moons.csv

Table of Contents

1. Dataset preparation

```
% check the dataset is present:
if ~exist('moons.csv', 'file')
    error('You need to download moons.csv from Moodle and place it into the
same directory as this live script');
end
% loading and checking data to use and store accordingly:

% re-seeding the random number generator to '0' for reproducible results
rng(0);

%read data from 'moons.csv'
data = readcell('moons.csv');

% looking at the first 10 rows of data. Identifying the target feature
values - first 2 coloumns contain feature values, 3rd column contains lables
data(1:1:10, 1:1:end);

 % read out the observations and store into variable called 'obs' (while
dropping the feature names): 150 rows, 3 columns of data (leaving out the
data titles e.g. 'Feature 1' etc...)
obs = data(2:end,:)
```

obs = 150x3 cell

	1	2	3
1	0.1567	1.3182	'Class A'
2	-0.9705	0.6211	'Class A'
3	-0.9086	0.1598	'Class A'
4	0.7076	0.2604	'Class A'
5	-1.2325	0.9279	'Class A'
6	-0.6282	0.8368	'Class A'
7	0.6124	0.2656	'Class A'
8	-0.7009	-0.0706	'Class A'
9	1.2655	1.3637	'Class A'

	1	2	3
10	0.4948	0.9364	'Class A'
11	-0.5249	1.0001	'Class A'
12	-0.8581	0.2206	'Class A'
13	-1.3395	-0.3469	'Class A'
14	0.9408	0.1181	'Class A'

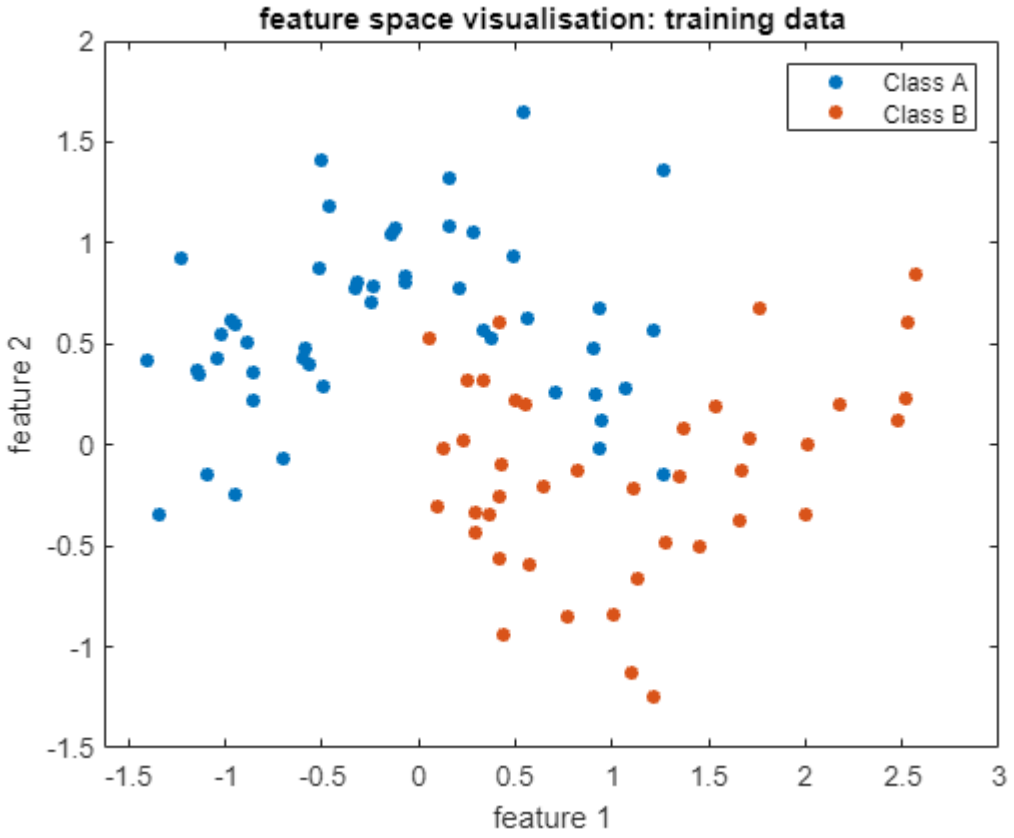
⋮

```
% shuffle the data to remove any ordering preasent and storing this data
into 'obs_shuffled'
obs_shuffled = obs(randperm(size(obs,1)), :);

% taking 40% of the data to use as testing data. 40% of 150 = 60. Storing
this into variable 'dTest'
nTest = round(0.4 * size(obs_shuffled,1));


% horizontal split:
% copying out our testing data (40%) shuffled into 'obs_test'
obs_test = obs_shuffled(1:1:nTest, :);

% copying out our shuffled training observation data (final 60% of the data)
obs_train = obs_shuffled(nTest+1:1:end, :);


% set the column index of the target feature values
label_index = 3;


% vertical split 1: 40% data
% separating the examples and the labels for the testing observations data:
test_labels = categorical(obs_test(:, label_index));
test_examples = cell2mat(obs_test(:, 1:end~=label_index));

% have a quick look at the sizes, as a sanity check
size(test_examples);
size(test_labels);


% vertical split 2: 60% data
% separate the examples and the labels for the training observations:
train_labels = categorical(obs_train(:, label_index));
train_examples = cell2mat(obs_train(:, 1:end~=label_index));

% have a quick look at the sizes, as a sanity check
size(train_examples);
```

```
size(train_labels);
```

2. Feature space visualisation

```
rng(0); % re-seed the random number for reproducible results

% Creating a visualisation of feature space showing the training data

figure; % creating figure window to draw gscatter
gscatter(train_examples(:,1), train_examples(:,2), train_labels);  %
creating gscatter graph using training data
xlabel('feature 1');  % labling the x-axis
ylabel('feature 2');  % labling the y-axis
title('feature space visualisation: training data'); % titling the feature
space
legend('Location', 'northeast'); % altering the location of the 'Class A,
Class B' Labels
```

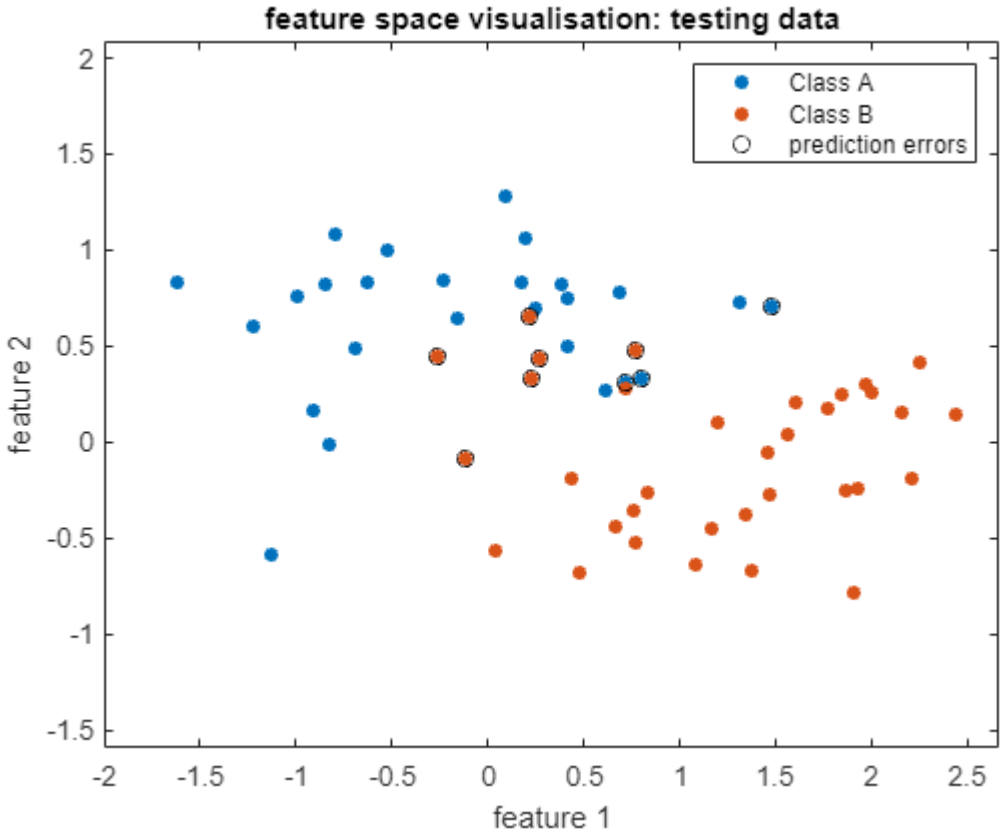



```
% Training naive bayes classifier with training data:
m_nb = fitcnb(train_examples, train_labels);  % training data using matlabs
built in naive bayes classifier
predictions = m_nb.predict(test_examples); % using the navie bayes
classifier to make predications based on our training examples
```

```
p_nb = sum(predictions == test_labels) / length(test_labels) % calculating a
measure of performance percentage, which is: 85% and storing it in 'p_nb'
```

```
p_nb = 0.8500
```

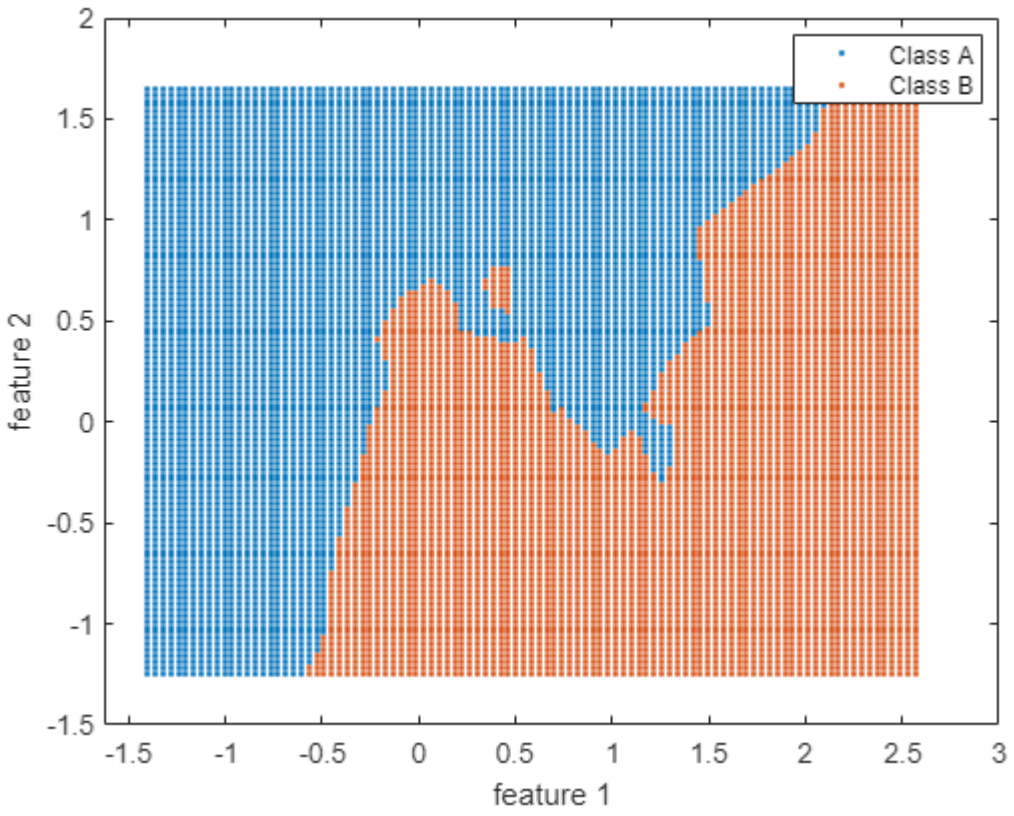
```


% Creating a second visulisation of feature space showing the testing data
figure; % creating figure window to draw gscatter
gscatter(test_examples(:,1), test_examples(:,2), test_labels);  % creating
gscatter graph using testing data
xlabel('feature 1');  % labling the x-axis
ylabel('feature 2');  % labling the y-axis
title('feature space visualisation: testing data'); % titling the feature
space
legend('Location', 'northeast'); % altering the location of the 'Class A,
Class B' Labels


% Adding black circles around testing examples that are missclassified by
% naive bayes:
mc = predictions ~= test_labels; % storing the missclassified data into
variable 'mc'
% set the scales of the two axes equal to allow easy visual estimation of
distances:
axis('equal');
% allow overlaying of plots:
hold('on');
scatter(test_examples(mc,1), test_examples(mc,2), 'ok', 'DisplayName',
'prediction errors'); % ploting the missclassified as a black circle:
```



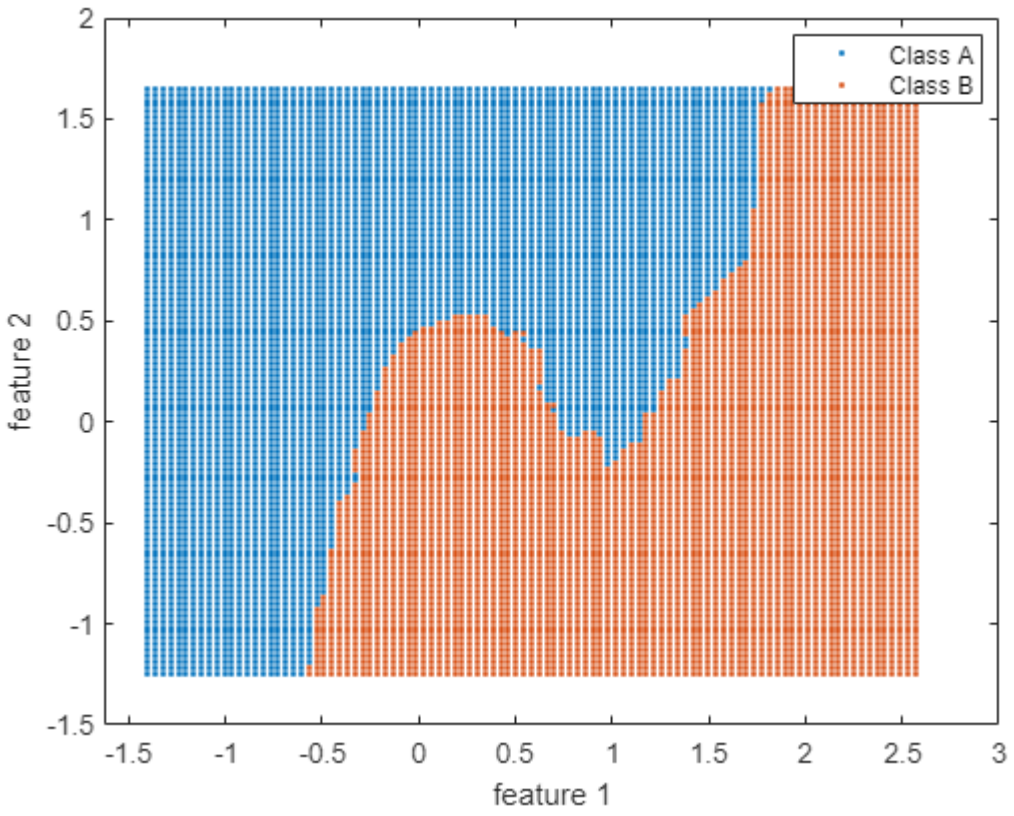

```
wp = find(predictions ~= test_labels); % founded misclassified
predications// checking the number of prediction errors matches the number
of black circles on the graph (for reassurance)
```

3. Abstraction visualisations

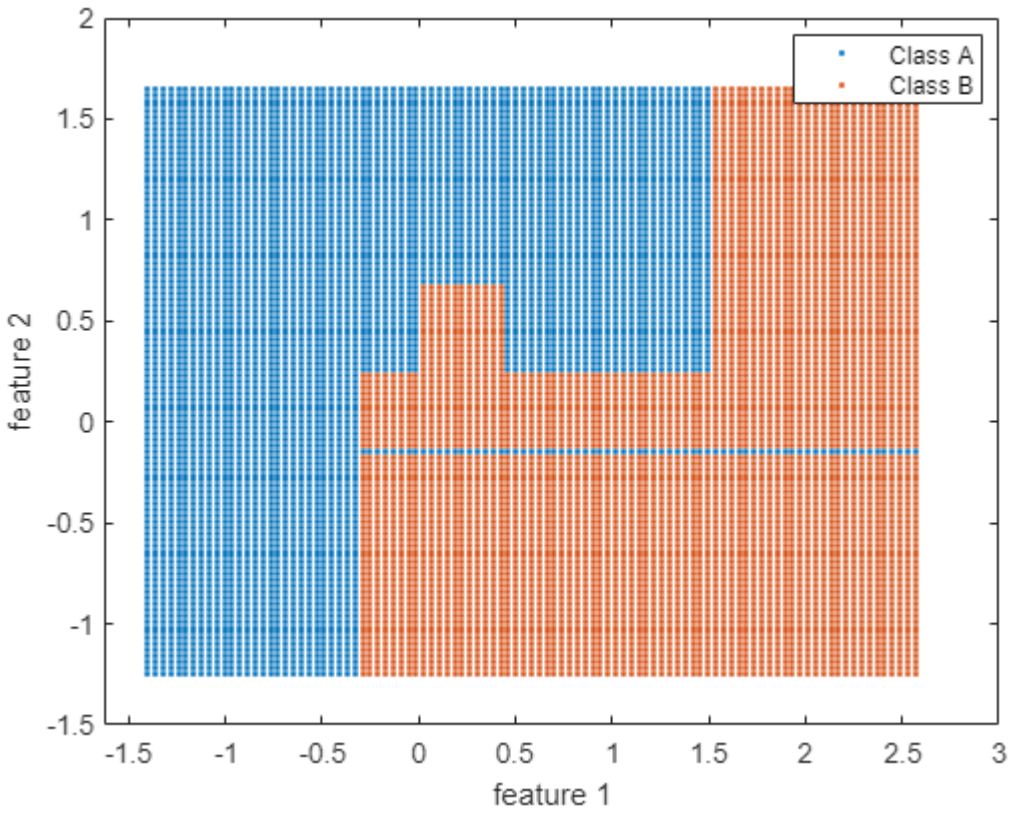
```
rng(0); % re-seed the random number for reproducible results


% KNN ABSTRACTION
m_knn1 = fitcknn(train_examples, train_labels, 'NumNeighbors', 1); % train
data with K-NN classifier
visual_abstraction(m_knn1)
xlabel('feature 1');  % labling the x-axis
ylabel('feature 2');  % labling the y-axis
```



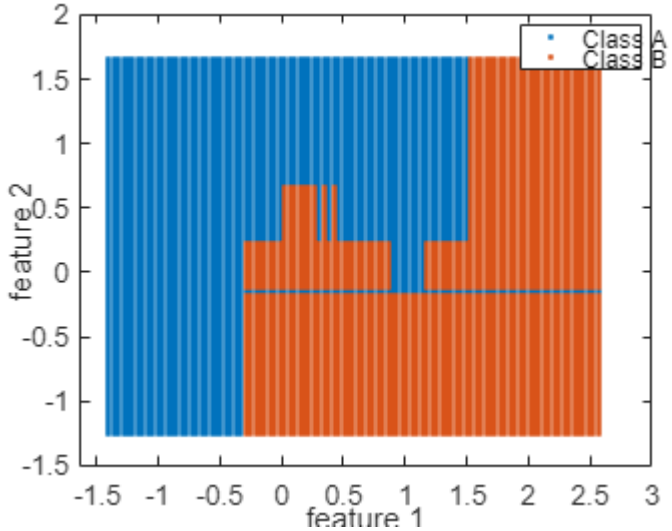

```
m_knn2 = fitcknn(train_examples, train_labels, 'NumNeighbors', 5); % train
data with K-NN classifier
visual_abstraction(m_knn2)
xlabel('feature 1');  % labling the x-axis
ylabel('feature 2');  % labling the y-axis
```




```
% DT ABSTRACTION
m_dt1 = fitctree(train_examples, train_labels, 'MinParentSize', 10,
'MergeLeaves', 'off', 'Prune', 'off'); % train data with decision tree
classifier
visual_abstraction(m_dt1)
xlabel('feature 1');  % labling the x-axis
ylabel('feature 2');  % labling the y-axis
```

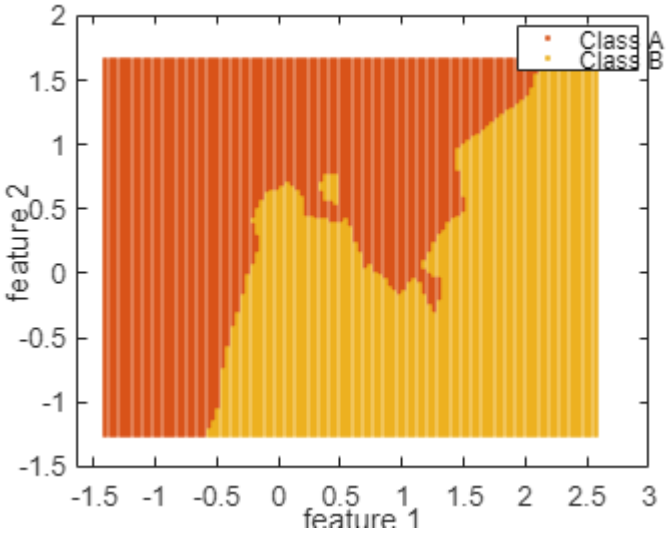



```
m_dt2 = fitctree(train_examples, train_labels, 'MinParentSize', 1,
'MergeLeaves', 'off', 'Prune', 'off'); % train data with decision tree
classifier
visual_abstraction(m_dt2)
xlabel('feature 1');  % labling the x-axis
ylabel('feature 2');  % labling the y-axis
```

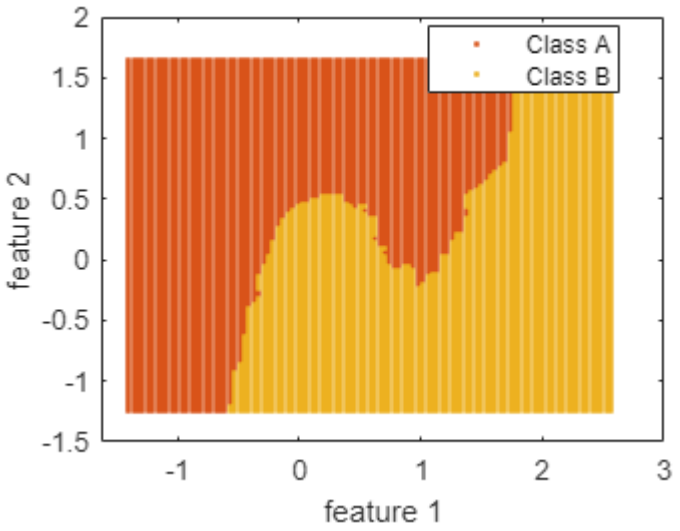


4. Classifier re-implementations

```
rng(0); % re-seed the random number for reproducible results

% add your code on the lines below:


%Reimplimentation of KNN, Displaying visual abtraction using the
%'visual_abstraction function (Needs to be independant of meshgrid format
m_knn1 = my_fitcknn(train_examples, train_labels, 'NumNeighbors', 1);
visual_abstraction(m_knn1)
xlabel('feature 1');  % labling the x-axis
ylabel('feature 2');  % labling the y-axis
```




```
m_knn2 = my_fitcknn(train_examples, train_labels, 'NumNeighbors', 5);
visual_abstraction(m_knn2)
xlabel('feature 1');  % labling the x-axis
ylabel('feature 2');  % labling the y-axis
```


Class A
Class B
feature 2
feature 1
2
1.5
1
0.5
0
-0.5
-1
-1.5
-1
0
1
2
3