Introduction to Interactive Learning

**Michele Ferrari [*21118273*]**

**21/11/2021**

Table of Contents

[Introduction](#MW_H_DBF8846F)   
[1. Parameter Definition & Problem Setting](#MW_H_D1ECE321)   
[2. Human Agent Class Definition](#MW_H_D97B359B)   
[3. Computer-initiated Approach](#MW_H_C9FEDD1A)   
 [3.1 Algorithm Description](#MW_H_69068859)   
 [3.2 Algorithm Implementation](#MW_H_C8A84F70)   
 [3.3 Plot Results](#MW_H_79116EC2)   
[4. Human-initiated Approach](#MW_H_44F98BEB)   
 [4.1 Algorithm Description](#MW_H_E841DF44)   
 [4.2 Algorithm Implementation](#MW_H_9001C07B)   
 [4.3 Plot Results](#MW_H_70C4B58A)   
[5. Mixed-initiative Approach](#MW_H_A92BDCAE)   
 [5.1 Algorithm Description](#MW_H_5FCF5597)   
 [5.2 Algorithm Implementation](#MW_H_467B6E2F)   
 [5.3 Plot Results](#MW_H_B0F94E7D)   
[6. Practical Application with Unknown Threshold: Cats vs Dogs](#MW_H_117CC56D)   
 [6.1 Load Data](#MW_H_ED911F00)   
 [6.2 Plot Training Dataset](#MW_H_677F64B3)   
 [6.3 Tuning of the Feature Extractor](#MW_H_3867C2AC)   
 [6.4 Extract Features for Test Set](#MW_H_2D1CCA12)   
 [6.5 Perform Interactive Learning](#MW_H_E5C96909)   
[Conclusion](#MW_H_F517296F)

# 

# Introduction

Interactive Machine Learning is a branch of Machine Learning in which the user takes part in the training phase. He is not limited anymore to simply providing the machine with a set of training samples and labels, but he can also act as a labeling oracle for each iteration of the learning process.

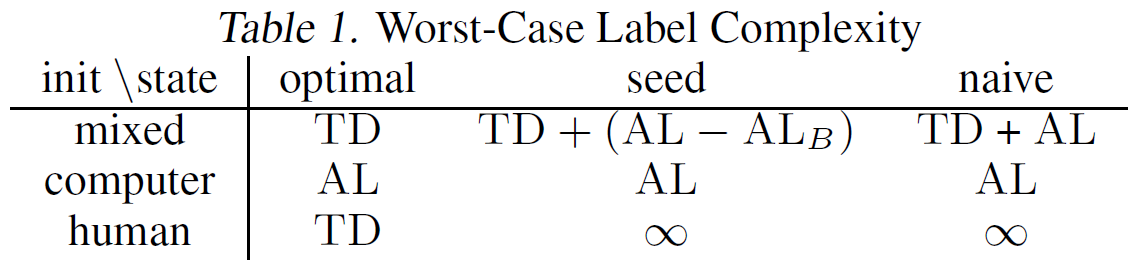
Here, three different approaches are implemented:

* ***Computer-Initiated***, where an active learning algorithm picks a query  and the human answers its label ,
* ***Human-Initiated***, where the human picks  and ,
* ***Mixed-Initiative***, where both parties can pick .

Moreover, different types of human oracles exist, ranging from very efficient to completely random ones (as described in [1]):

* **Optimal teacher**, that provides a optimal teaching set,
* **Seed teacher**, when the teacher is not optimal but can provide at least one item for each class,
* **Naive teacher**, when the teacher is not able to wisely choose labels and simply performs a random choice (which can also be detrimental).

In **Table 1**, it can be seen how different types of teachers can impact on the performance of the three different approaches. Referring to [1] for a detailed explanation, we simply remark that the C*omputer-Initiated* approach is not influenced by the type of human agent, whereas the other two approaches benefit in general from having a teacher that can provide labels more efficiently.



**Table 1:** Impact of the kind of teacher on the performance for every approach [1].

In this work, the different kind of teacher are simulated thanks to a class defined on purpose (Human\_agent), but the user is also allowed to manually input items and labels. Furthermore, the simulated teacher can be either infallible or fallible, meaning that the code can deal also with situations in which labelization errors occur.

Finally, the class Human\_agent is also dedigned to deal with probabilistic labels, however the three approaches have been implemented considering binary labels (), that here are indicated as either "Positive" or "Negative".

[1] *Jina Suh, Xiaojin Zhu, Saleema Amershi ; Proceedings of The 33rd International Conference on Machine Learning, PMLR 48 :2800-2809, 2016.*

# 1. Parameter Definition & Problem Setting

Algorithms are tested using a fixed threshold towards which they have to converge. In this case, the input space ranges from 0 to 1, the true threshold is 0.65 and the tolerance band is 0.02 wide. Classes are labeled as follows:

* "Positive" if ,
* "Negative" if .

clear; close all; clc

input\_space = [0 1];

tolerance = 0.01;

delta\_input = input\_space(2)/10;

true\_threshold = 0.65;

certainty\_mu = 0.8;

certainty\_sigma = 0.1;

# 2. Human Agent Class Definition

Select the type of human teacher among four different options:

* optimal teacher
* seed teacher
* naive teacher
* manual teacher (inputs are provided by an actual user typing  values and labels)

response\_flag = "Infallible";

optimality\_flag = "Seed";

**N.B.:**

* response\_flag has to be set to *Manual*, if optimality\_flag is set to *Manual*, and viceversa.
* optimality\_flag only makes sense in the *Human-Initiated* and *Mixed-Initiative* approaches

% Create the teacher

deterministic = true;

teacher = Human\_agent(response\_flag,optimality\_flag,deterministic,true\_threshold,certainty\_mu,certainty\_sigma);

# 3. Computer-initiated Approach

## 3.1 Algorithm Description

In the *Computer-initiated* approach, *Active Learning* (AL) is performed until convergence is achieved. In this case, convergence is attained when the absolute error between the true threshold and the computed one is smaller than a given tolerance.

AL consists in the choice of an item  for each iteration  based on previous pairs of items and labels  (step 1). The user is then presented with this value of , to which he has to assign a label  (step 2). Therefore the human acts as a *labeling oracle*.

In the code below, the function provide\_input\_computer performs step 1, whereas the method teacher\_response of the object teacher of type Human\_agent performs step 2, according to the kind of human agent (either fallible or infallible).

Step 1 can be implemented using different algorithms, in this case the *bisection method* has been applied.

The following algorithm allows also for a manual execution of both steps.

## 3.2 Algorithm Implementation

% Initialize variables to hold results

initialization

iter\_error = Inf;

while iter\_error > tolerance

x = provide\_input\_computer(inputs,labels,input\_space(1),input\_space(2));

if strcmp(response\_flag,'Manual')

message = sprintf("Value: %f", x);

uiwait(msgbox(message));

end

label\_prob = teacher\_response(teacher,x);

% Assign a label, given the label probability P(label == 'Positive')

labels = label\_prob\_2\_label(label\_prob,labels);

prev\_x = x;

iter = iter+1;

inputs = [inputs; x];

label\_probs = [label\_probs; label\_prob];

iters = [iters; iter];

% Check convergence

idx\_pos = contains(labels,"Positive");

idx\_neg = contains(labels,"Negative");

min\_pos = min(inputs(idx\_pos));

max\_neg = max(inputs(idx\_neg));

if ~isempty(find(idx\_pos)) && ~isempty(find(idx\_neg)) %#ok<EFIND>

thresh = (min\_pos+max\_neg)/2;

iter\_error = abs(thresh-true\_threshold);

end

thresholds = [thresholds; thresh];

iter\_errors = [iter\_errors; iter\_error];

% Check feasibility (not guaranteed if the teacher is fallible)

if min\_pos < max\_neg

disp("Failure! Values belonging to 'Negative' cannot be greater than values belonging to 'Positive'.")

failure = true;

break

end

end

% Create a structure to gather results, to be saved in the results folder

metrics.labels = labels;

metrics.iters = iters;

metrics.inputs = inputs;

metrics.label\_probabilities = label\_probs;

metrics.thresholds = thresholds;

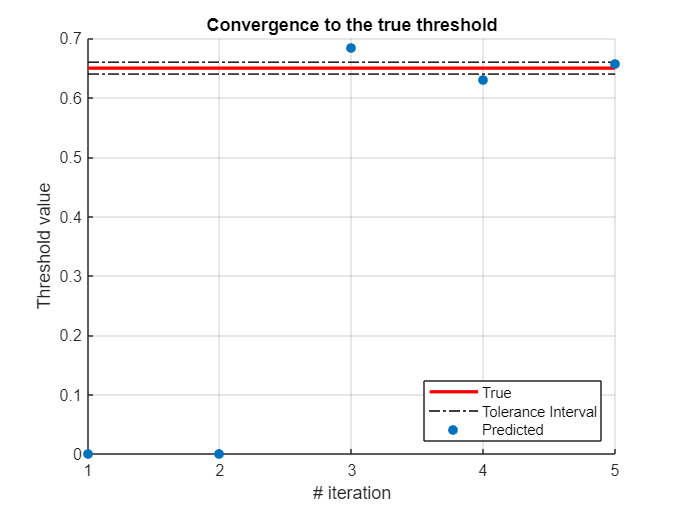
metrics.iter\_errors = iter\_errors;

metrics.failure = failure;

save(strcat('.\results\','ComputerInit\_',response\_flag,'\_',optimality\_flag,'.mat'),'metrics')

## 3.3 Plot Results

plot\_results(thresholds,true\_threshold,tolerance)

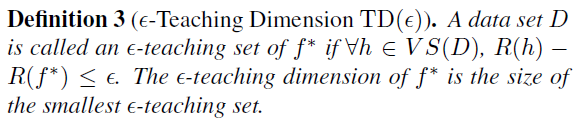


# 4. Human-initiated Approach

## 4.1 Algorithm Description

In the *Human-initiated* approach, the human can choose any item  for each iteration  (the choice depends on its degree of optimality). The human can decide how many iterations to teach, based on its level of confidence in the model (threshold) learned by the machine. Since in this simulation the threshold is known (not generally true in real applications), the learning process is stopped when convergence is attained.

To be more rigorous, the quantity TD can be defined as in **Definition 3** below:



Hence, the automatic algorithm will allow the user to input up to TD item, which are sufficient to guarantee the convergence of the method.

In the code below, the method teacher\_input provides the machine with the input according to the kind of human agent (optimal, seed, naive,...), whereas the method teacher\_response provides the machine with the label (exact or with a certain margin of error).

The following algorithm allows also for a manual execution of both steps.

## 4.2 Algorithm Implementation

In this case, it makes no sense distinguishing among different types of teachers. We can only simulate teachers that can or cannot miss the labelization, but we cannot specify their degree of optimality.

% Initialize variables to hold results

initialization

prev\_x = input\_space(1); % Arbitrary previous input

iter\_error = abs(thresh-true\_threshold);

while iter\_error > tolerance

[x,teacher] = teacher\_input(teacher,input\_space(1),input\_space(2),delta\_input,prev\_x);

label\_prob = teacher\_response(teacher,x);

% Assign a label, given the label probability P(label == 'Positive')

labels = label\_prob\_2\_label(label\_prob,labels);

prev\_x = x;

iter = iter+1;

inputs = [inputs; x];

label\_probs = [label\_probs; label\_prob];

iters = [iters; iter];

% Check convergence

idx\_pos = contains(labels,"Positive");

idx\_neg = contains(labels,"Negative");

min\_pos = min(inputs(idx\_pos));

max\_neg = max(inputs(idx\_neg));

if ~isempty(find(idx\_pos)) && ~isempty(find(idx\_neg)) %#ok<EFIND>

thresh = (min\_pos+max\_neg)/2;

iter\_error = abs(thresh-true\_threshold);

end

thresholds = [thresholds; thresh];

iter\_errors = [iter\_errors; iter\_error];

% Check feasibility (not guaranteed if the teacher is fallible)

if min\_pos < max\_neg

disp("Failure! Values belonging to 'Negative' cannot be greater than values belonging to 'Positive'.")

failure = true;

break

end

end

% Create a structure to gather results, to be saved in the results folder

metrics.labels = labels;

metrics.iters = iters;

metrics.inputs = inputs;

metrics.label\_probabilities = label\_probs;

metrics.thresholds = thresholds;

metrics.iter\_errors = iter\_errors;

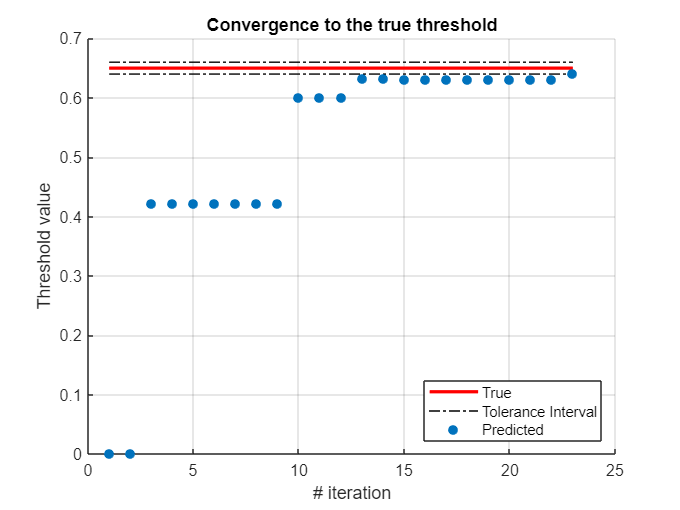
metrics.failure = failure;

save(strcat('.\results\','HumanInit\_',response\_flag,'\_',optimality\_flag,'.mat'),'metrics')

# 

## 4.3 Plot Results

plot\_results(thresholds,true\_threshold,tolerance)



# 5. Mixed-initiative Approach

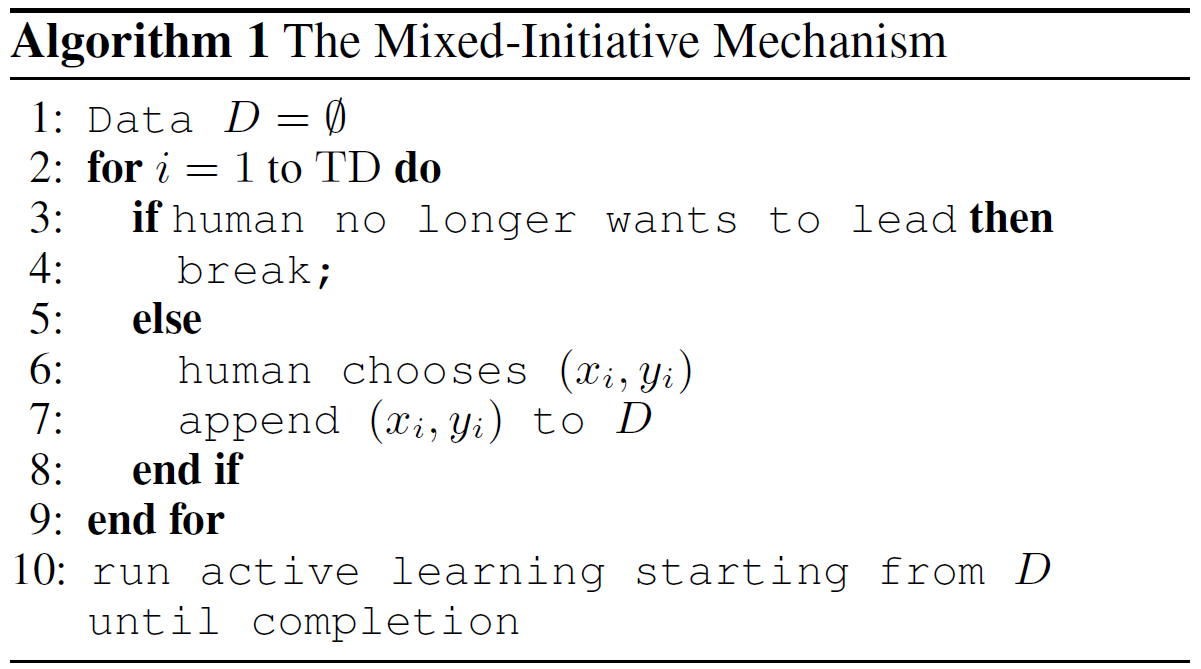
## 5.1 Algorithm Description

This approach is a combination of the previous ones, since it consists in two main phases:

1. In the beginning, the human can provide a certain number of labeled items ,
2. Then the machine starts AL exploiting the data the human agent provided in the first phase. In this second phase the human acts as labeling oracle.

The algorithm runs until convergence is attained.

The pseudo-code for this mechanism can be seen in Fig.1.



**Fig.1:** *Mixed-Initiative* approach, pseudocode taken from [1].

As a final remark, in the case of the optimal teacher, after the first phase the training samples allow to determine the correct threshold, without even running AL. That is because an optimal teacher provides exactly TD items (see Definition 3), which are already sufficent to guarantee the convergence of the algorithm.

## 5.2 Algorithm Implementation

% Initialize variables to hold results

initialization

prev\_x = input\_space(1); % Arbitrary previous input

iter\_error = abs(thresh-true\_threshold);

numIter\_human = 5; % Number of (input,label) pairs provided by the human teacher before starting active learning

while iter\_error > tolerance

if iter < numIter\_human

[x,teacher] = teacher\_input(teacher,input\_space(1),input\_space(2),delta\_input,prev\_x);

else

x = provide\_input\_computer(inputs,labels,input\_space(1),input\_space(2));

end

label\_prob = teacher\_response(teacher,x);

% Assign a label, given the label probability P(label == 'Positive')

labels = label\_prob\_2\_label(label\_prob,labels);

prev\_x = x;

iter = iter+1;

inputs = [inputs; x];

label\_probs = [label\_probs; label\_prob];

iters = [iters; iter];

% Check convergence

idx\_pos = contains(labels,"Positive");

idx\_neg = contains(labels,"Negative");

min\_pos = min(inputs(idx\_pos));

max\_neg = max(inputs(idx\_neg));

if ~isempty(find(idx\_pos)) && ~isempty(find(idx\_neg)) %#ok<EFIND>

thresh = (min\_pos+max\_neg)/2;

iter\_error = abs(thresh-true\_threshold);

end

thresholds = [thresholds; thresh];

iter\_errors = [iter\_errors; iter\_error];

% Check feasibility (not guaranteed if the teacher is fallible)

if min\_pos < max\_neg

disp("Failure! Values belonging to 'Negative' cannot be greater than values belonging to 'Positive'.")

failure = true;

break

end

end

% Create a structure to gather results, to be saved in the results folder

metrics.labels = labels;

metrics.iters = iters;

metrics.inputs = inputs;

metrics.label\_probabilities = label\_probs;

metrics.thresholds = thresholds;

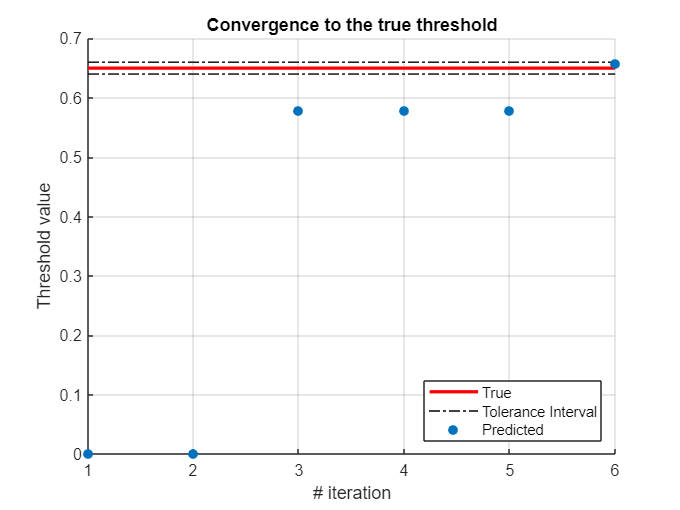
metrics.iter\_errors = iter\_errors;

metrics.failure = failure;

save(strcat('.\results\','MixedInit\_',response\_flag,'\_',optimality\_flag,'.mat'),'metrics')

## 5.3 Plot Results

plot\_results(thresholds,true\_threshold,tolerance)



# 6. Practical Application with Unknown Threshold: Cats vs Dogs

In order to apply this reasoning to a more practical example, the problem of classifying a given image as "Dog" or "Cat" is tackled. In this approach, the machine asks the user to label training samples and extracts from them features, based on which it can determine the boundary between the two classes. In this application, as in real-world, the threshold is not even known to the labeling oracle (who cannot know the value of each feature that exactly separates two classes).

The user can select to stop or continue the training phase anytime, depending on how satisfied he is with the classification accuracy (computed on a given test dataset).

This method is clearly *Human-Initiated.*

*Since we are only interested in developing a framework to make this algorithm work, only few training samples are provided to the user. In reality, a much larger training set is required, if we want to obtain a robust model that can be better generalized to unknown test samples.*

## 6.1 Load Data

load\_datasets

response\_flag = "Manual";

optimality\_flag = "Manual";

## 6.2 Plot Training Dataset

figure

set(gcf,'Visible','on')

tiledlayout(floor(sqrt(2\*nSamples\_train)),ceil(sqrt(2\*nSamples\_train)));

for i = 1:2\*nSamples\_train

nexttile

imshow(train{i})

end

## 6.3 Tuning of the Feature Extractor

HOG (Histogram of Oriented Gradients) features are extracted, after a **preprocessing phase** in which the image is masked.

% Feature Vector Example

processedImage = imbinarize(im2gray(train{1}));

% Extract HOG features and HOG visualization

[hog\_2x2, vis2x2] = extractHOGFeatures(processedImage,'CellSize',[2 2]);

[hog\_4x4, vis4x4] = extractHOGFeatures(processedImage,'CellSize',[4 4]);

[hog\_8x8, vis8x8] = extractHOGFeatures(processedImage,'CellSize',[8 8]);

% Show the original image

figure;

subplot(2,3,1:3); imshow(processedImage);

% Visualize the HOG features

subplot(2,3,4);

plot(vis2x2);

title({'CellSize = [2 2]'; ['Length = ' num2str(length(hog\_2x2))]});

subplot(2,3,5);

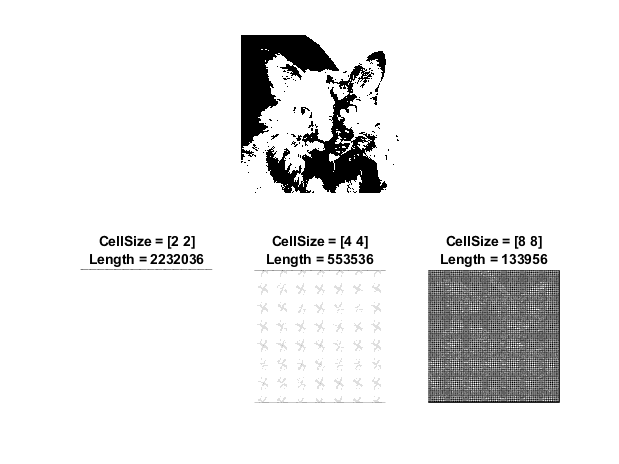
plot(vis4x4);

title({'CellSize = [4 4]'; ['Length = ' num2str(length(hog\_4x4))]});

subplot(2,3,6);

plot(vis8x8);

title({'CellSize = [8 8]'; ['Length = ' num2str(length(hog\_8x8))]});



## 6.4 Extract Features for Test Set

% Extract Test Features

features\_test = [];

for i = 1:2\*nSamples\_test

processedImage = imbinarize(im2gray(test{i}));

[new\_features,~] = extractHOGFeatures(processedImage,'CellSize',[4 4]);

features\_test = [features\_test; new\_features];

end

## 6.5 Perform Interactive Learning

After each iteration, a SVM (Support Vector Machine) model is trained, based on the current and all previous training samples. The hyperplane that separates the two classes is updated every iteration, so that the threshold between the two classes can be better and better identified.

inputs = {};

label\_probs = [];

labels = [];

features = [];

for i = 1:2\*nSamples\_train

figure

set(gcf,'Visible','on')

image(train{i})

axis equal

answer = questdlg('To which class does the current sample belong?', ...

'Sample Labelization', ...

'Cat','Dog','Cat');

close

switch answer

case 'Cat'

label\_prob = 1; % P(label == 'Cat')

case 'Dog'

label\_prob = 0;

end

processedImage = imbinarize(im2gray(train{i}));

[new\_features,hogVisualization] = extractHOGFeatures(processedImage,'CellSize',[4 4]);

inputs = [inputs; train{i}];

label\_probs = [label\_probs; label\_prob];

labels = [labels; string(answer)];

features = [features; new\_features];

idx\_pos = contains(labels,"Cat");

idx\_neg = contains(labels,"Dog");

if ~isempty(find(idx\_pos)) && ~isempty(find(idx\_neg)) %#ok<EFIND>

classifier = fitcecoc(features, labels);

label\_predicted = predict(classifier, features\_test);

figure

set(gcf,'Visible','on')

confMat = confusionchart(label\_test,label\_predicted,'RowSummary','row-normalized','ColumnSummary','column-normalized');

title('Test Confusion Matrix')

pause(2)

close

if i ~= 2\*nSamples\_train

answer = questdlg('Continue Training?', ...

'Continue', ...

'Yes','No','Yes');

close

switch answer

case 'Yes'

continue

case 'No'

break

end

end

end

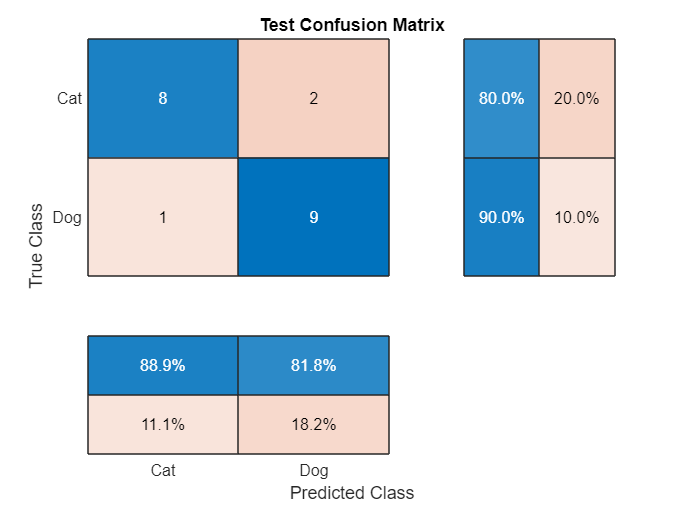
end

Final Confusion Matrix:

figure

confMat = confusionchart(label\_test,label\_predicted,'RowSummary','row-normalized','ColumnSummary','column-normalized');

title('Test Confusion Matrix')



# Conclusion

Running the code multiple times, according to the possible different combinations of the parameters defining the human teacher:

* (Optimal, Infallible) teacher,
* (Optimal, Fallible) teacher,
* (Seed, Infallible) teacher,
* (Seed, Fallible) teacher,
* (Naive, Infallible) teacher,
* (Naive, Fallible) teacher,
* Manual teacher,

we can see the difference that the optimality of the teacher can make on the performance of the algorithm.

Moreover, if the teacher can provide wrong labels, the algorithms are not even guaranteed to converge, leading to a failure of the teaching process.