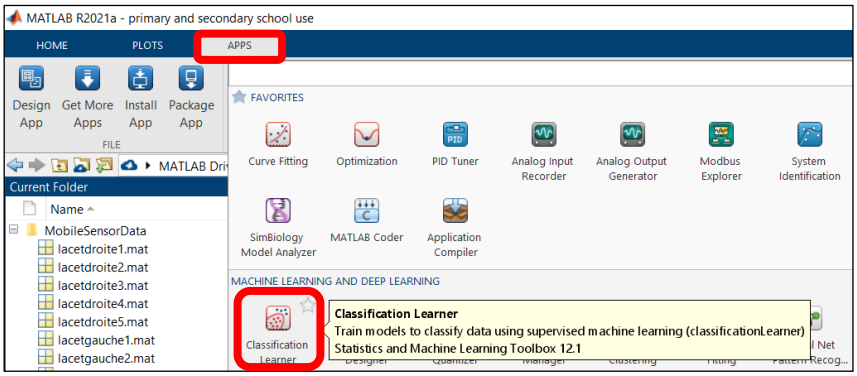
Logiciel | Matlab Classification Learner

L'application *Classification Learner* est un outil interactif présent dans le logiciel *Matlab* permettant d'effectuer des tâches d'apprentissage comme l'exploration de données, l'extraction de caractéristiques, l'entraînement et la validation de modèles ou l'évaluation de résultats.
L'un des principaux avantages de l'application par rapport à l'écriture de lignes de code est que celle-ci permet de comparer facilement et rapidement plusieurs modèles d'entraînement.
En plus, cette application génère automatiquement les lignes de code *Matlab* correspondant au modèle sélectionné. Une fois le code généré, il peut être modifié et intégré dans une application existante.

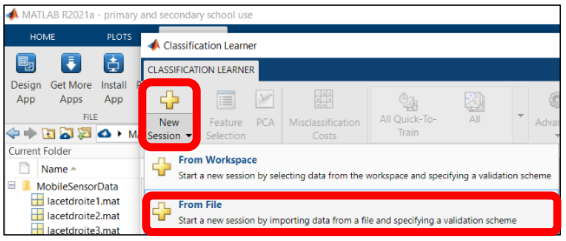
1 - Chargement de l'application :

Lancement du logiciel *Matlab* → *Apps* → *Classification Learner*

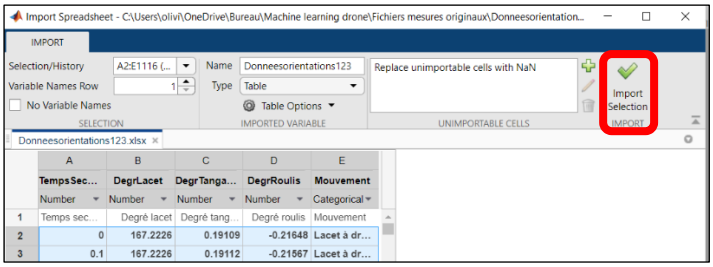


2 – Importation des données d'apprentissage :

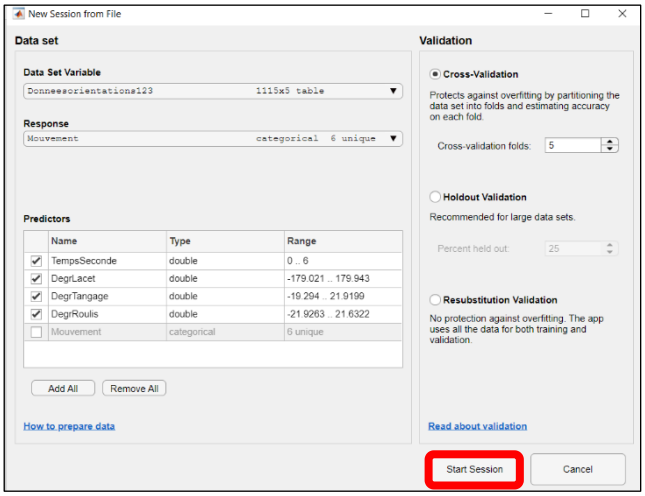
New Session → *From File* → charger le fichier *Donneesorientations123.xlsx*



Import Selection



Start Session



3 – Choix d'un modèle d'apprentissage :

L'information d'appartenance à un comportement du drone est exprimée par des nuages de points de couleurs différentes. Ce graphique permet d'identifier des critères de séparations de variables. Sur la figure 1a, le comportement de tangage arrière ou de tangage avant est clairement séparé des autres. En revanche, les comportements de roulis et de lacet sont moins bien séparés. Il y a des zones de recouvrement entre certains comportements.
Pour entraîner un modèle d'apprentissage, il suffit d'aller dans la galerie des modèles de classification (*Model Type*), d'en choisir un et de cliquer sur *Train*.

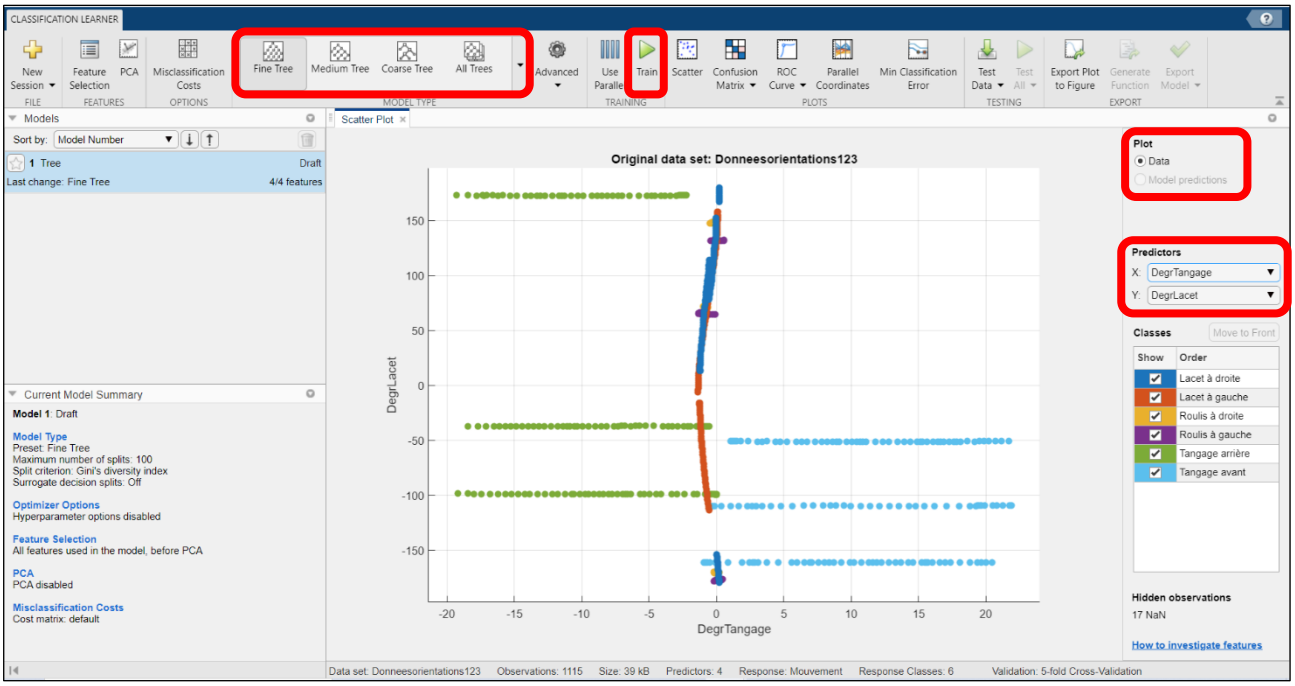


Figure 1a – Répartition des données : Degré de lacet = f (Degré tangage)

Une fois l'entraînement terminé sur des données en provenance des capteurs, le modèle apparait à gauche, dans l'historique, avec l'indication de son niveau de précision (voir figure 1b). Plus ce pourcentage est élevé, plus le modèle sera précis dans la classification de nouvelles données. Pour faciliter la comparaison de plusieurs modèles, le plus performant est surligné.

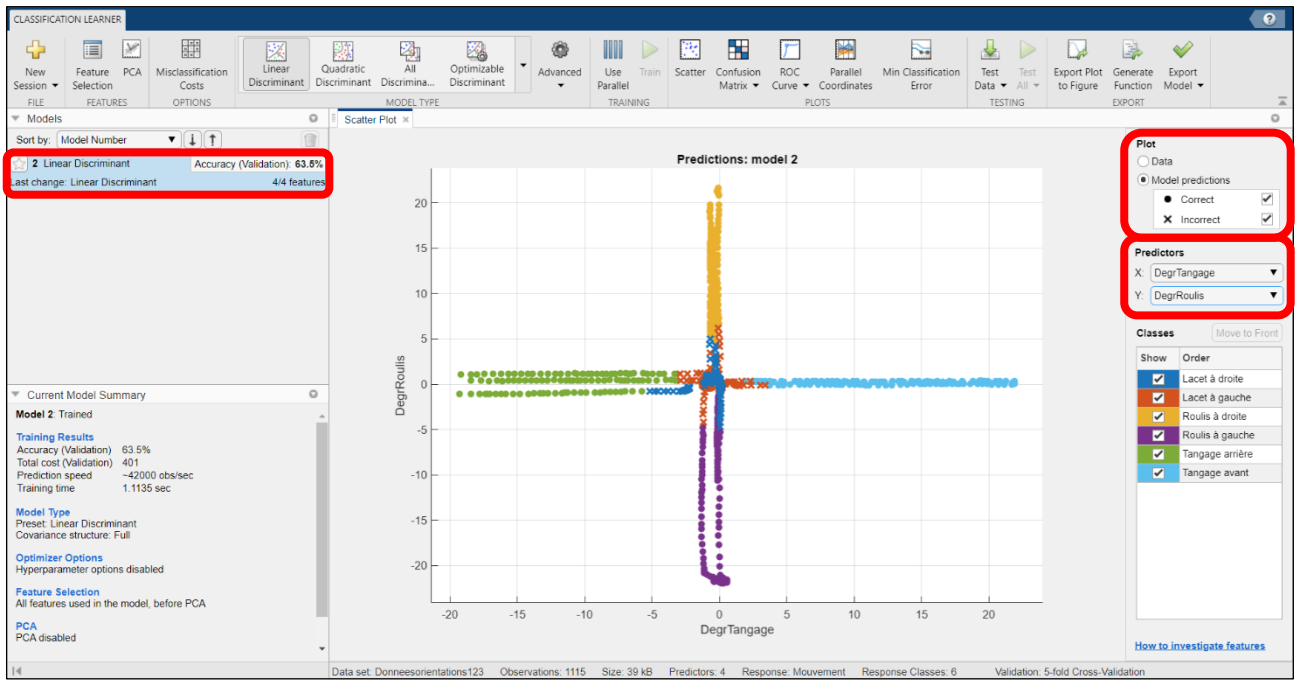


Figure 1b – Modèle de prédiction *Discriminant* : Degré de roulis = f (Degré tangage)

Dans le domaine de la classification, il n'existe pas un modèle qui soit plus performant que tous les autres. En fonction du problème traité, l'un ou l'autre de ces modèles sera plus adapté : arbres de décision, machines à vecteur de support, méthode des plus proches voisins, méthodes ensemblistes ou réseaux de neurones…. Des paramètres définis par défaut devraient convenir à de nombreux problèmes.

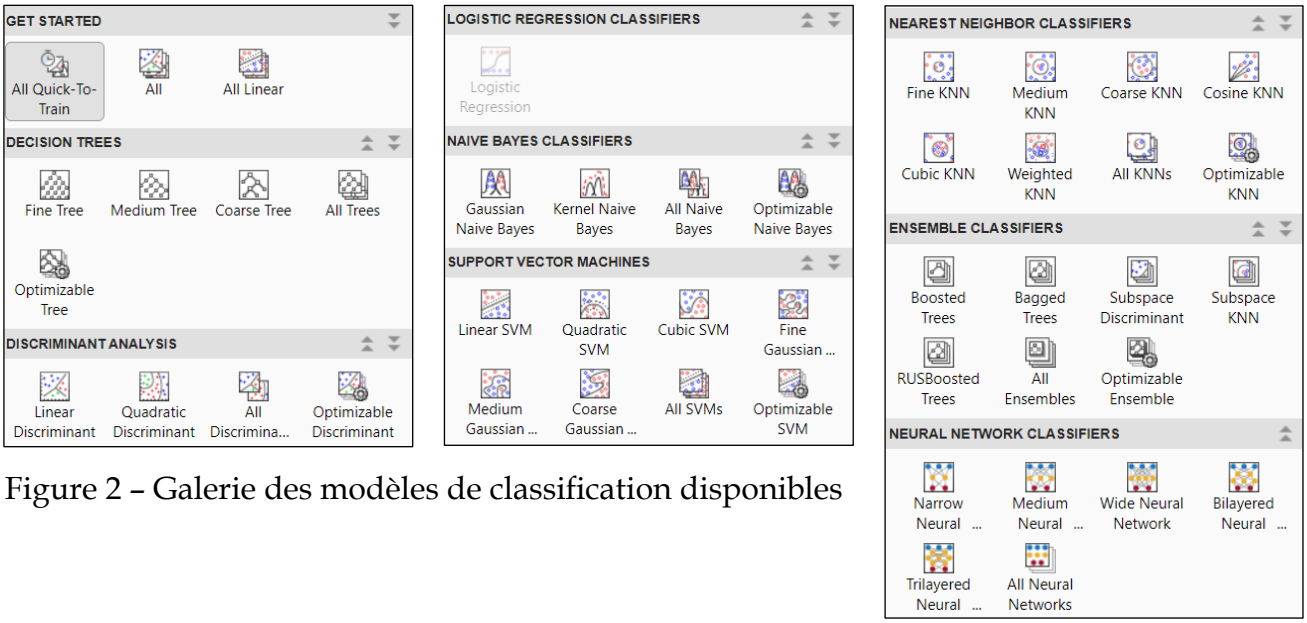
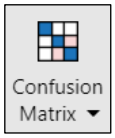


Figure 2 – Galerie des modèles de classification disponibles

4 – Matrice de confusion :

Les matrices de confusion sont des outils très utiles pour déterminer rapidement si un modèle de classification de données est efficace. Les données qui sont placées sur la diagonale sont correctement classées. Toutes celles qui sont en dehors de la diagonale ont été mal classées par le modèle. Un modèle parfait aurait un score de 100% sur la diagonale et un score de 0% partout ailleurs.



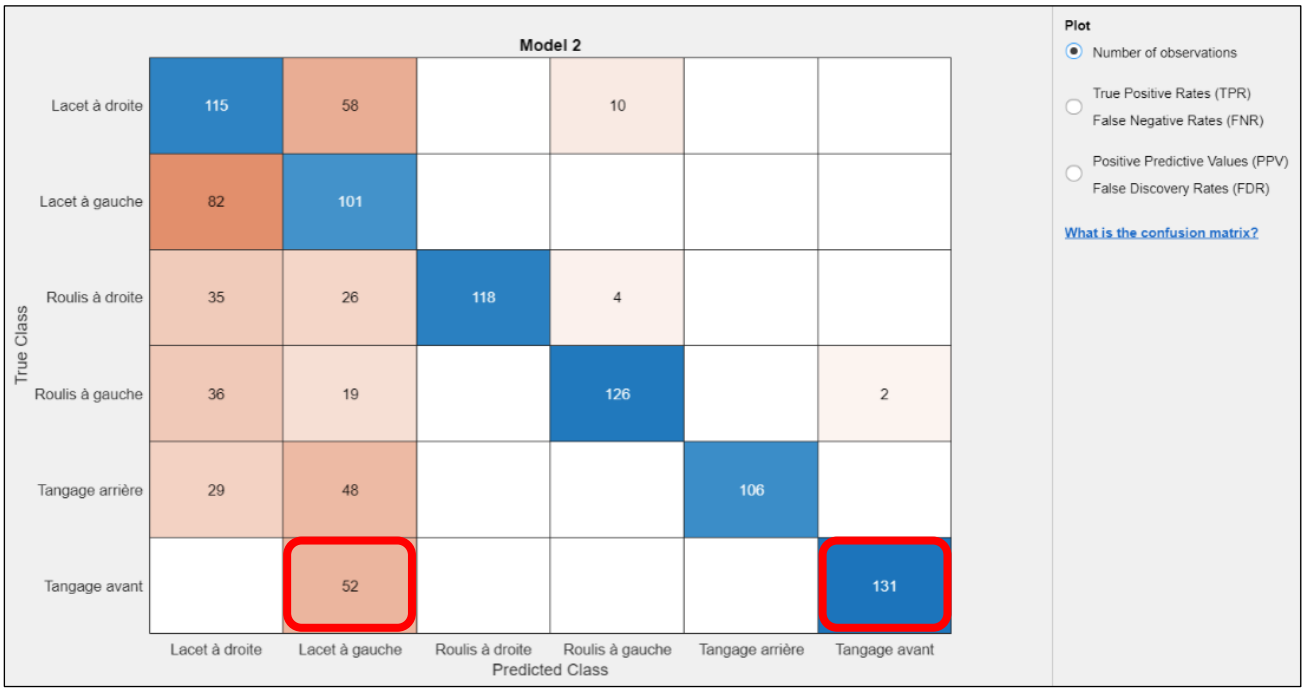


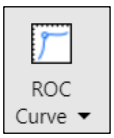
Figure 3 – Matrice de confusion du modèle *Discriminant*

Pour le comportement *Tangage avant*, pour 131 cas (72 %) le modèle a correctement prédit l'activité, en revanche dans 52 cas (28 %), c'est-à-dire dans tous les autres cas, le modèle a prédit le comportement *Lacet à gauche* au lieu du comportement *Tangage avant*. Le modèle a donc confondu ces deux comportements.

Il est possible de passer de modèles en modèles afin de comparer les matrices de confusion.

5 – Courbe ROC :

Un autre type de diagnostic proposé par l'application *Classification Learner* est la courbe *ROC*. Elle utilisée pour représenter la sensibilité des classificateurs binaires. L'allure de la courbe illustre le compromis réalisé entre sensibilité et spécificité. Elle représente le nombre de vrais positifs en fonction du nombre de faux positifs



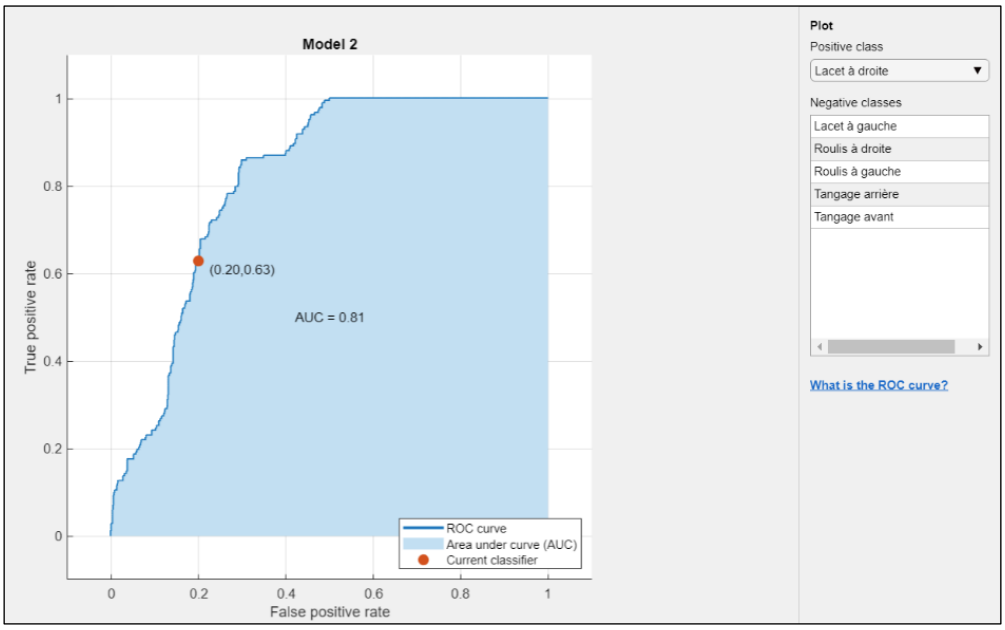


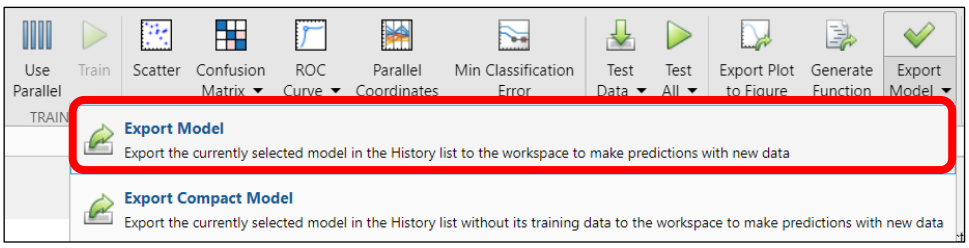
Figure 4 – Courbe *ROC* (*Receiver Operating Characteristic*)

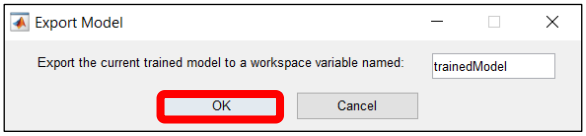
Le marqueur (point rouge) sur le tracé montre les performances du classificateur actuellement sélectionné. Le marqueur affiche les valeurs du taux de faux positifs (*FPR*) et du taux de vrais positifs (*TPR*). Par exemple, un taux de faux positifs (*FPR*) de 0,2 indique que le classificateur actuel attribue de manière incorrecte 20 % des observations à la classe positive. Un vrai taux positif de 0,63 indique que le classificateur actuel attribue correctement 63 % des observations à la classe positive.

Un résultat parfait sans point mal classé est un angle droit en haut à gauche du tracé. La surface sous la courbe (*AUC*) est une mesure de la qualité globale du classificateur. Des valeurs de zone sous la courbe plus grandes indiquent de meilleures performances du classificateur.

6 – Exportation du modèle :

Après avoir sélectionné le modèle d'apprentissage le plus performant de la liste, il faut l'exporter pour continuer à l'exploiter sous *Matlab*.



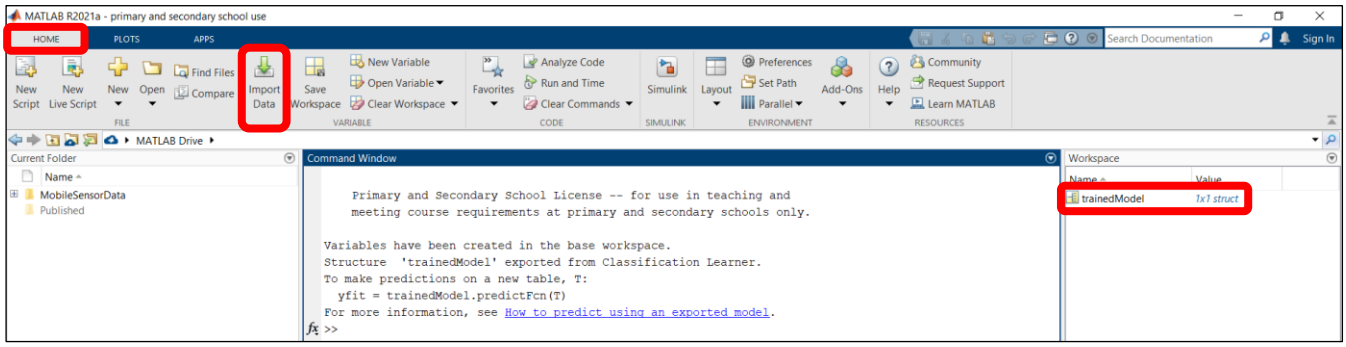


Le modèle exporté apparait dans le *Workspace* avec la désignation *trainedModel*.

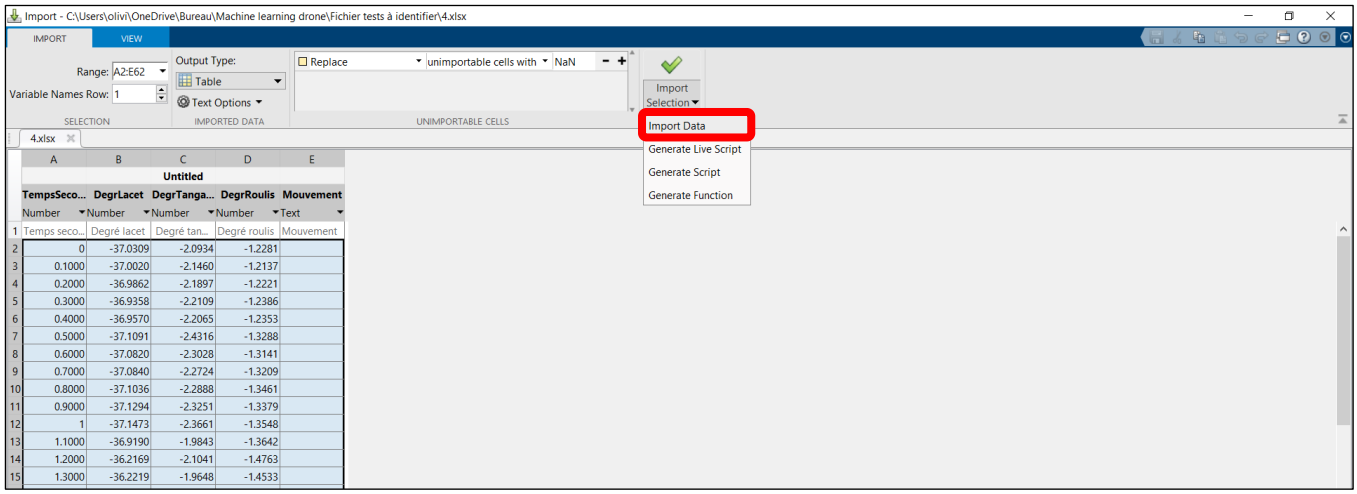
7 – Test du modèle d'apprentissage :

Le modèle d'entraînement *trainedModel* est utilisé pour classer des nouvelles données de test et visualiser les résultats obtenus.

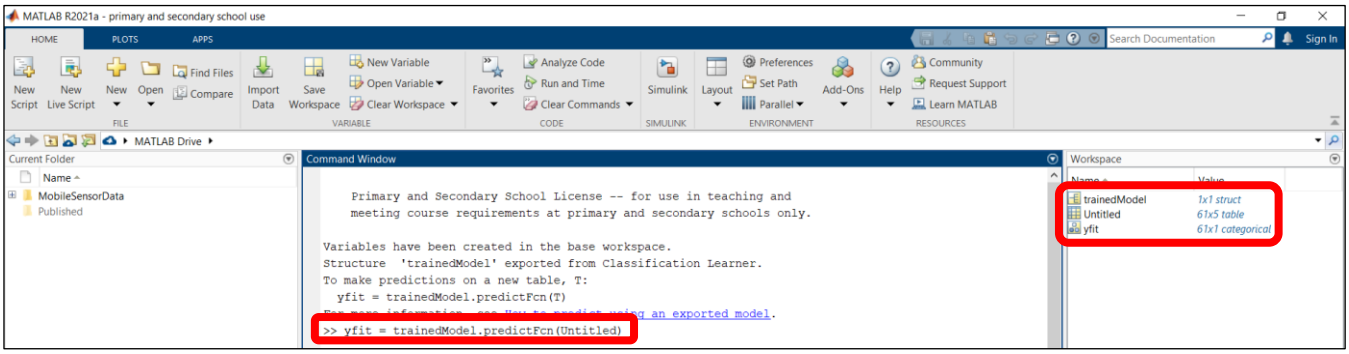
Importation des nouvelles données → *Import Data* → charger le fichier *4.xlsx*



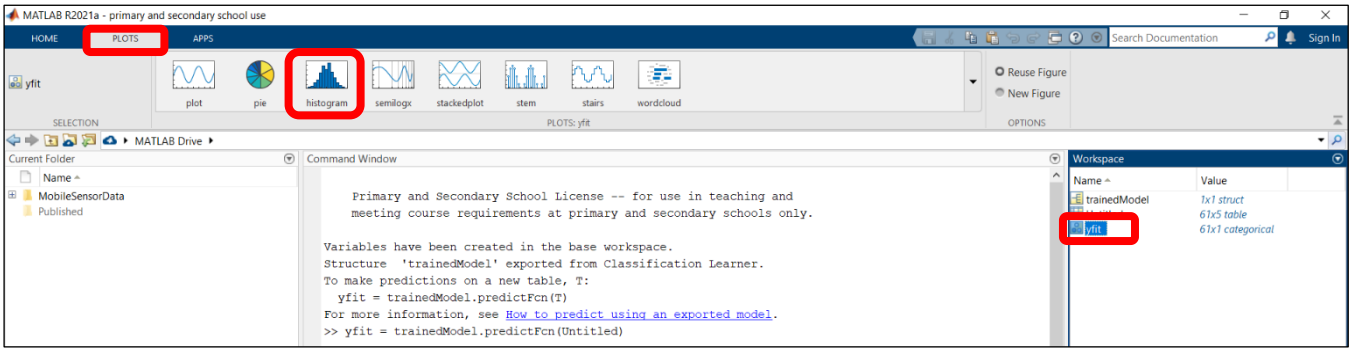
Import Selection → *Import Data*



Le fichier de données de test importé apparait avec la désignation *Untitled*. Le modèle d'apprentissage sélectionné précédemment peut être testé avec la ligne de commande *yfit*.



La prédiction du modèle entrainé est visualisable sous forme de graphique.



Le modèle d'apprentissage indique que le fichier de données test est plutôt relatif à un comportement de roulis à gauche. En comparant cette prédiction à la représentation graphique des données de test, la réponse est exacte. Cependant, le résultat du modèle sélectionné est perfectible (rappel : niveau de précision du modèle utilisé = 63,5 %).

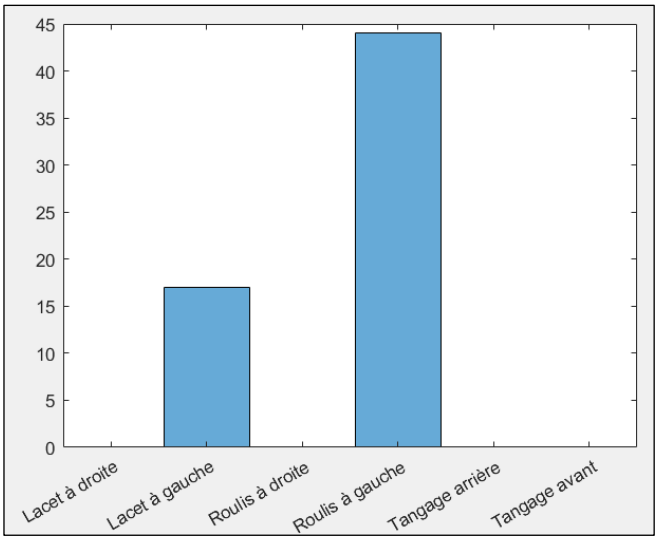


Figure 5 – Prédiction du modèle d'apprentissage à partir du fichier de données test 4

Pour le tracé des courbes relatives à l'évolution des angles du drone (mesures réelles) :
- cliquer sur le fichier de données de test importé (*Untitled*) dans le Workspace,
- cliquer sur l'onglet Stackeplot.

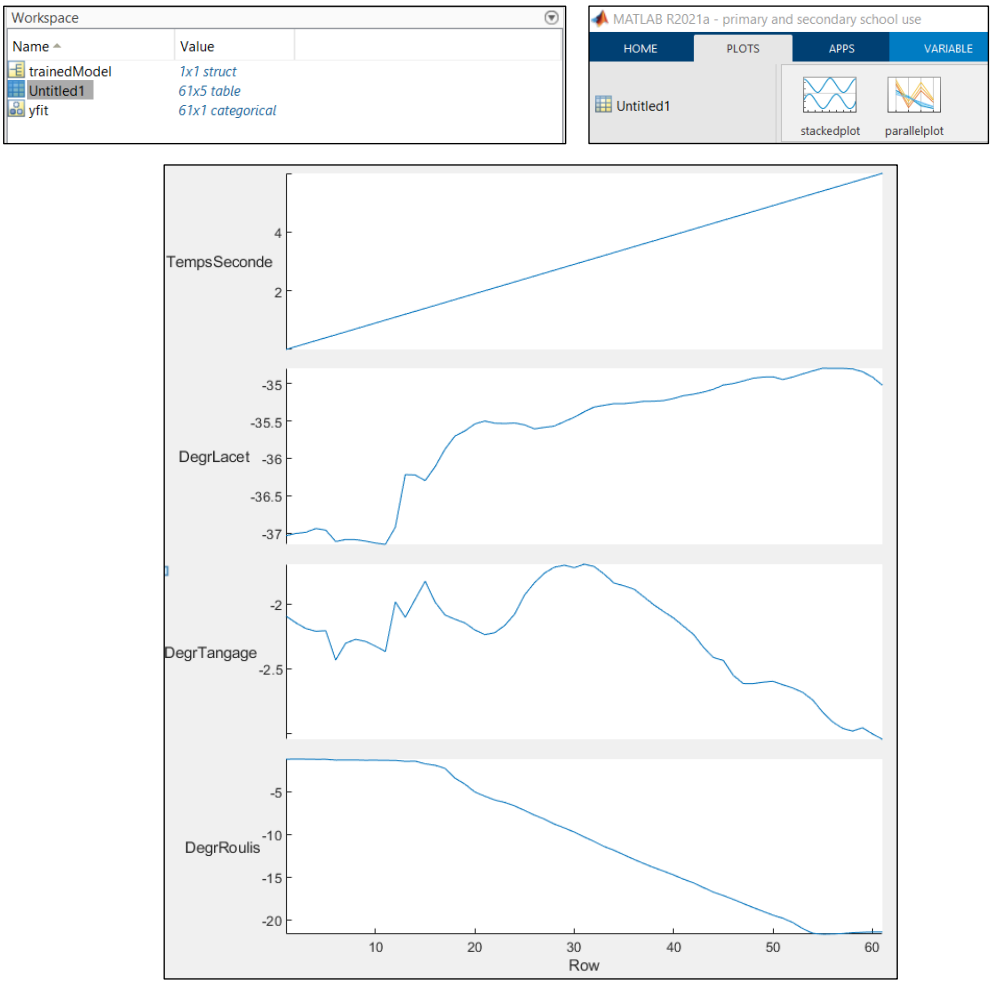
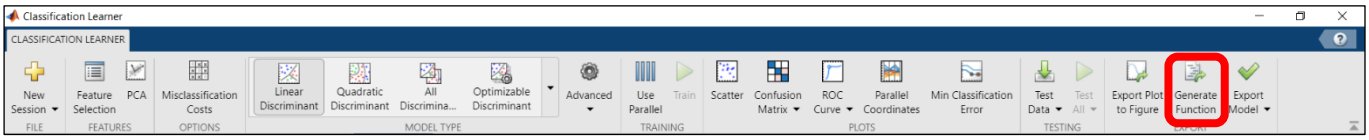


Figure 6 – Représentation graphique des données de test attestant du roulis à gauche

8 – Génération du code *Matlab*

L'application *Classification Learner* permet de générer le code *Matlab* correspondant au modèle d'apprentissage afin d'automatiser ultérieurement l'ensemble des étapes effectuées lors de l'entraînement.



Le code complet et documenté est généré de manière totalement automatique. On retrouve les différentes étapes : la sélection des prédicteurs, l'entrainement du type de modèle, le choix de la méthode de validation…. Une fois le code généré, il peut être modifié et intégré dans une application existante.

```
function [trainedClassifier, validationAccuracy] = trainClassifier(trainingData)
% [trainedClassifier, validationAccuracy] = trainClassifier(trainingData)
% Returns a trained classifier and its accuracy. This code recreates the
% classification model trained in Classification Learner app. Use the
% generated code to automate training the same model with new data, or to
% learn how to programmatically train models.
%
%  Input:
%      trainingData: A table containing the same predictor and response
%       columns as those imported into the app.
%
%  Output:
%      trainedClassifier: A struct containing the trained classifier. The
%       struct contains various fields with information about the trained
%       classifier.
%
%      trainedClassifier.predictFcn: A function to make predictions on new
%       data.
%
%      validationAccuracy: A double containing the accuracy in percent. In
%       the app, the History list displays this overall accuracy score for
%       each model.
%
% Use the code to train the model with new data. To retrain your
% classifier, call the function from the command line with your original
% data or new data as the input argument trainingData.
%
% For example, to retrain a classifier trained with the original data set
% T, enter:
%   [trainedClassifier, validationAccuracy] = trainClassifier(T)
%
% To make predictions with the returned 'trainedClassifier' on new data T2,
% use
%   yfit = trainedClassifier.predictFcn(T2)
%
% T2 must be a table containing at least the same predictor columns as used
% during training. For details, enter:
%   trainedClassifier.HowToPredict

% Auto-generated by MATLAB on 25-Aug-2021 21:01:52


% Extract predictors and response
% This code processes the data into the right shape for training the
% model.
inputTable = trainingData;
predictorNames = {'TempsSeconde', 'DegrLacet', 'DegrTangage', 'DegrRoulis'};
predictors = inputTable(:, predictorNames);
response = inputTable.Mouvement;
isCategoricalPredictor = [false, false, false, false];

% Train a classifier
% This code specifies all the classifier options and trains the classifier.
```

Figure 7a – Code modèle de prédiction *Discriminant* partie 1/2

```
classificationDiscriminant = fitcdiscr(...
    predictors, ...
    response, ...
    'DiscrimType', 'linear', ...
    'Gamma', 0, ...
    'FillCoeffs', 'off', ...
    'ClassNames', categorical({'Lacet à droite'; 'Lacet à gauche'; 'Roulis à
droite'; 'Roulis à gauche'; 'Tangage arrière'; 'Tangage avant'}));

% Create the result struct with predict function
predictorExtractionFcn = @(t) t(:, predictorNames);
discriminantPredictFcn = @(x) predict(classificationDiscriminant, x);
trainedClassifier.predictFcn = @(x) discriminantPredictFcn(predictorExtractionFcn
(x));

% Add additional fields to the result struct
trainedClassifier.RequiredVariables = {'DegrLacet', 'DegrRoulis', 'DegrTangage',
'TempsSeconde'};
trainedClassifier.ClassificationDiscriminant = classificationDiscriminant;
trainedClassifier.About = 'This struct is a trained model exported from
Classification Learner R2021a.';
trainedClassifier.HowToPredict = sprintf('To make predictions on a new table, T,
use: \n  yfit = c.predictFcn(T) \nreplacing ''c'' with the name of the variable
that is this struct, e.g. ''trainedModel''. \n \nThe table, T, must contain the
variables returned by: \n  c.RequiredVariables \nVariable formats (e.g.
matrix/vector, datatype) must match the original training data. \nAdditional
variables are ignored. \n \nFor more information, see <a href="matlab:helpview
(fullfile(docroot, ''stats'', ''stats.map''),
''appclassification_exportmodeltoworkspace'')">How to predict using an exported
model</a>.');

% Extract predictors and response
% This code processes the data into the right shape for training the
% model.
inputTable = trainingData;
predictorNames = {'TempsSeconde', 'DegrLacet', 'DegrTangage', 'DegrRoulis'};
predictors = inputTable(:, predictorNames);
response = inputTable.Mouvement;
isCategoricalPredictor = [false, false, false, false];

% Perform cross-validation
partitionedModel = crossval(trainedClassifier.ClassificationDiscriminant, 'KFold',
5);

% Compute validation predictions
[validationPredictions, validationScores] = kfoldPredict(partitionedModel);

% Compute validation accuracy
validationAccuracy = 1 - kfoldLoss(partitionedModel, 'LossFun', 'ClassifError');
```

Figure 7b – Code modèle de prédiction *Discriminant* partie 2/2