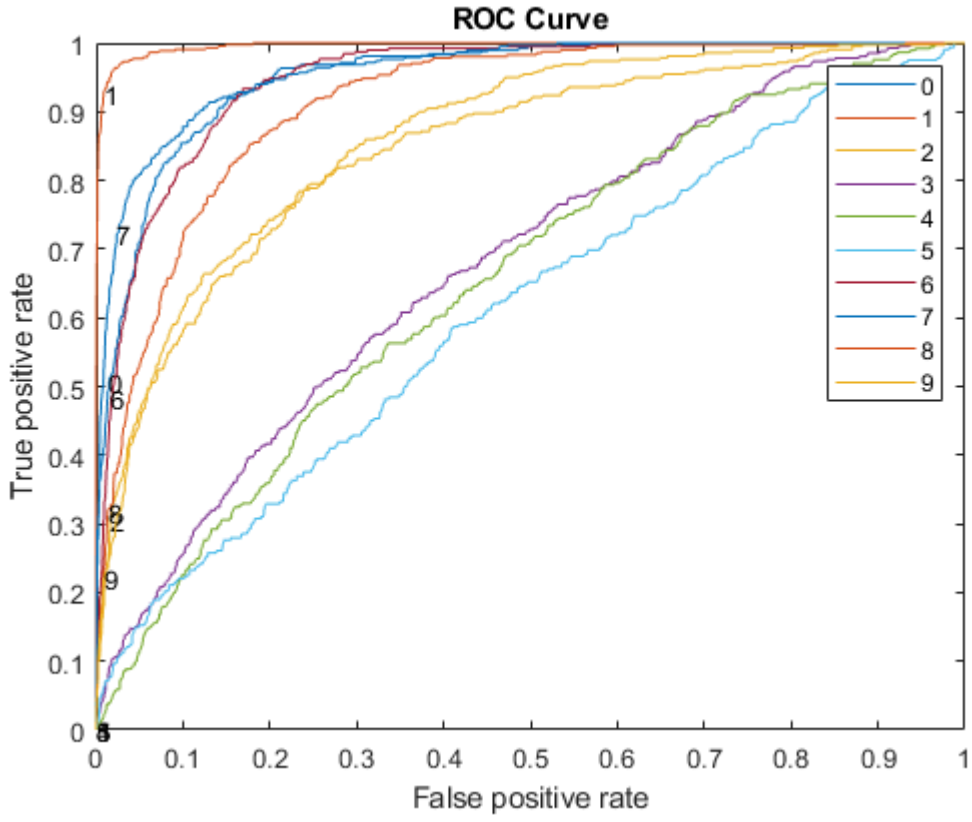





```
study_of_model(maxModel,testscrs,testlabels)
```




```
accuracy = 0.4790
```



Estudio Neural Network

Convolutional Network Practica 2

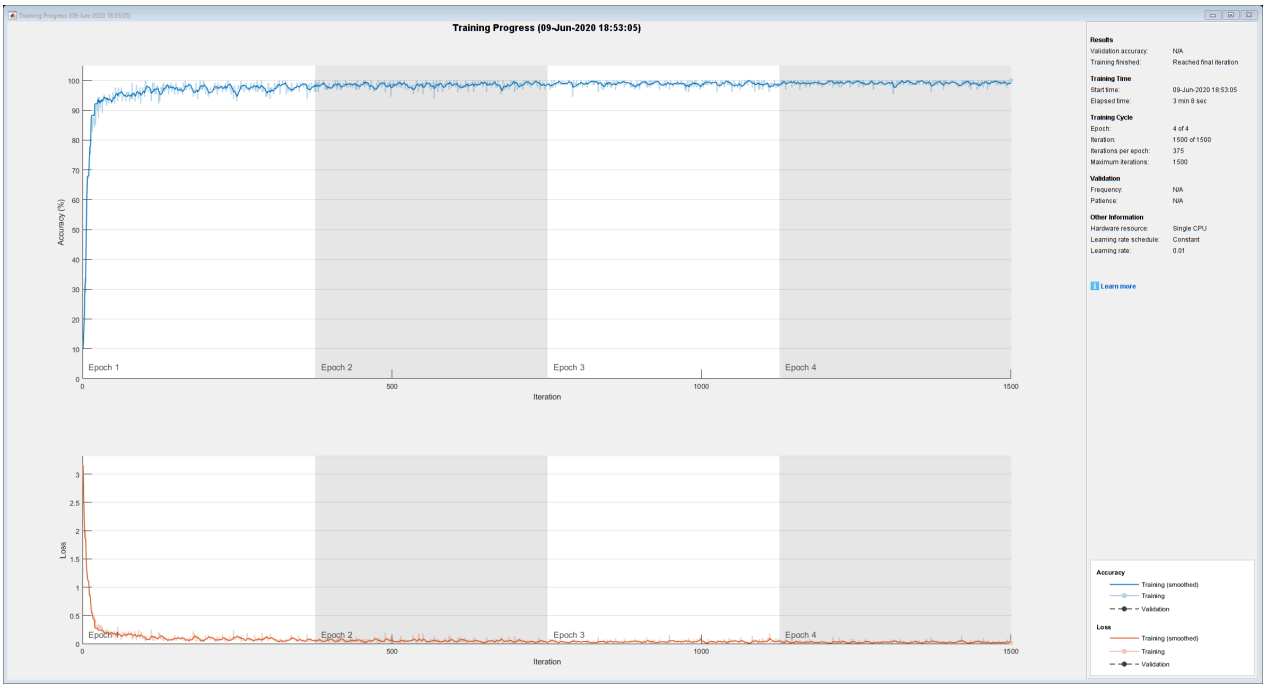
```
[images labels] = readMNIST('train-images.idx3-ubyte', 'train-labels.idx1-ubyte',60000 ,0);
[trainimages2 trainlabels2 images2 labels2]  = splitdata(images,labels, 0.8);
trainimages = zeros(20,20,1,length(trainimages2));
for i = 1:length(trainimages)
    trainimages(:,:,:,i) = trainimages2(:,:,i);
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);
```

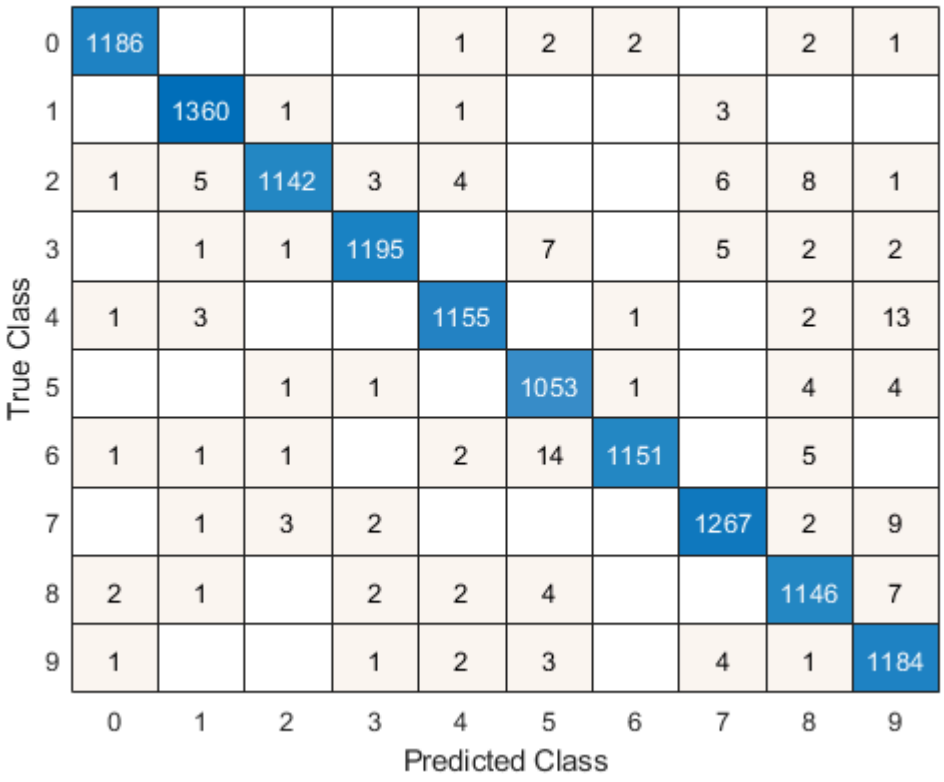


Classify Validation Images and Compute Accuracy

```
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
```