Export Classification Model to Predict New Data

Export the Model to the Workspace to Make Predictions for New Data

After you create classification models interactively in Classification Learner, you can export your best model to the workspace. You can then use the trained model to make predictions using new data.

1. In Classification Learner, select the model you want to export in the History list.
2. On the **Classification Learner** tab, in the **Export** section, click one of the export options:
   - If you want to include the data used for training the model, then select **Export Model**.

     You export the trained model to the workspace as a structure containing a classification object, such as a `ClassificationTree`, `ClassificationDiscriminant`, `ClassificationSVM`, `ClassificationKNN`, `ClassificationEnsemble`, etc.
   - If you do not want to include the training data, select **Export Compact Model**. This option exports the model with unnecessary data removed where possible. For some classifiers this is a compact classification object that does not include the training data (e.g., `CompactClassificationTree`). You can use a compact classification object for making predictions of new data, but you can use fewer other methods with it.
3. In the Export Model dialog box, edit the name for your exported variable if you want, and then click **OK**. The default name for your exported model, `trainedModel`, increments every time you export to avoid overwriting your classifiers, e.g., `trainedModel1`.

   The new variable, e.g., `trainedModel`, appears in your workspace.

   The app displays information about the exported model in the command window. Read the message to learn how to make predictions with new data.

Note

The final exported model is always trained using the full data set. The validation scheme that you use only affects the way that Classification Learner computes validation metrics.

Make Predictions for New Data

After you export a model to the workspace from Classification Learner, or run the code generated from the app, you get a `trainedModel` structure that you can use to make predictions using new data. The structure contains a classification object and a function for prediction. The structure allows you to make predictions for models that include principal component analysis (PCA).

1. To use the exported classifier to make predictions for new data, `T`, use the form:

   ```
   yfit = C.predictFcn(T)
   ```

   where `C` is the name of your variable, e.g., `trainedModel`.

   Supply the data `T` in same data type as your training data used in the app (table or matrix).
   - If you supply a table, ensure it contains the same predictor names as your training data. The `predictFcn` ignores additional variables in tables. Variable formats (e.g. matrix or vector, data type) must match the original training data.
   - If you supply a matrix, it must contain the same predictor columns or rows as your training data, in the same order and format. Do not include a response variable, any variables that you did not import in the app, or other unused variables.

   The output `yfit` contains a class prediction for each data point.
2. Examine the fields of the exported structure. For help making predictions, enter:

   ```
   C.HowToPredict
   ```

You can also extract the classification object from the exported struct for further analysis (e.g., `trainedModel.ClassificationSVM`, `trainedModel.ClassificationTree`, etc., depending on your model type). Be aware that if you used feature transformation such as PCA in the app, you will need to take account of this transformation by using the information in the PCA fields of the struct.

Generate MATLAB Code to Train the Model with New Data

After you create classification models interactively in Classification Learner, you can generate MATLAB® code for your best model. You can then use the code to train the model with new data.

Generate MATLAB code to:

- Train on huge data sets. Explore models in the app trained on a subset of your data, then generate code to train a selected model on a larger data set
- Create scripts for training models without needing to learn syntax of the different functions

- Examine the code to learn how to train classifiers programmatically
- Modify the code for further analysis, for example to set options that you cannot change in the app
- Repeat your analysis on different data and automate training

1. In Classification Learner, in the History list, select the model you want to generate code for.
2. On the **Classification Learner** tab, in the **Export** section, click **Export Model** > **Generate Code**.

   The app generates code from your session and displays the file in the MATLAB Editor. The file includes the predictors and response, the classifier training methods, and validation methods. Save the file.
3. To retrain your classifier model, call the function from the command line with your original data or new data as the input argument. New data must be the same shape.

   Copy the first line of the generated code excluding the word `function`, and edit the `trainingData` input argument to the variable name of your training data or new data. For example, to retrain a classifier trained with the `fisheriris` data set, enter:

   ```
   [trainedModel, validationAccuracy] = trainClassifier(fisheriris)
   ```

   The generated code returns a `trainedModel` structure that contains the same fields as the struct you create when you export a classifier from Classification Learner to the workspace.
4. If you want to automate training the same classifier with new data, or learn how to programmatically train classifiers, examine the generated code. The code shows you how to:
   - Process the data into the right shape
   - Train a classifier and specify all the classifier options
   - Perform cross-validation
   - Compute validation accuracy
   - Compute validation predictions and scores

Generate C Code for Prediction

If you train an SVM model using Classification Learner, you can generate C code for prediction.

C code generation requires:

- MATLAB Coder™ license
- SVM model (binary or multiclass)
- No categorical predictors or response in your data set

1. After you train an SVM model in Classification Learner, export the model to the workspace.

   Find the name of the classification model object in the exported struct. Examine the fields of the struct to find the model name, for example, `C.ClassificationSVM`, where `C` is the name of your struct, e.g., `trainedModel`.

   Model name depends on what type of SVM you trained (binary or multiclass) and whether you exported a compact model or not. Models can be `ClassificationSVM`, `CompactClassificationSVM`, `ClassificationECOC`, or `CompactClassificationECOC`.
2. Use the function `saveCompactModel` to prepare the model for code generation: `saveCompactModel(Mdl,filename)`. For example:

   ```
   saveCompactModel(C.ClassificationSVM, 'mySVM')
   ```
3. Create a function that loads the saved model and makes predictions on new data. For example:

   ```
   function label = classifyX (X) %#codegen
   %CLASSIFYX Classify using SVM Model
   %  CLASSIFYX classifies the measurements in X
   %  using the SVM model in the file mySVM.mat, and then
   %  returns class labels in label.

   CompactMdl = loadCompactModel('mySVM');
   label = predict(CompactMdl,X);
   end
   ```
4. Generate a MEX function from your function. For example:

   ```
   codegen classifyX.m -args {data}
   ```

   The `%#codegen` compilation directive indicates that the MATLAB code is intended for code generation. To ensure that the MEX function can use the same input, specify the data in the workspace as arguments to the function using the `-args` option. `data` must be a matrix containing only the predictor columns used to train the model.
5. Use the MEX function to make predictions. For example:

   ```
   labels = classifyX_mex(data);
   ```

If you used feature selection or PCA feature transformation in the app, then you need additional steps. If you used manual feature selection, supply the same columns in `X`. `X` is the input to your function.

If you used PCA in the app, use the information in the PCA fields of the exported struct to take account of this transformation. It does not matter whether you imported a table or a matrix into the app, as long as `X` contains the matrix columns in the same order. Before generating code, follow these steps:

1. Save the PCACenters and PCACoefficients fields of the trained classifier struct, `C`, to file using the following command:

   ```
   save('pcaInfo.mat', '-struct', 'C', 'PCACenters', 'PCACoefficients');
   ```

2. In your function file, include additional lines to perform the PCA transformation. Create a function that loads the saved model, performs PCA, and makes predictions on new data. For example:

   ```
   function label = classifyX (X) %#codegen
   %CLASSIFYX Classify using SVM Model
   %  CLASSIFYX classifies the measurements in X
   %  using the SVM model in the file mySVM.mat, and then
   %  returns class labels in label.
   % If you used manual feature selection in the app, ensure that X contains only the columns you included in the model.

   CompactMdl = loadCompactModel('mySVM');
   pcaInfo = load('pcaInfo.mat', 'PCACenters', 'PCACoefficients');
   % performs pca transformation
   pcaTransformedX = bsxfun(@minus, X, pcaInfo.PCACenters) * pcaInfo.PCACoefficients;
   [label, scores] = predict(CompactMdl, pcaTransformedX);
   end
   ```

For more information on C code generation workflow and limitations, see Code Generation. For examples, see `saveCompactModel` and `loadCompactModel`.

See Also

Functions

`fitcdiscr` | `fitcecoc` | `fitcensemble` | `fitcknn` | `fitcsvm` | `fitctree` | `fitglm`

Classes

`ClassificationBaggedEnsemble` | `ClassificationDiscriminant` | `ClassificationECOC` | `ClassificationEnsemble` | `ClassificationKNN` | `ClassificationSVM` | `ClassificationTree` | `CompactClassificationDiscriminant` | `CompactClassificationECOC` | `CompactClassificationEnsemble` | `CompactClassificationSVM` | `CompactClassificationTree` | `GeneralizedLinearModel`

Related Topics

- Train Classification Models in Classification Learner App