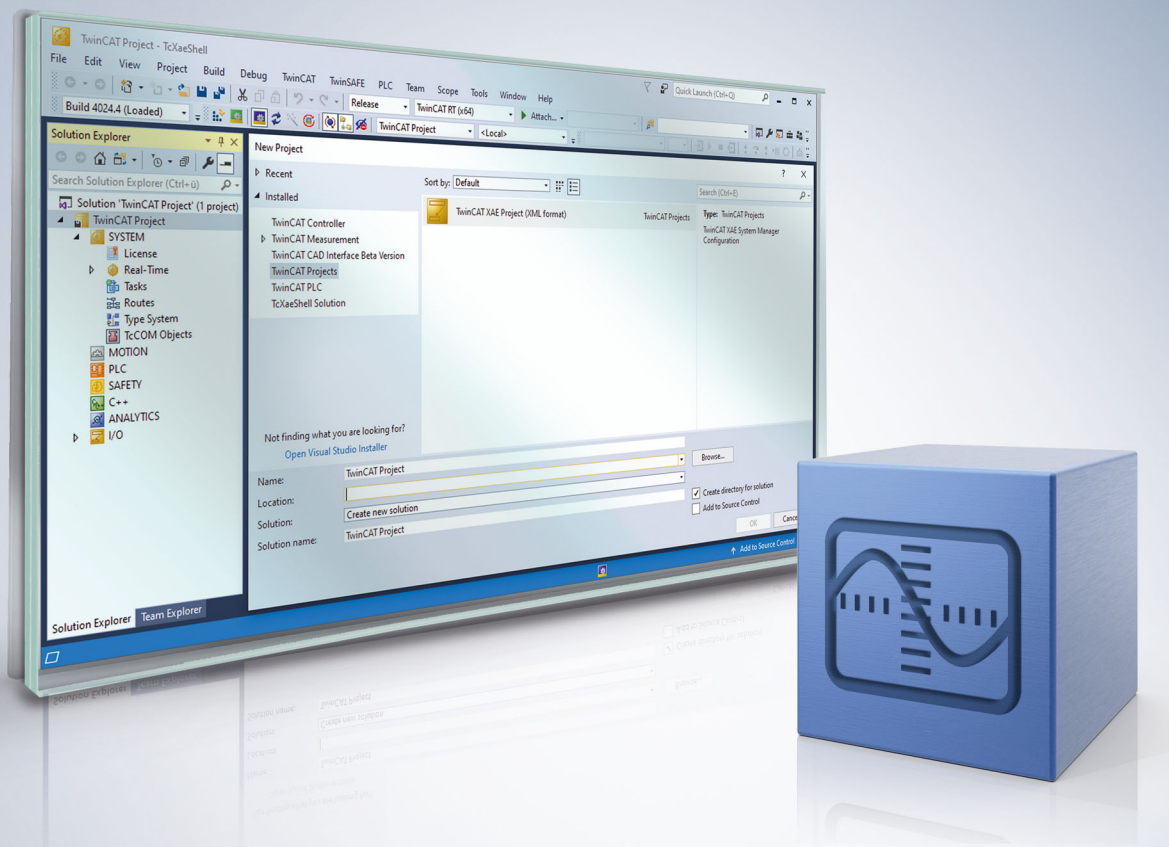
BECKHOFF New Automation Technology

Handbuch | DE

TF3800, TF3810

TwinCAT 3 | Machine Learning- und Neural Network Inference Engine



13.10.2023 | Version: 1.6.4

Inhaltsverzeichnis

1 Vorwort

1.1 Hinweise zur Dokumentation

Diese Beschreibung wendet sich ausschließlich an ausgebildetes Fachpersonal der Steuerungs- und Automatisierungstechnik, das mit den geltenden nationalen Normen vertraut ist.
Zur Installation und Inbetriebnahme der Komponenten ist die Beachtung der Dokumentation und der nachfolgenden Hinweise und Erklärungen unbedingt notwendig.
Das Fachpersonal ist verpflichtet, stets die aktuell gültige Dokumentation zu verwenden.

Das Fachpersonal hat sicherzustellen, dass die Anwendung bzw. der Einsatz der beschriebenen Produkte alle Sicherheitsanforderungen, einschließlich sämtlicher anwendbaren Gesetze, Vorschriften, Bestimmungen und Normen erfüllt.

Disclaimer

Diese Dokumentation wurde sorgfältig erstellt. Die beschriebenen Produkte werden jedoch ständig weiterentwickelt.
Wir behalten uns das Recht vor, die Dokumentation jederzeit und ohne Ankündigung zu überarbeiten und zu ändern.
Aus den Angaben, Abbildungen und Beschreibungen in dieser Dokumentation können keine Ansprüche auf Änderung bereits gelieferter Produkte geltend gemacht werden.

Marken

Beckhoff®, TwinCAT®, TwinCAT/BSD®, TC/BSD®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, Safety over EtherCAT®, TwinSAFE®, XFC®, XTS® und XPlanar® sind eingetragene und lizenzierte Marken der Beckhoff Automation GmbH.
Die Verwendung anderer in dieser Dokumentation enthaltenen Marken oder Kennzeichen durch Dritte kann zu einer Verletzung von Rechten der Inhaber der entsprechenden Bezeichnungen führen.

Patente

Die EtherCAT-Technologie ist patentrechtlich geschützt, insbesondere durch folgende Anmeldungen und Patente:
EP1590927, EP1789857, EP1456722, EP2137893, DE102015105702
mit den entsprechenden Anmeldungen und Eintragungen in verschiedenen anderen Ländern.

EtherCAT® ist eine eingetragene Marke und patentierte Technologie lizenziert durch die Beckhoff Automation GmbH, Deutschland

Copyright

1.2 Zu Ihrer Sicherheit

Sicherheitsbestimmungen

Lesen Sie die folgenden Erklärungen zu Ihrer Sicherheit.
Beachten und befolgen Sie stets produktspezifische Sicherheitshinweise, die Sie gegebenenfalls an den entsprechenden Stellen in diesem Dokument vorfinden.

Haftungsausschluss

Die gesamten Komponenten werden je nach Anwendungsbestimmungen in bestimmten Hard- und Software-Konfigurationen ausgeliefert. Änderungen der Hard- oder Software-Konfiguration, die über die dokumentierten Möglichkeiten hinausgehen, sind unzulässig und bewirken den Haftungsausschluss der Beckhoff Automation GmbH & Co. KG.

Qualifikation des Personals

Diese Beschreibung wendet sich ausschließlich an ausgebildetes Fachpersonal der Steuerungs-, Automatisierungs- und Antriebstechnik, das mit den geltenden Normen vertraut ist.

Signalwörter

Im Folgenden werden die Signalwörter eingeordnet, die in der Dokumentation verwendet werden. Um Personen- und Sachschäden zu vermeiden, lesen und befolgen Sie die Sicherheits- und Warnhinweise.

Warnungen vor Personenschäden

⚠ GEFAHR

Es besteht eine Gefährdung mit hohem Risikograd, die den Tod oder eine schwere Verletzung zur Folge hat.

⚠ WARNUNG

Es besteht eine Gefährdung mit mittlerem Risikograd, die den Tod oder eine schwere Verletzung zur Folge haben kann.

⚠ VORSICHT

Es besteht eine Gefährdung mit geringem Risikograd, die eine mittelschwere oder leichte Verletzung zur Folge haben kann.

Warnung vor Umwelt- oder Sachschäden

HINWEIS

Es besteht eine mögliche Schädigung für Umwelt, Geräte oder Daten.

Information zum Umgang mit dem Produkt

Diese Information beinhaltet z. B.:
Handlungsempfehlungen, Hilfestellungen oder weiterführende Informationen zum Produkt.

1.3 Hinweise zur Informationssicherheit

Die Produkte der Beckhoff Automation GmbH & Co. KG (Beckhoff) sind, sofern sie online zu erreichen sind, mit Security-Funktionen ausgestattet, die den sicheren Betrieb von Anlagen, Systemen, Maschinen und Netzwerken unterstützen. Trotz der Security-Funktionen sind die Erstellung, Implementierung und ständige Aktualisierung eines ganzheitlichen Security-Konzepts für den Betrieb notwendig, um die jeweilige Anlage, das System, die Maschine und die Netzwerke gegen Cyber-Bedrohungen zu schützen. Die von Beckhoff verkauften Produkte bilden dabei nur einen Teil des gesamtheitlichen Security-Konzepts. Der Kunde ist dafür verantwortlich, dass unbefugte Zugriffe durch Dritte auf seine Anlagen, Systeme, Maschinen und Netzwerke verhindert werden. Letztere sollten nur mit dem Unternehmensnetzwerk oder dem Internet verbunden werden, wenn entsprechende Schutzmaßnahmen eingerichtet wurden.

Zusätzlich sollten die Empfehlungen von Beckhoff zu entsprechenden Schutzmaßnahmen beachtet werden. Weiterführende Informationen über Informationssicherheit und Industrial Security finden Sie in unserem https://www.beckhoff.de/secguide.

Die Produkte und Lösungen von Beckhoff werden ständig weiterentwickelt. Dies betrifft auch die Security-Funktionen. Aufgrund der stetigen Weiterentwicklung empfiehlt Beckhoff ausdrücklich, die Produkte ständig auf dem aktuellen Stand zu halten und nach Bereitstellung von Updates diese auf die Produkte aufzuspielen. Die Verwendung veralteter oder nicht mehr unterstützter Produktversionen kann das Risiko von Cyber-Bedrohungen erhöhen.

Um stets über Hinweise zur Informationssicherheit zu Produkten von Beckhoff informiert zu sein, abonnieren Sie den RSS Feed unter https://www.beckhoff.de/secinfo.

2 Übersicht

Einführung

Die Idee des maschinellen Lernens ist, auf Basis von Beispieldaten einen generalisierten Zusammenhang zwischen Eingaben und Ausgaben zu erlernen. Entsprechend wird hierzu eine gewisse Menge an Trainingsdaten benötigt, anhand derer ein **Modell** trainiert wird. Im Training des Modells werden Parameter des Modells über ein mathematisches Verfahren automatisch an die Trainingsdaten angepasst. Im maschinellen Lernen stehen dem Anwender eine Vielzahl von verschiedenen Modellen zur Verfügung. Die Wahl und die Auslegung der Modelle ist Teil des Engineering Prozesses. Unterschiedliche Modell-Typen oder Auslegungen von Modellen erfüllen unterschiedliche Aufgaben, wobei die wichtigste Unterteilung die Klassifikation und die Regression bildet.

Klassifikation: Das Modell erhält eine Eingabe (ein Bild, einen/mehrere Vektoren, …) und ordnet dieser eine Klasse zu. Der Ausgang ist entsprechend eine kategorische Größe. Diese Klassen können beispielsweise Gut-Teil oder Schlecht-Teil sein. Es könnten auch mehrere Klassen unterschieden werden, zum Beispiel Güteklasse A, B, C, D.

Regression: Das Modell erhält eine Eingabe und erzeugt eine kontinuierliche Ausgabe. Dabei werden nicht nur direkt gelernte Eingaben direkt gelernten Ausgaben zugeordnet (wie bei einem Look-up-Table), sondern das Modell ist zudem in der Lage, nicht gelernte Eingaben zu interpolieren, beziehungsweise zu extrapolieren, insofern es gut generalisiert. Es wird ein funktionaler Zusammenhang gelernt.

Ist ein Modell trainiert worden, kann es für die gelernte Aufgabe eingesetzt werden, d. h. das Modell wird zur **Inferenz** eingesetzt.

TwinCAT 3 Machine Learning Runtime

Beckhoff stellt mit den Produkten **TF3800 TwinCAT 3 Machine Learning Inference Engine** und **TF3810 Neural Network Inference Engine** Komponenten für die Inferenz von Modellen in der TwinCAT XAR zur Verfügung. Für beide Produkte wird eine gemeinsame Software-Basis genutzt, welche im Folgenden als Machine Learning Runtime (kurz: ML Runtime) bezeichnet wird.

Die ML Runtime ist ein in TwinCAT 3 integriertes Modul (TcCOM), welches in der TwinCAT XAR ausgeführt wird. Somit ist der Zugriff auf das Modell Interface (Modell-Inputs und Modell-Outputs) sowie auch die Ausführung des in der ML Runtime geladenen Modells in harter Echtzeit möglich.

Die Auftrennung zwischen TF3800 und TF3810 ist durch unterschiedliche Lizenzen begründet. Die benötigte Lizenz richtet sich nach dem zu ladenden ML-Modell. Grundsätzlich wird für das Laden und Ausführen von klassischen ML-Modellen die TF3800 vorausgesetzt. Für das Laden von Neuronalen Netzen wird die Lizenz TF3810 abgefragt. Die TF3810 beinhaltet die TF3800 Lizenz.

Weiterführende Informationen zu unterstützten Modellen und benötigten Lizenzen. [▶ 21]

Workflow

Grundsätzlich besteht der Ablauf des maschinellen Lernens und die Integration in TwinCAT 3 aus drei Phasen:

1. Dem Sammeln von Daten
2. Dem Trainieren eines Modells
3. Dem Deployment in die TwinCAT XAR

Zur **Erhebung von Daten** aus der Steuerung stehen Ihnen eine Vielzahl von TwinCAT-Produkten zur Verfügung:
Siehe TwinCAT Scope, TwinCAT Database Server, TwinCAT Analytics Logger, TwinCAT IoT, …

ML-Modelle können in einer Vielzahl von Software-Werkzeugen trainiert werden. Die **Erstellung von ML-Modellen** wird in der Regel in Programmierumgebungen wie Python oder R durchgeführt. Es existieren diverse Open Source und kostenfreie Werkzeuge, wie PyTorch, Keras und Scikit-learn, welche sich für das Erstellen von ML-Modellen eignen. Trainierte Modelle können aus diesen Werkzeugen in einem

standardisierten Format, als ONNX Datei, exportiert werden. Die ONNX-Datei ist eine standardisierte Beschreibung des trainierten ML-Modells. Diese Datei wird zunächst in ein für TwinCAT aufbereitetes Format konvertiert (XML- oder BML-Datei).

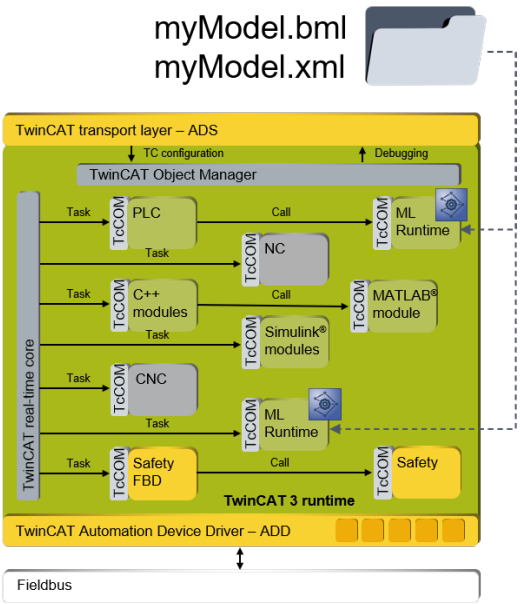
Weitere Informationen:

- Erstellen von ONNX-Dateien [▶ 63]
- Konvertieren von ONNX-Dateien [▶ 65]

Für das **Deployment des Modells** werden in TwinCAT zwei Wege angeboten:

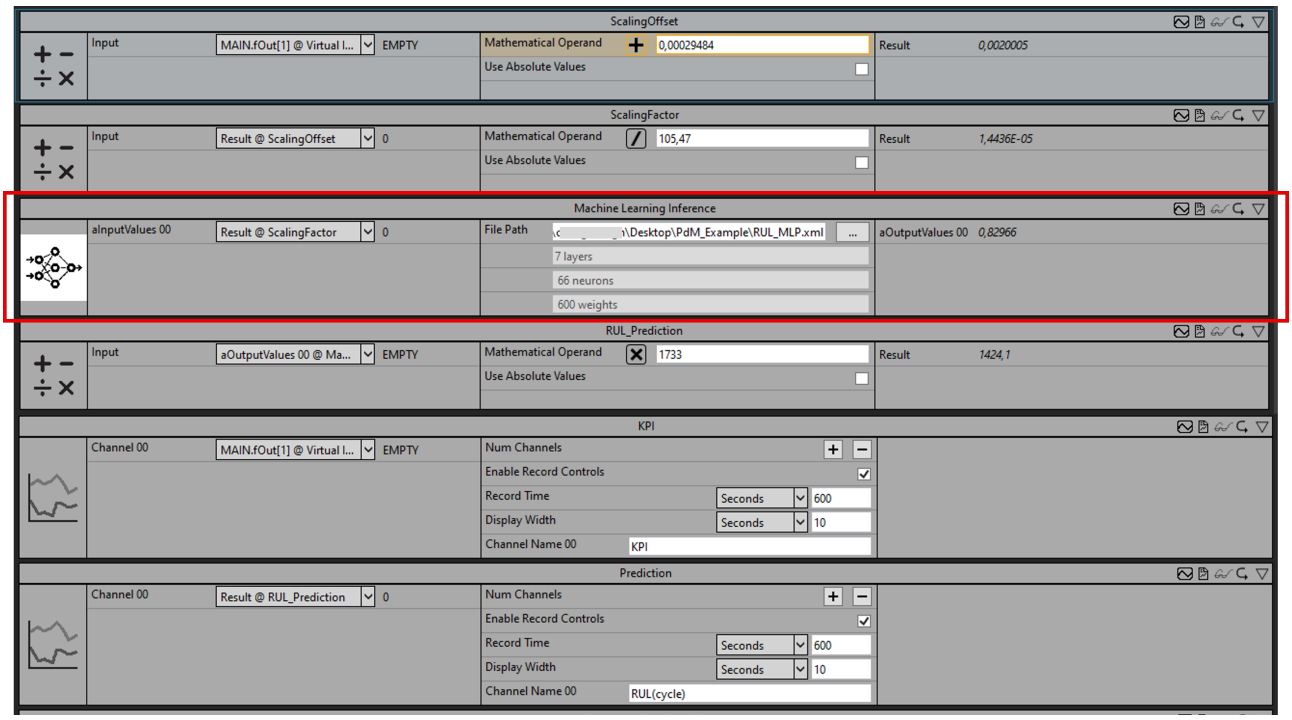
1. Es wird die Bibliothek TC3_MLL zur Nutzung in der SPS-Umgebung bereitgestellt. ML-Modelle können über einen Methodenaufruf asynchron geladen und anschließend, durch Aufruf einer weiteren Methode, zyklisch im SPS-Programm ausgeführt werden. Weiterführende Informationen zur PLC API [▶ 83]
2. Eine einfache Methode für maschinelles Lernen ohne Programmieraufwand: Ein TcCOM-Objekt, welches im TwinCAT-Objektbaum in der Entwicklungsumgebung eingefügt und konfiguriert werden kann. Das TcCOM lädt zum Systemstart das konfigurierte Modell und führt dieses in der zugeordneten Zykluszeit aus.
   Weiterführende Informationen zum ML-TcCOM [▶ 81]

Das folgende Schaubild illustriert die tiefe Integration der Machine Learning Runtime in die TwinCAT XAR. Das Modul ist, wie alle TwinCAT Runtime Objekte, ein TcCOM und es ist entsprechend tief verankert in der harten Echtzeit.



Integration von Machine Learning in TwinCAT Analytics

Die Produkte Machine Learning Inference Engine und Neural Network Inference Engine können auch in den TwinCAT Analytics Workflow integriert werden. Nähere Informationen dazu entnehmen Sie der TwinCAT Analytics Dokumentation.


ScalingOffset
Input
MAIN.fOut[1] @ Virtual I...
EMPTY
Mathematical Operand
0,00029484
Use Absolute Values
Result
0,0020005
ScalingFactor
Input
Result @ ScalingOffset
0
Mathematical Operand
105,47
Use Absolute Values
Result
1,4436E-05
Machine Learning Inference
aInputValues 00
Result @ ScalingFactor
0
File Path
\Desktop\PdM_Example\RUL_MLP.xml
...
aOutputValues 00
0,82966
7 layers
66 neurons
600 weights
RUL_Prediction
Input
aOutputValues 00 @ Ma...
EMPTY
Mathematical Operand
1733
Use Absolute Values
Result
1424,1
KPI
Channel 00
MAIN.fOut[1] @ Virtual I...
EMPTY
Num Channels
Enable Record Controls
Record Time
Seconds
600
Display Width
Seconds
10
Channel Name 00
KPI
Prediction
Channel 00
Result @ RUL_Prediction
0
Num Channels
Enable Record Controls
Record Time
Seconds
600
Display Width
Seconds
10
Channel Name 00
RUL(cycle)

3 Installation

3.1 Systemvoraussetzungen

Runtime

Technische Daten	Beschreibung
Betriebssystem	Windows 7 64bit, Windows Embedded Standard 7 64bit, Windows 10 64bit, TwinCAT/BSD
Zielplattform	PC-Architektur (x64)
Minimale TwinCAT-Version	TwinCAT 3.1 Build 4024.0 und höher
Erforderliches TwinCAT-Setup-Level	TwinCAT 3 XAR
Erforderliche TwinCAT-Lizenz	TF3800 TC3 Machine Learning Inference Engine oder TF3810 TC3 Neural Network Inference Engine

Die erforderliche Lizenz ist abhängig vom geladenen ML-Modell, siehe Unterstützte Modelle des maschinellen Lernens [▶ 21].

Engineering

Technische Daten	Beschreibung
Minimale TwinCAT-Version	TwinCAT 3.1. Build 4024.0 und höher
Erforderliches TwinCAT-Setup-Level	TwinCAT 3 XAE

Das Setup ist sowohl auf dem Engineering als auch auf dem Runtime PC auszuführen.
Es können 7-Tage Testlizenzen für die Runtime generiert werden

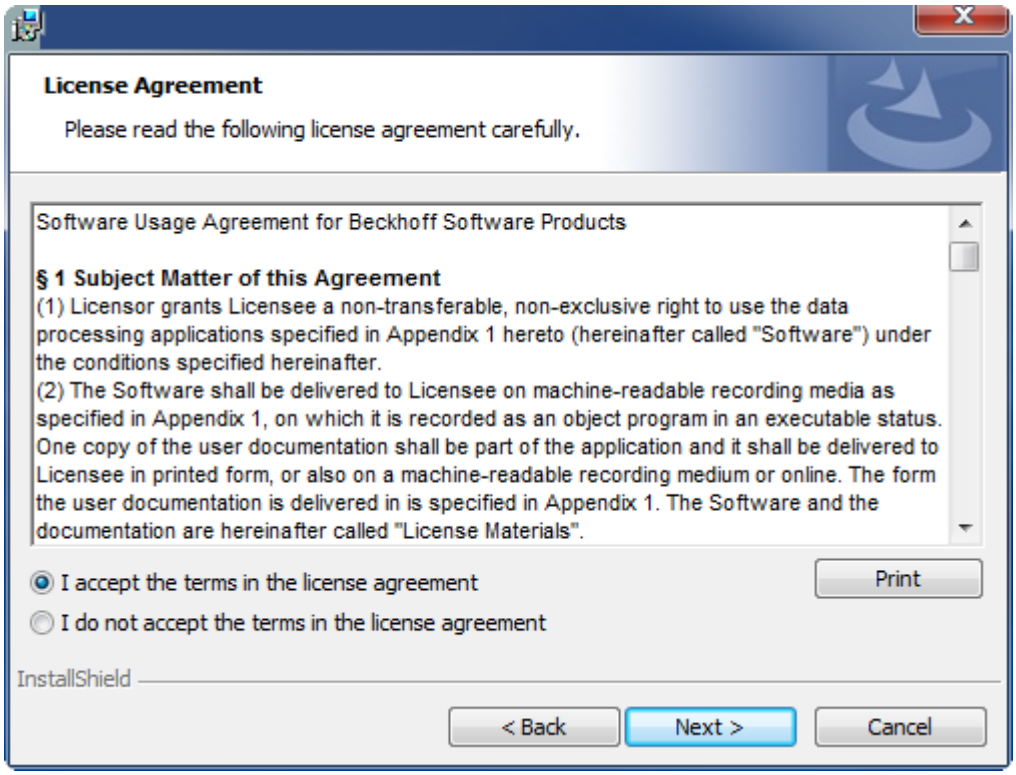
3.2 Installation

Nachfolgend wird beschrieben, wie die TwinCAT 3 Function für Windows-basierte Betriebssysteme installiert wird.

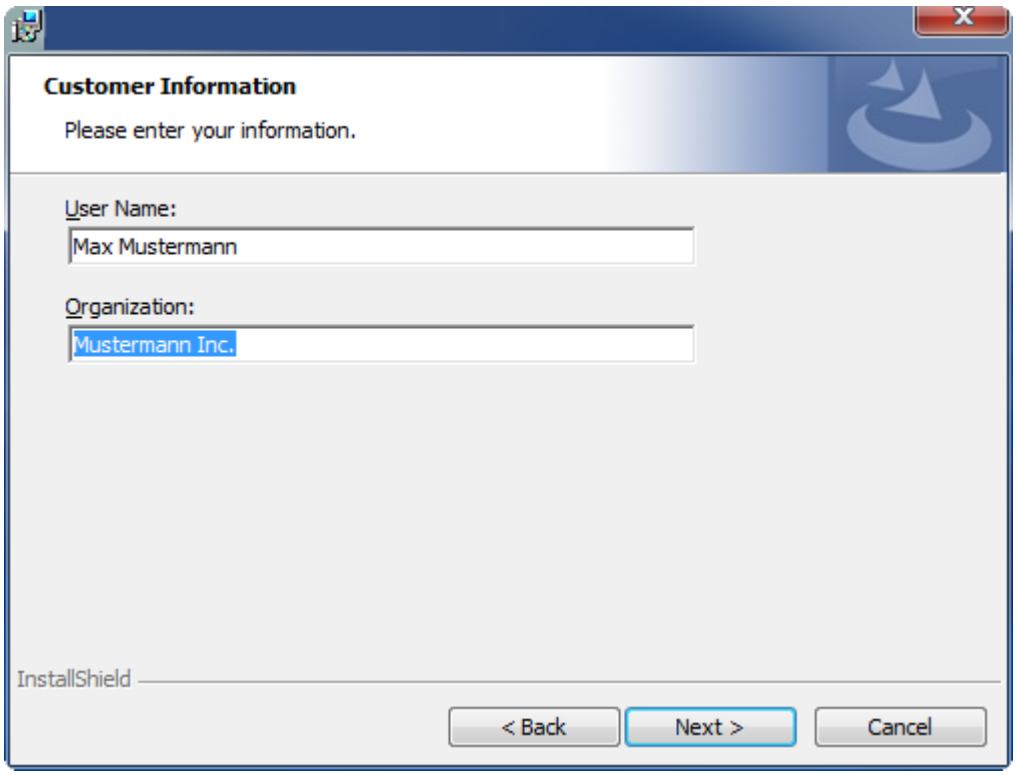
✓ Die Setup-Datei der TwinCAT 3 Function wurde von der Beckhoff-Homepage heruntergeladen.

1. Führen Sie die Setup-Datei als Administrator aus. Wählen Sie dazu im Kontextmenü der Datei den Befehl **Als Administrator ausführen**.
   ⇨ Der Installationsdialog öffnet sich.

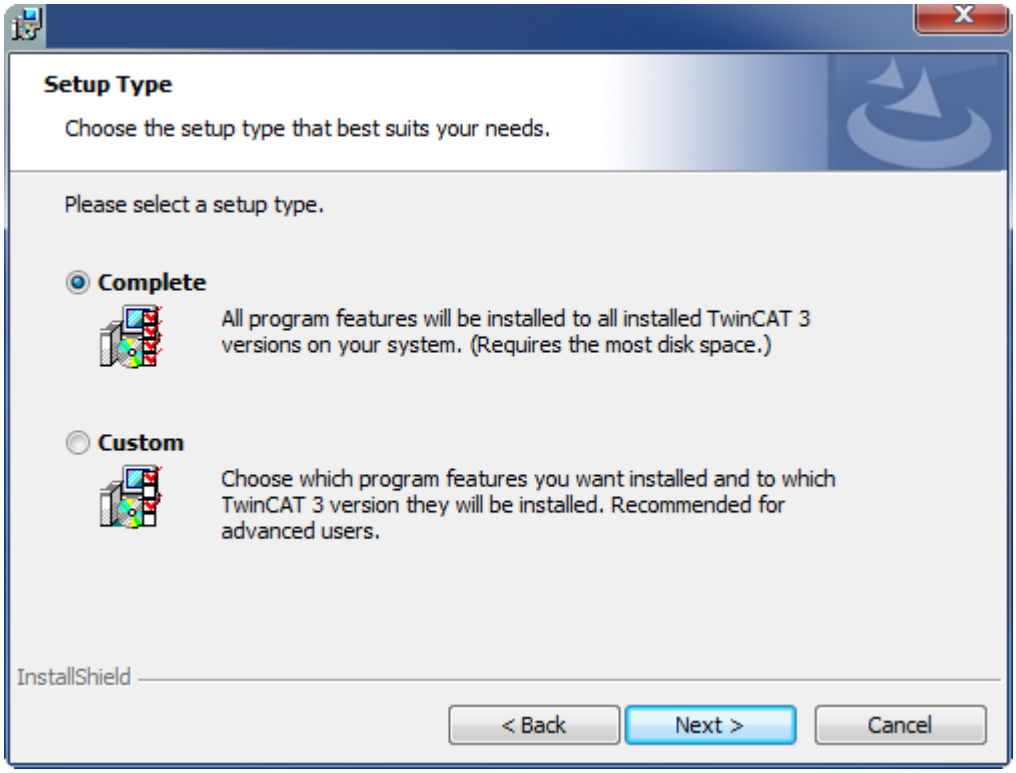
2. Akzeptieren Sie die Endbenutzerbedingungen und klicken Sie auf **Next**.



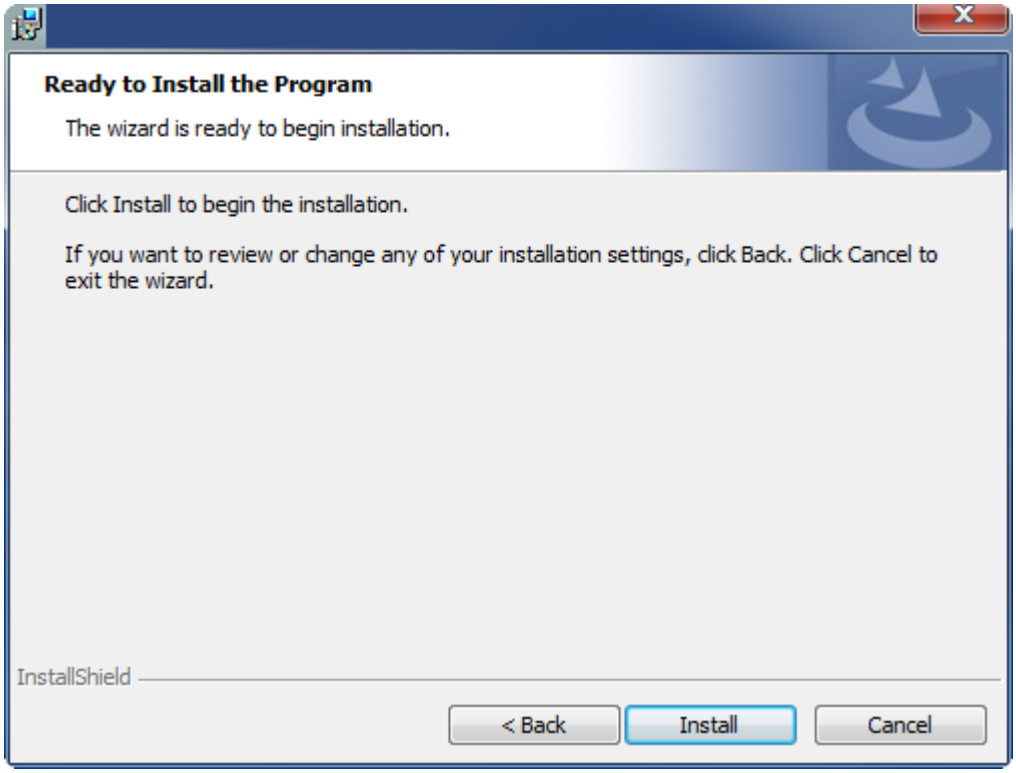
3. Geben Sie Ihre Benutzerdaten ein.



4. Wenn Sie die TwinCAT 3 Function vollständig installieren möchten, wählen Sie **Complete** als Installationstyp. Wenn Sie die Komponenten der TwinCAT 3 Function separat installieren möchten, wählen Sie **Custom**.

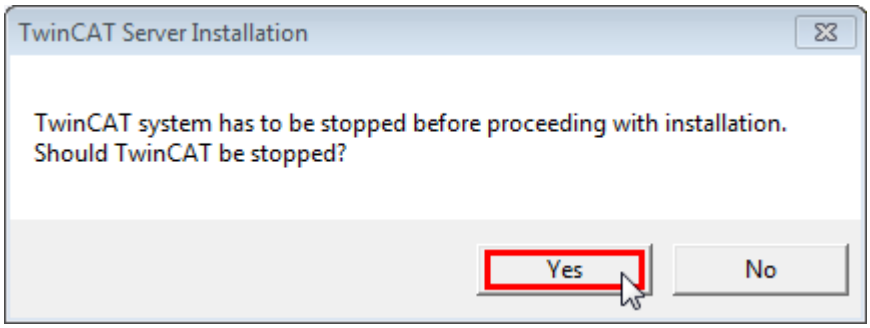


5. Wählen Sie **Next** und anschließend **Install**, um die Installation zu beginnen.

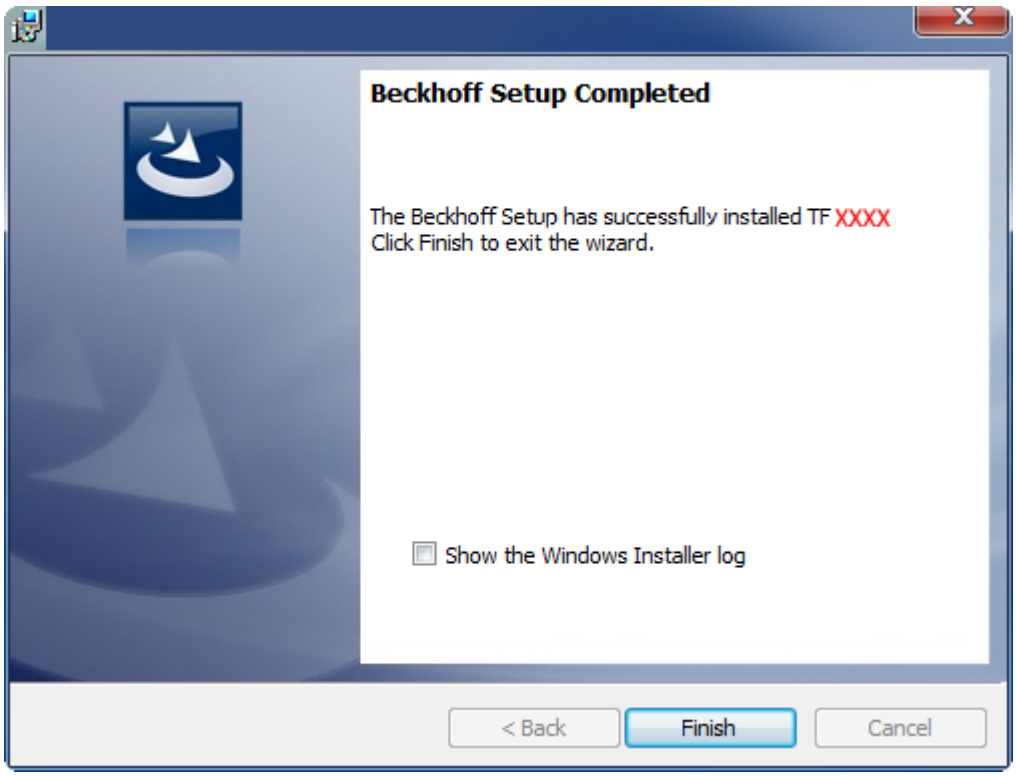


⇨ Ein Dialog weist Sie darauf hin, dass das TwinCAT-System für die weitere Installation gestoppt werden muss.

6. Bestätigen Sie den Dialog mit **Yes.**



7. Wählen Sie **Finish**, um das Setup zu beenden.



⇨ Die TwinCAT 3 Function wurde erfolgreich installiert und kann lizenziert werden (siehe Lizenzierung [▶ 14]).

3.3 Lizenzierung

Die TwinCAT 3 Function ist als Vollversion oder als 7-Tage-Testversion freischaltbar. Beide Lizenztypen sind über die TwinCAT-3-Entwicklungsumgebung (XAE) aktivierbar.

Lizenzierung der Vollversion einer TwinCAT 3 Function

Die Beschreibung der Lizenzierung einer Vollversion finden Sie im Beckhoff Information System in der Dokumentation „TwinCAT 3 Lizenzierung“.

Lizenzierung der 7-Tage-Testversion einer TwinCAT 3 Function

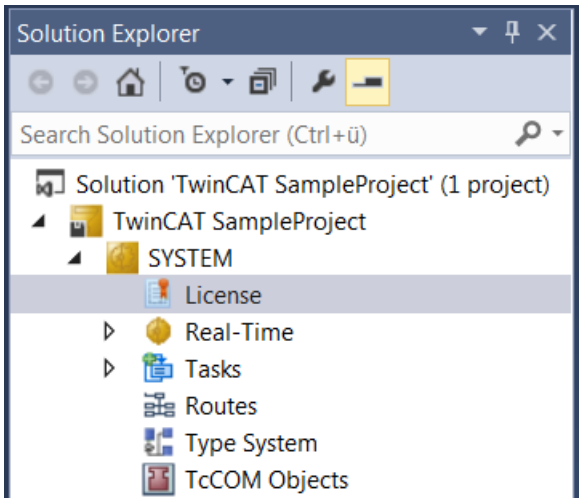
Eine 7-Tage-Testversion kann nicht für einen TwinCAT-3-Lizenz-Dongle freigeschaltet werden.

1. Starten Sie die TwinCAT-3-Entwicklungsumgebung (XAE).
2. Öffnen Sie ein bestehendes TwinCAT-3-Projekt oder legen Sie ein neues Projekt an.

3. Wenn Sie die Lizenz für ein Remote-Gerät aktivieren wollen, stellen Sie das gewünschte Zielsystem ein. Wählen Sie dazu in der Symbolleiste in der Drop-down-Liste **Choose Target System** das Zielsystem aus.

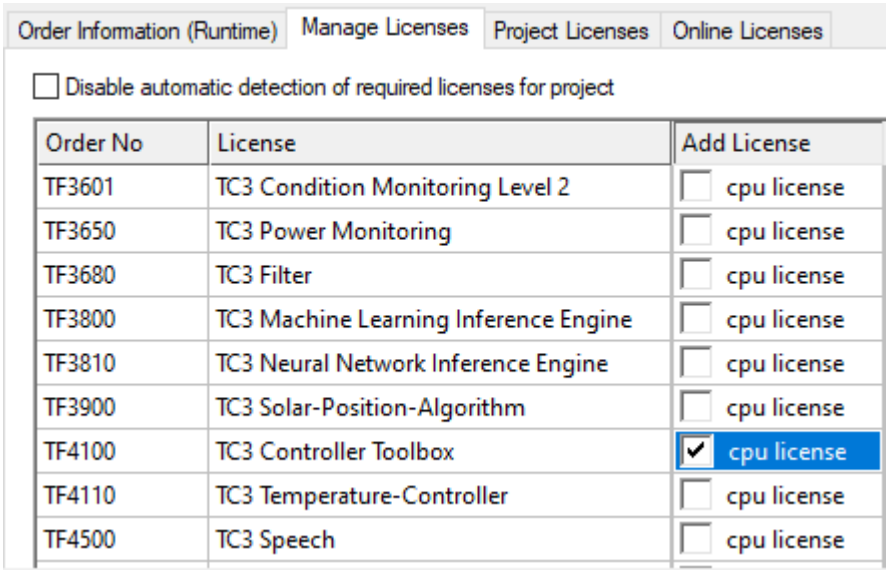
   ⇨ Die Lizenzierungseinstellungen beziehen sich immer auf das eingestellte Zielsystem. Mit der Aktivierung des Projekts auf dem Zielsystem werden automatisch auch die zugehörigen TwinCAT-3-Lizenzen auf dieses System kopiert.

4. Klicken Sie im **Solution Explorer** im Teilbaum **SYSTEM** doppelt auf **License.**



   ⇨ Der TwinCAT-3-Lizenzmanager öffnet sich.

5. Öffnen Sie die Registerkarte **Manage Licenses.** Aktivieren Sie in der Spalte **Add License** das Auswahlkästchen für die Lizenz, die Sie Ihrem Projekt hinzufügen möchten (z. B. „TF4100 TC3 Controller Toolbox“).

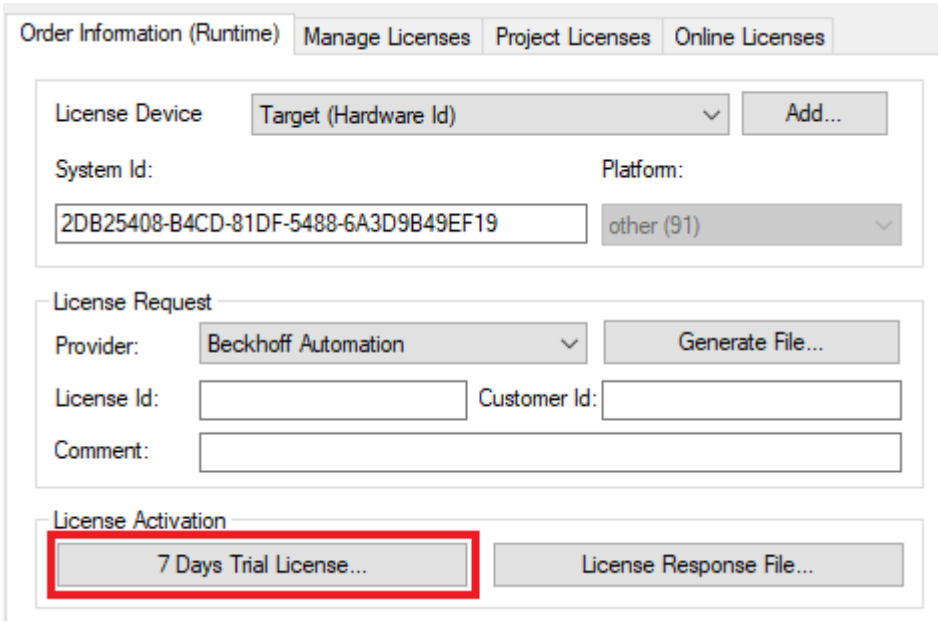


Order No	License	Add License
TF3601	TC3 Condition Monitoring Level 2	cpu license
TF3650	TC3 Power Monitoring	cpu license
TF3680	TC3 Filter	cpu license
TF3800	TC3 Machine Learning Inference Engine	cpu license
TF3810	TC3 Neural Network Inference Engine	cpu license
TF3900	TC3 Solar-Position-Algorithm	cpu license
TF4100	TC3 Controller Toolbox	cpu license
TF4110	TC3 Temperature-Controller	cpu license
TF4500	TC3 Speech	cpu license

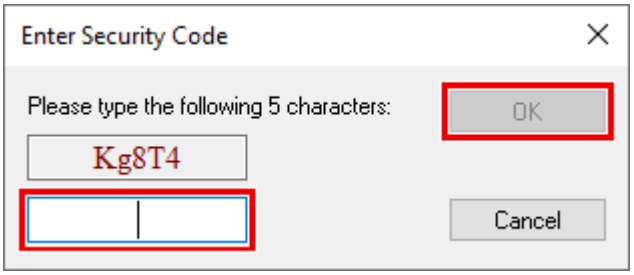
6. Öffnen Sie die Registerkarte **Order Information (Runtime).**

   ⇨ In der tabellarischen Übersicht der Lizenzen wird die zuvor ausgewählte Lizenz mit dem Status „missing“ angezeigt.

7. Klicken Sie auf **7 Days Trial License...**, um die 7-Tage-Testlizenz zu aktivieren.



⇨ Es öffnet sich ein Dialog, der Sie auffordert, den im Dialog angezeigten Sicherheitscode einzugeben.



8. Geben Sie den Code genauso ein, wie er angezeigt wird, und bestätigen Sie ihn.
9. Bestätigen Sie den nachfolgenden Dialog, der Sie auf die erfolgreiche Aktivierung hinweist.
   ⇨ In der tabellarischen Übersicht der Lizenzen gibt der Lizenzstatus nun das Ablaufdatum der Lizenz an.
10. Starten Sie das TwinCAT-System neu.

⇨ Die 7-Tage-Testversion ist freigeschaltet.

4 Schnellstart

Modell konvertieren (optional)

Eine ML-Beschreibungsdatei (*KerasMLPExample_cos.XML*) liegt bereits im ZIP des https://infosys.beckhoff.com/content/1031/tf38x0_tc3_ML_NN_Inference_Engine/Resources/8746884875.zip. Diese Datei kann direkt für die Einbindung in TwinCAT verwendet werden oder optional in ein Binärformat (BML) konvertiert werden.

Im Folgenden wird exemplarisch gezeigt, wie eine Beschreibungsdatei konvertiert wird. Dasselbe Vorgehen gilt auch für den üblicheren Fall, dass eine ONNX-Datei konvertiert werden soll.

✓ Der Machine Learning Model Manager [▶ 67] ist geöffnet.
Menüleiste: (Extensions)* > TwinCAT > Machine Learning > Machine Learning Model Manager
* Nur bei Visual Studio 2019

1. Nutzen Sie das Convert tool.
2. Klicken Sie auf **Select file** und wählen Sie die Datei *KerasMLPExample_cos.XML.*
3. Wählen Sie im Drop-Down-Menü **Convert to *.bml**
4. Klicken Sie auf **Convert files.**

⇨ Das entsprechende BML erscheint unter *<TwinCATpath>\Functions\TF38xx-Machine-Learning\ConvertToolFiles*

Den Default-Pfad zum Ablegen konvertierter ML-Beschreibungsdateien können Sie im Machine Learning Model Manager mit *Select Target Path* verändern. Die Änderung ist persistent.

Sie können auch mehrere Dateien gleichzeitig durch Multi-Select konvertieren.
Die konvertierten Dateien werden standardmäßig auf dem XAE System im Ordner *<TwinCATpath>\Functions\TF38xx-Machine-Learning\ConvertToolFiles* abgelegt.

Einbindung in TwinCAT über die SPS API

Nachfolgend wird beschrieben, wie Sie die ML-Beschreibungsdatei in TwinCAT laden und zyklisch ausführen. Dazu wird zunächst auf die SPS API [▶ 83] eingegangen.

- Erstellen Sie zunächst ein TwinCAT Projekt und legen Sie ein SPS Projekt an
- Fügen Sie die SPS Bibliothek Tc3_MLL unter dem Knoten References hinzu

In der **Deklaration** erstellen Sie bitte eine Instanz des FB_MllPrediction. In diesem einfachen Fall enthält die Beschreibungsdatei ein MLP mit einem Eingang und einem Ausgang vom Typ FP32, entsprechend werden Variablen für Input und Output als REAL angelegt. Eine allgemeingültigere Möglichkeit zur Handhabung der Ein- und Ausgänge finden Sie in den Beispielen zur PLC API [▶ 96].

Außerdem erstellen Sie eine String-Variable, welche den Dateinamen inkl. Pfad zur ML-Beschreibungsdatei beinhaltet (Pfad auf dem Zielsystem). Kopieren Sie die entsprechende Datei an diesen Ort auf dem Zielsystem (FTP, RDP, Shared Folder, …).
Weiterführende Informationen zu diesem Schritt finden Sie hier [▶ 78].

HINWEIS

Pfad der Beschreibungsdatei auf dem Zielsystem

Achten Sie auf die Einstellungen des File Writer und der Schreibrechte auf dem Zielsystem.

```
PROGRAM MAIN
VAR
   fbPredict   :  FB_MllPrediction;      // ML Interface
   nInputDim, nOutputDim : UDINT := 1;
   fDataIn, fDataOut : REAL;
   sModelName    : T_MaxString  := 'C:/TwinCAT/3.1/Boot/ML_Boot/KerasMLPExample_cos.xml';
   hrErrorCode   : HRESULT;
   bLoadModel    : BOOL;
   nState        : INT := 0;
END_VAR
```

Im **Implementationsteil** erstellen Sie beispielsweise einen Zustandsautomaten, welcher es Ihnen ermöglicht, zwischen dem Idle state, Config state, Predict state und Error state zu wechseln.

Im ersten Zustand warten Sie zunächst auf den Befehl zum Laden einer Beschreibungsdatei. Anschließend wird die Configure-Methode [▶ 87] des `fbPredict` Funktionsbausteines so lange aufgerufen bis ein `TRUE` zurückgeliefert wird. Dieses steht einen Zyklus lang an und bedeutet, dass die Konfiguration abgeschlossen ist. Es wird dann geprüft, ob ein Fehler aufgetreten ist. Wenn kein Fehler aufgetreten ist, wird zum nächsten Zustand, dem Predict state, geschaltet. Der Zustandsautomat verbleibt im Predict state solange kein Fehler auftritt oder ein (neues) Modell geladen werden soll.

```
CASE nState OF
    0: // idle state
      IF bLoadModel THEN
         bLoadModel := FALSE;
         nState := 10;
      END_IF
   10: // Config state
      fbPredict.stPredictionParameter.MlModelFilepath := sModelName; // provide model path and name
      IF fbPredict.Configure() THEN // load model
         IF fbPredict.bError THEN
            nState := 999;
            hrErrorCode := fbPredict.hrErrorCode;
         ELSE // no error -> proceed to predict state
            nState := 20;
         END_IF
      END_IF
   20: // Predict state
      fbPredict.Predict(
         pDataInp := ADR(fDataIn),
         nDataInpDim := nInputDim,
         fmtDataInpType := ETcMllDataType.E_MLLDT_FP32_REAL,
         pDataOut := ADR(fDataOut),
         nDataOutDim := nOutputDim,
         fmtDataOutType := ETcMllDataType.E_MLLDT_FP32_REAL,
         nEngineId := 0,
         nConcurrencyId := 0);

      IF fbPredict.bError THEN // error handling
         nState := 999;
         hrErrorCode := fbPredict.hrErrorCode;
      ELSIF bLoadModel THEN // load (updated) model
         bLoadModel := FALSE;
         nState := 10;
      END_IF;
   999: // Error state
       // add error handling here
END_CASE
```

Im Predict state wird die Predict-Methode [▶ 92] des `fbPredict` verwendet. Diese führt das geladene Modell aus. Der Methode werden die Eingangsgrößen über die ersten drei Parameter der Methode mitgeteilt – Pointer auf eine SPS-Variable, Anzahl der Eingänge und zugehöriger Datentyp. Selbiges ist für die Ausgangsgrößen anzugeben (Parameter 4 bis 6). Die `nEngineId` und `nConcurrencyId` werden in diesem einfachen Beispiel nicht benötigt und immer mit dem Wert Null übergeben. Details zu diesen Parametern finden Sie in den Beispielen Ausführliches Beispiel [▶ 96] und Paralleler, nicht-blockierender, Zugriff auf ein Inferenzmodul [▶ 96].

Bevor Sie das Projekt auf einem Target aktivieren, müssen Sie im Projektbaum unter System>License im Tab **Manage Licenses** die TF3810 Lizenz **manuell** anwählen, da Sie ein Mehrlagiges Perzeptron (MLP) laden möchten.

Sie können nun die Konfiguration aktivieren. Loggen Sie sich in die SPS ein und starten Sie das Programm. Indem Sie nun die `bLoadModel` Variable im Online View auf `TRUE` setzen wird das Modell geladen und beginnt mit der Prädiktion Sie können die Eingabevariable `fDataIn` manipulieren und sich die Ergebnisse in der Ausgabe `fDataOut` ansehen. Das geladene Mehrlagiges Perzeptron bildet näherungsweise eine Cosinusfunktion im Eingangsbereich von [-pi, pi] auf den Wertebereich [-1,1] ab. Außerhalb des Bereichs [-pi,pi] divergiert die Funktion zunehmend von der Cosinusfunktion.

TwinCAT_Project5.Untitled1.MAIN

Expression	Type	Value
fbPredict	FB_MllPrediction	
nInputDim	UDINT	1
nOutputDim	UDINT	1
fDataIn	REAL	3.14
fDataOut	REAL	-1.018022
sModelName	T_MaxString	'C:/TwinCAT/3.1...
hrErrorCode	HRESULT	16#00000000
bLoadModel	BOOL	FALSE
nState	INT	20

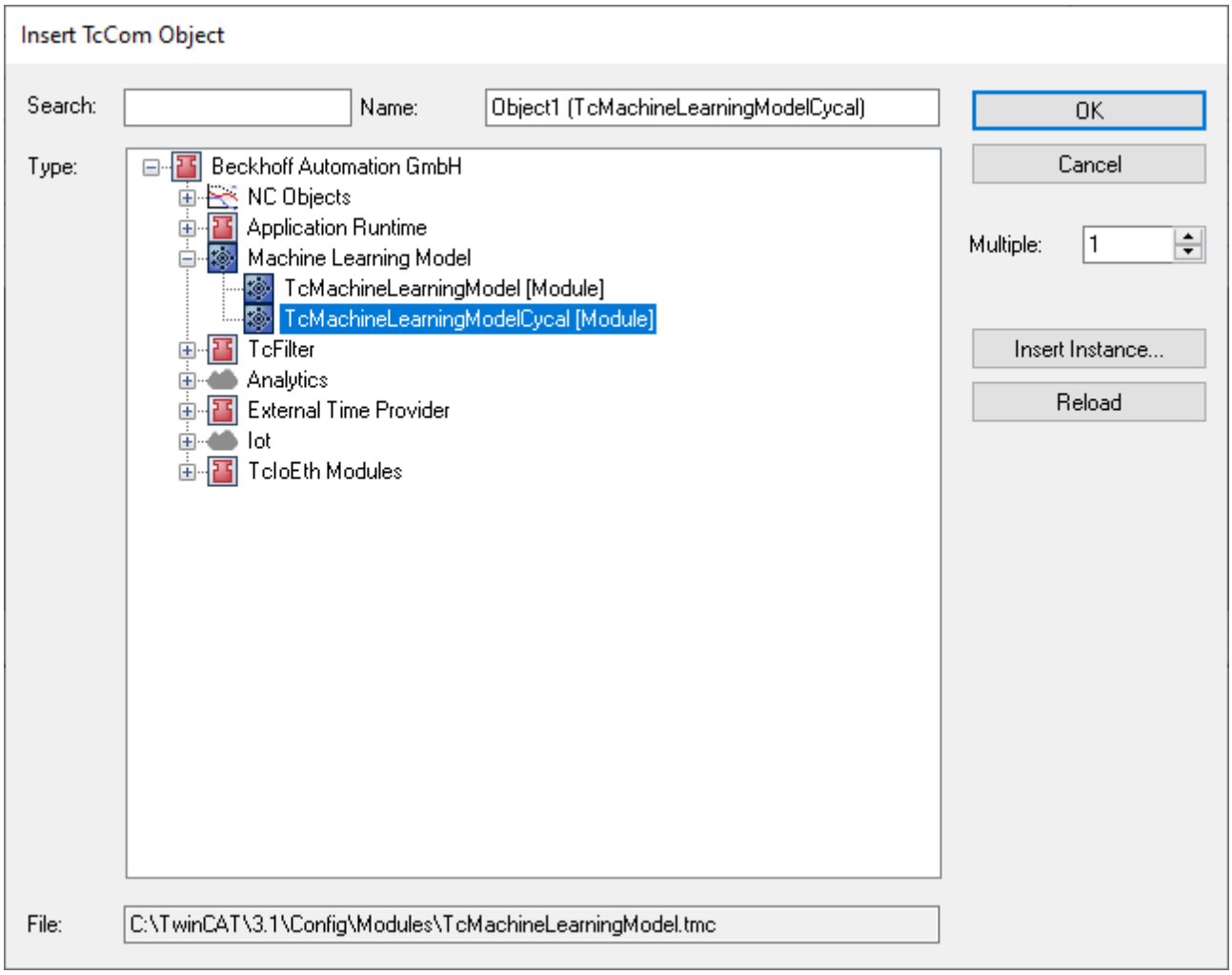
Das oben beschriebene Beispiel können Sie hier [▶ 96] herunterladen.

Einbeziehung eines Modelles mittels TcCOM-Objekt

In diesem Abschnitt wird die Ausführung von Modellen des Maschinellen Lernens mittels eines vorbereiteten TcCOM-Objektes behandelt. Eine ausführliche Beschreibung dazu finden Sie hier [▶ 81]. Diese Schnittstelle bietet eine einfache und übersichtliche Möglichkeit, Modelle zu laden, in Echtzeit auszuführen und mittels des Prozessabbilds entsprechende Verknüpfungen in die eigene Applikation zu erzeugen.

Vorbereitetes TcCOM-Objekt TcMachineLearningModelCycal erzeugen

1. Selektieren Sie dazu den Knoten **TcCOM Objects** mit der rechten Maustaste und wählen Sie **Add New Item…**

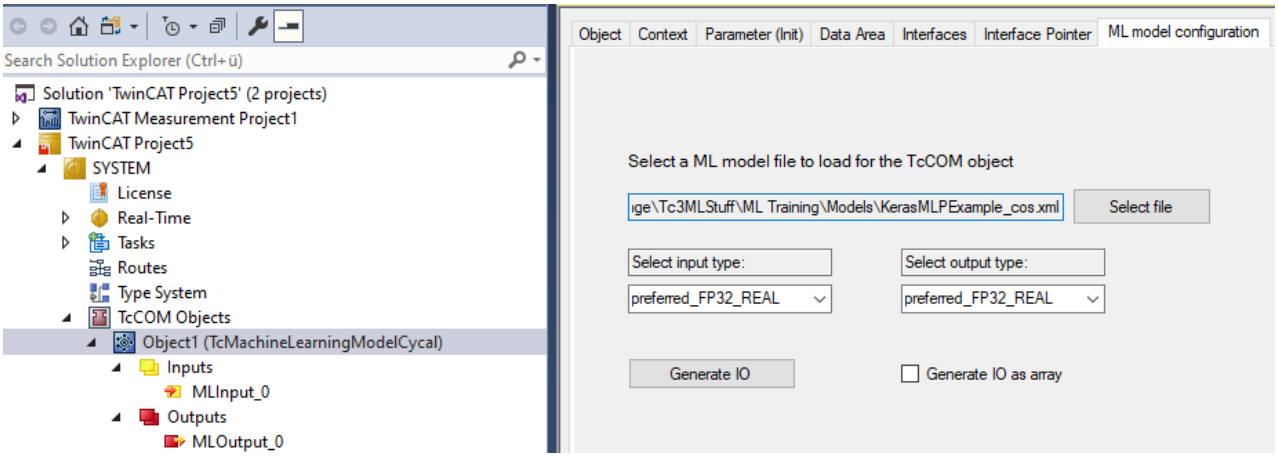


Unter Tasks eine neue TwinCAT Task erzeugen und der neu erzeugten Instanz des TcMachineLearningModelCycal diesen Task-Kontext zuweisen

2. Gehen Sie dazu auf den Reiter **Context** des erzeugten Objekts.
3. Wählen Sie im Drop-Down-Menü Ihre erzeugte Task aus.

⇨ Die Instanz des TcMachineLearningModelCycal verfügt über einen Reiter **ML model configuration**. Hier können Sie die Beschreibungsdatei des ML Algorithmus (XML oder BML) laden und bekommen im Anschluss daran verfügbare Datentypen für Ein- und Ausgänge des selektierten Modells angezeigt.

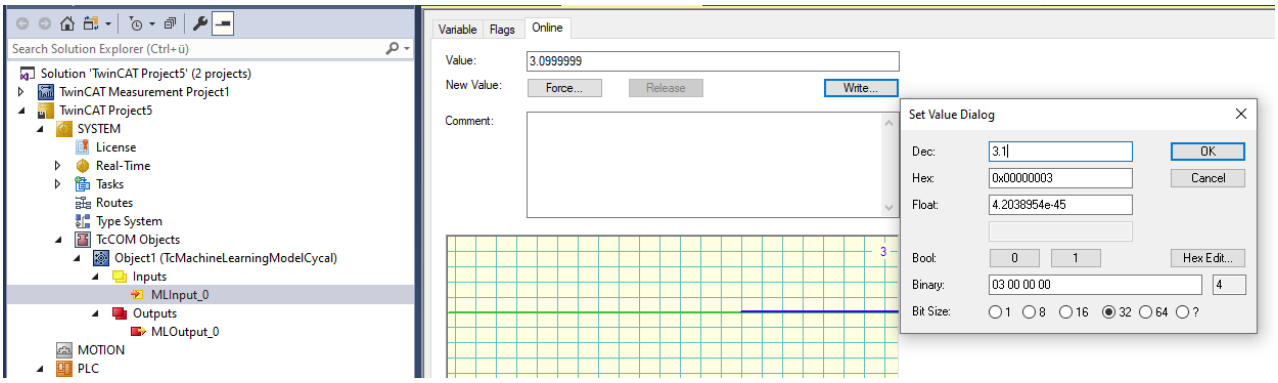
- Die Datei muss sich nicht auf dem Zielsystem befinden. Es kann vom Entwicklungssystem aus selektiert werden und wird dann beim Aktivieren der Konfiguration auf das Zielsystem überspielt.
  - Es wird zwischen präferierten und unterstützen Datentypen unterschieden. Der Unterschied ist lediglich, dass zur Laufzeit eine Konvertierung des Datentyps erfolgt, sollte ein nicht präferierter ausgewählt werden.
    Dies kann bei Verwendung nicht-präferierter Datentypen zu minimalen Performance-Einbußen führen.
- Automatisch werden die Datentypen für Ein- und Ausgänge zunächst auf den präferierten Datentypen gesetzt. Durch einen Klick auf **Generate IO** wird das Prozessabbild des selektierten Modells erstellt. Entsprechend erhalten Sie durch das Laden der *KerasMLPExample_cos.xml* ein Prozessabbild mit einem Input vom Typ REAL und ein Output vom Typ REAL.



Projekt auf Target aktivieren

1. Bevor Sie das Projekt auf einem Target aktivieren, müssen Sie im Projektbaum unter System>License im Tab **Manage Licenses** die TF3810 Lizenz manuell anwählen, da Sie ein Mehrlagiges Perzeptron laden möchten.
2. Aktivieren Sie die Konfiguration.

⇨ Sie können nun das Modell durch händisches Schreiben auf dem Input testen.



5 Machine Learning Modelle und Dateiformate

In diesem Kapitel wird eine Übersicht über unterstützte Machine Learning Modelle [▶ 21] gegeben sowie Anhaltspunkte für die Wahl eines passenden Modells [▶ 59] aufgeführt. Darüber hinaus wird für jeden Algorithmus anhand von Beispielen beschrieben, wie Sie trainierte Modelle als ONNX-Datei aus einer Python Umgebung exportieren [▶ 61] können. Die exportierte ONNX-Datei muss in eine TwinCAT-spezifische XML- oder BML-Datei umgewandelt [▶ 65] werden. Dazu stehen mehrere Interfaces (CLI, API und GUI) zu einem Konverter zur Verfügung, um den Dateimanagement Prozess in Ihre existierende Software-Landschaften einzubinden.

5.1 Unterstützte Modelle des maschinellen Lernens

In untenstehender Tabelle sind alle unterstützten Modell-Typen, inkl. der dafür vorausgesetzten Lizenz und Softwareversion, aufgeführt.

Auswahl eines Modells

Einen Einstieg, welche Kriterien Sie für die Wahl eines Modells in Betracht ziehen sollten, finden Sie hier: Machine Learning Cheat Sheet: Auswahl von Modellen [▶ 59].

ONNX-Export für unterstützte Modell-Typen

Für alle unterstützten Modell-Typen sind im Abschnitt Beispiele zum ONNX-Export [▶ 63] Python-Beispiele für den ONNX-Export aus unterschiedlichen Frameworks gegeben.

Erforderliche Lizenz für unterstützte Modell-Typen

Anhand des Modell-Typs, welcher in die Machine Learning Runtime geladen wird, wird die benötigte TwinCAT Lizenz differenziert. Zu beachten ist, dass die Lizenz TF3810 die Lizenz TF3800 beinhaltet, also bei gültiger TF3810 Lizenz alle Modelle geladen werden können, welche eine TF3800 oder TF3810 Lizenz benötigen.

Unterstützte Modelle

Zur Lizenzierung siehe auch: Systemvoraussetzungen [▶ 11].

Modell-Typ	Lizenz	Verfügbar ab Setup Version
Stützvektormaschine [▶ 28] (SVM)	TF3800	3.1.42.0
Principal Component Analysis [▶ 35] (PCA)	TF3800	3.1.57.0
k-Means [▶ 34]	TF3800	3.1.57.0
Random Forest [▶ 44]	TF3800	3.1.58.0
Mehrlagiges Perzeptron [▶ 22] (MLP)	TF3810	3.1.42.0
Decision Tree [▶ 37]	TF3800	3.1.62.0
Extra Tree [▶ 39]	TF3800	3.1.62.0
Extra Trees [▶ 42]	TF3800	3.1.62.0
Gradient Boosting [▶ 47]	TF3800	3.1.62.0
Hist Gradient Boosting [▶ 49]	TF3800	3.1.62.0
XGBoost [▶ 52]	TF3800	3.1.62.0
LightGBM [▶ 55]	TF3800	3.1.62.0

5.1.1 Mehrlagiges Perzeptron

Ein mehrlagiges Perzeptron (MLP) kann sowohl zur Klassifikation als auch zur Regression [▶ 61] genutzt werden. Die Grundidee eines MLP ist die Verkettung kleinster Einheiten, sogenannter Neuronen, in einem Netzwerk. Jedes Neuron nimmt Informationen von vorherigen Neuronen oder direkt über Modelleingänge auf und verarbeitet diese. Dabei findet ein gerichteter Informationsfluss durch dieses Netzwerk von Eingaben zu Ausgaben statt.

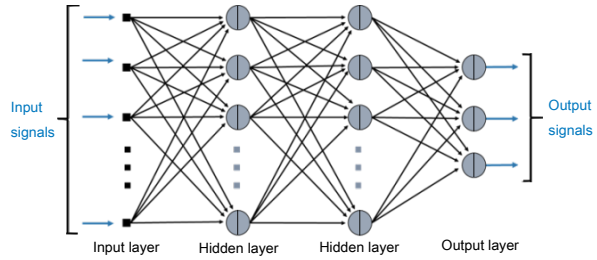
Ein Neuron verarbeitet dessen Eingaben **x** als gewichtete Summe plus einen Ordinatenwert und transformiert das Zwischenergebnis mit einer Aktivierungsfunktion.

$$y = f_{\mathrm{act}}(\mathbf{w}\mathbf{x} + w_0)$$

Neuronen werden üblicherweise in Schichten (Lagen) angeordnet, welche dann hintereinander verknüpft werden. Hat ein Netzwerk mehr als eine Lage zwischen Eingaben und Ausgaben, so spricht man von einem mehrlagigen Perzeptron.

Im untenstehenden Schaubild wird der Aufbau verdeutlicht.

- **Input layer:** Weist keine eigenen Neuronen auf. Dient als Eingabeschicht und definiert die Anzahl sowie den Datentyp der Eingaben.
- **Hidden layer:** Schicht mit eigenen Neuronen. Die Schicht wird charakterisiert durch die Anzahl der Neuronen sowie der gewählten Aktivierungsfunktion. Es können beliebig viele hidden Layer hintereinander geschichtet werden.
- **Output layer:** Schicht mit eigenen Neuronen. Die Anzahl der Neuronen sowie deren Aktivierungsfunktion richtet sich nach der zu realisierenden Applikation.



Unterstützte Eigenschaften

ONNX-Support

Unterstützt werden die ONNX-Operatoren:

- MatMul (matrix multiplication) gefolgt von ADD (add)
- GEMM (general matrix multiplication)

Darüber hinaus werden folgende Aktivierungsfunktionen unterstützt:

Aktivierungsfunktion	Beschreibung
tanh	Tangens Hyperbolicus (-1,1)
sigmoid	Sigmoid Funktion – eine Exponentialfunktion (0,1)
softmax	Softmax – eine normalisierte Exponentialfunktion – oft für Klassifikation verwendet (0,1)
sine	Sinus Funktion (-1,1)
cosine	Cosinus Funktion (-1,1)
relu	„Rectifier“ – Positiver Anteil ist linear – gute Lerneigenschaften bei tiefen Netzen (0,inf)
abs	Absolutwert der Eingabe (0,inf)
linear/id	Lineare Identität – einfache lineare Funktion f(x)=x (-inf,inf)
exp	Eine einfache Exponentialfunktion e^(x) (0,inf)
logsoftmax	Logarithmus von Softmax – manchmal effizienter als Softmax in der Berechnung (-inf,inf)
sign	Vorzeichenfunktion (-1,1)
softplus	Manchmal besser als relu, aufgrund der Differenzierbarkeit (0,inf)
softsign	Bedingt besseres Konvergenzverhalten als tanh (-1,1)

Beispiele für den ONNX-Support von MLPs aus Pytorch, Keras und Scikit-learn finden Sie hier: ONNX-Export eines MLP [▶ 23].

Unterstützte Datentypen

Es ist zu unterscheiden zwischen „supported datatype“ und „preferred datatype“. Der preferred datatype entspricht der Präzision der Ausführungs-Engine.

Der preferred datatype ist floating point 32 (E_MLLDT_FP32-REAL).

Bei Verwendung eines supported datatype wird eine effiziente Typ-Konvertierung automatisch in der Bibliothek durchgeführt. Durch die Typ-Konvertierung kann es zu minimalen Performance-Einbußen kommen.

Eine Liste der supported datatypes finden Sie unter ETcMlIDataType [▶ 83].

Weitere Anmerkungen

Es sind seitens der Software keine Limitierungen bezüglich der Anzahl von Layern oder der Anzahl von Neuronen vorgesehen. Hinsichtlich Berechnungsdauer und Speicherbedarf sind die Grenzen der verwendeten Hardware zu beachten.

5.1.1.1 ONNX-Export eines MLP

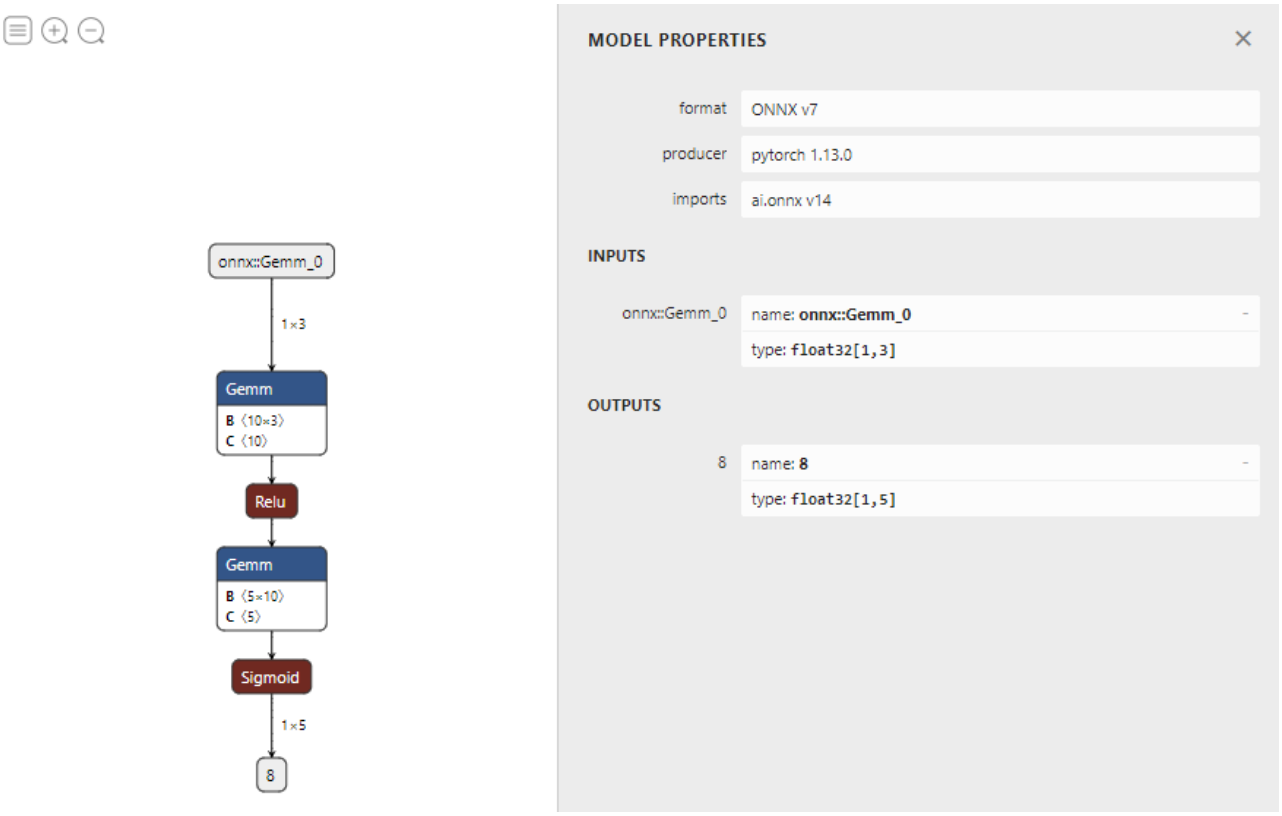
Download Python Samples

Ein Zip-Archiv mit allen Samples finden Sie hier: Beispiele zum ONNX-Export [▶ 63]

MLP Regressor mit PyTorch

```
import torch
import numpy as np

input_dim = 3
output_dim = 5
n_samples = 100
dummy_input = np.random.random((n_samples, input_dim))
dummy_output = np.random.random((n_samples, output_dim))
tensor_in = torch.FloatTensor(dummy_input)
tensor_out = torch.FloatTensor(dummy_output)
train_dataset = torch.utils.data.TensorDataset(tensor_in, tensor_out)
train_loader = torch.utils.data.DataLoader(train_dataset, shuffle=True, batch_size=32)
class MLP_Net(torch.nn.Module):
    def __init__(self):
        super().__init__()
```

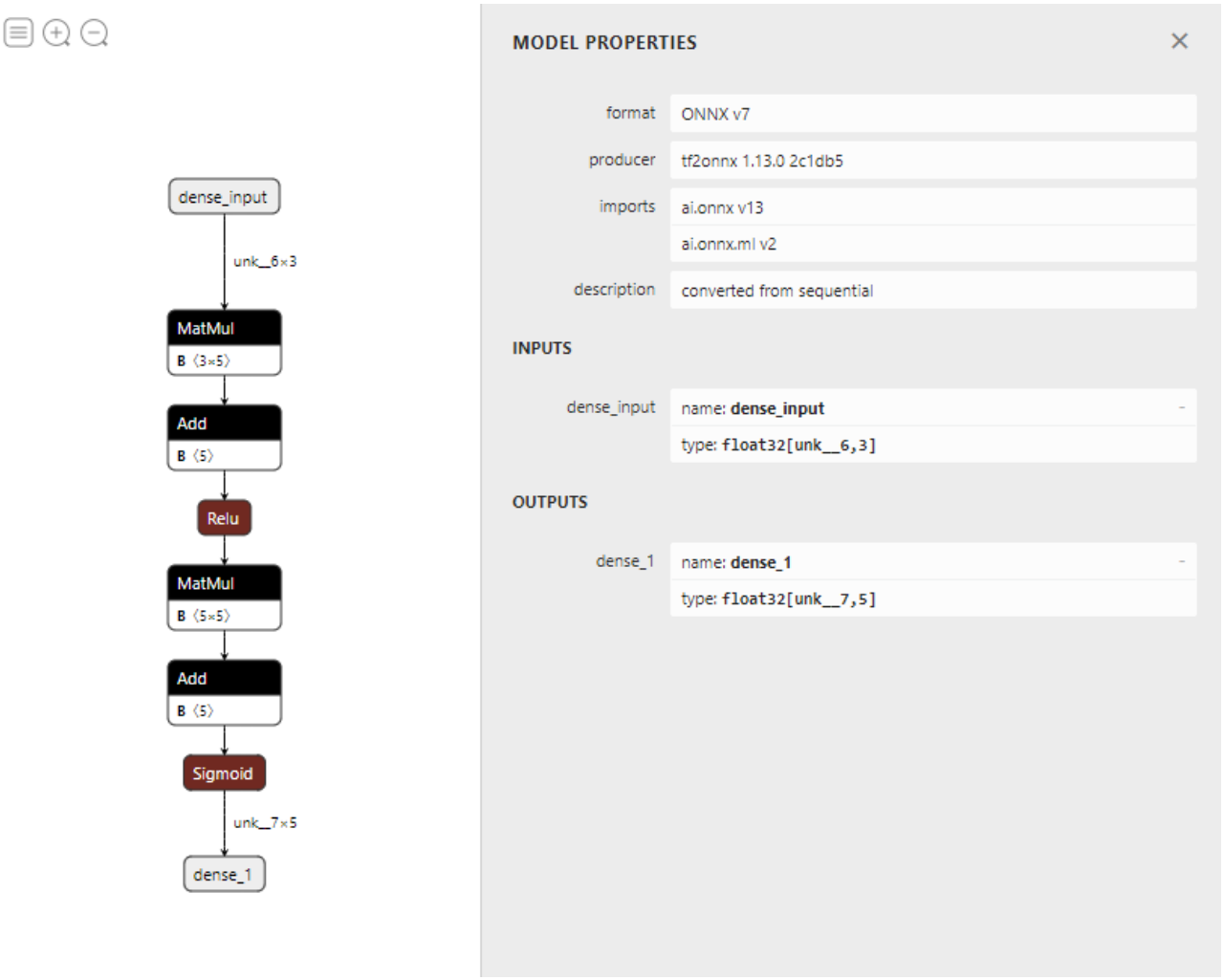
```
        self.fc1 = torch.nn.Linear(input_dim, 10)
        self.fc2 = torch.nn.Linear(10, output_dim)

    def forward(self, x):
        x = torch.nn.functional.relu(self.fc1(x))
        x = torch.sigmoid(self.fc2(x))
        return x
mlp_net = MLP_Net()
optimizer = torch.optim.Adam(mlp_net.parameters(), lr =0.001)
n_epochs = 5
for epoch in range(n_epochs):
    for batch in train_loader:
        input, output = batch
        mlp_net.zero_grad()
        pred_out = mlp_net(input)
        criterion = torch.nn.MSELoss()
        loss = criterion(pred_out, output)
        loss.backward()
        optimizer.step()
onnx_file = 'pytorch_mlp.onnx'
tensor_input_size = torch.FloatTensor(np.random.random((1,input_dim))) # First dimension must be 1
torch.onnx.export(mlp_net, tensor_input_size, onnx_file, verbose=True)
```



MLP Regressor mit Keras

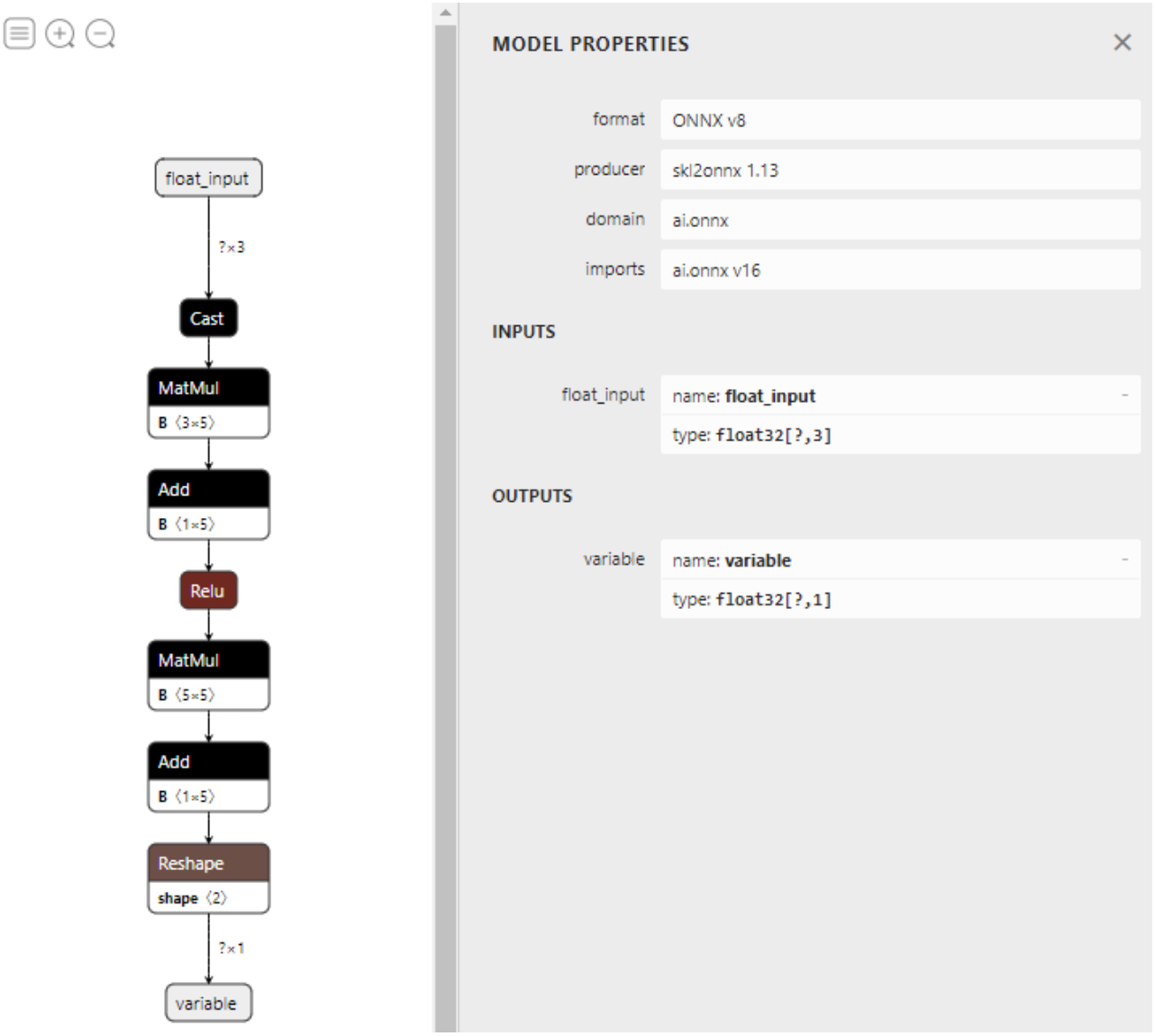
```
import tensorflow as tf
import numpy as np
import tf2onnx

input_dim = 3
output_dim = 5
n_samples = 100

dummy_input = np.random.random((n_samples, input_dim,))
dummy_output = np.random.random((n_samples, output_dim))

model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(5, input_shape = (input_dim,), activation=tf.keras.activations.relu,
 use_bias = True))
model.add(tf.keras.layers.Dense(output_dim, activation='sigmoid'))
model.build()
model.summary()
learning_rate = 0.001
loss = 'mean_squared_error'
optimizer = tf.keras.optimizers.Adamax(lr=learning_rate)
```

```
model.compile(optimizer=optimizer, loss=loss)
model.fit(x=dummy_input, y=dummy_output, batch_size=32 ,epochs=5,verbose=1, shuffle=True)

filename = 'tf_keras_mlp'
onnx_model, _ = tf2onnx.convert.from_keras(model)
with open(filename+'.onnx','wb') as f:
    f.write(onnx_model.SerializeToString())
```



MLP Regressor mit Scikit-learn

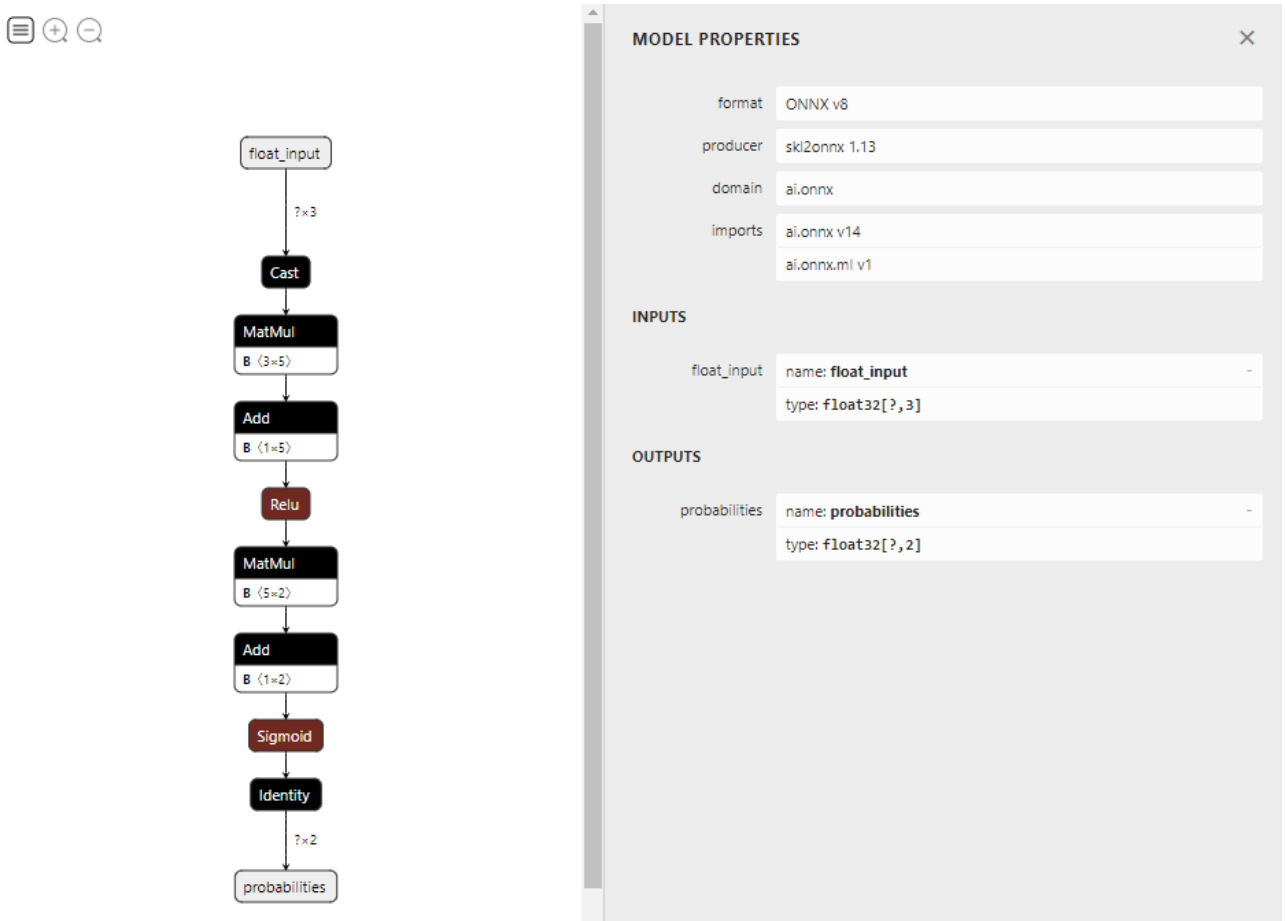
```
import sklearn.neural_network as skl
import numpy as np

input_dim = 3
output_dim = 5
n_samples = 100
dummy_input = np.random.random((n_samples,input_dim))
dummy_output = np.random.random((n_samples,output_dim))

model = skl.MLPRegressor(hidden_layer_sizes =(5), activation ='relu')
model.fit(dummy_input,dummy_output)

filename = 'skl_relu_reg'
from skl2onnx import convert_sklearn
from skl2onnx.common.data_types import FloatTensorType
initial_type = [('float_input',FloatTensorType([None,input_dim]))]

onx = convert_sklearn(model,initial_types=initial_type)
with open(filename+'.onnx','wb') as f:
    f.write(onx.SerializeToString())
```



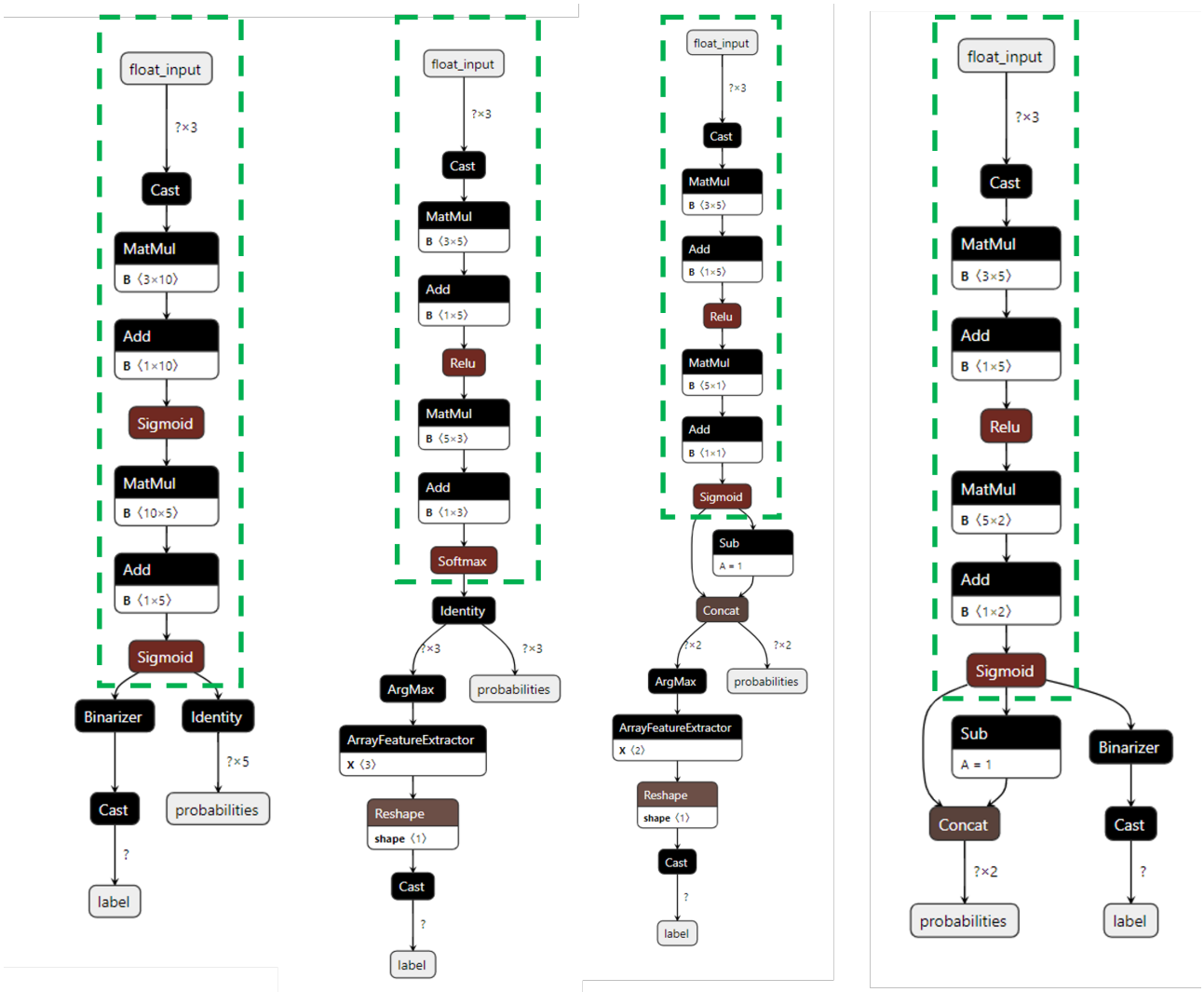
MLP Classifier mit Scikit-learn

```
import sklearn.neural_network as skl
import numpy as np
import onnx
from skl2onnx import convert_sklearn
from skl2onnx.common.data_types import FloatTensorType

def modify_onnx_MLPClassifier(onnx_MLPClassifier):
    """ Function to modify onnx model from MLPClassifier to make it suitable for TwinCAT Machine Lea
rning
    The function removes unsupported nodes and uses the probability estimates of the classes as outp
ut for
    the model.
    The output of the modified onnx model is the same as the output of the predict_proba() method fr
om
    sklearn.neural_network.MLPClassifier.
    To get the class as an integer output either a binarization or an argmax function must be applie
d to the
    model output.
    """
    # Delete nodes after last sigmoid/softmax layer
    output_node_types = {'Sigmoid', 'Softmax'}
    len_onx_init = len(onnx_MLPClassifier.graph.node)
    erased_node_inputs = []
    for idx, node in enumerate(reversed(onnx_MLPClassifier.graph.node)):
        if node.op_type in output_node_types:
            idx_last_node = len_onx_init - idx -1
            break
        else:
            for idx, input in enumerate(node.input):
```

```
                erased_node_inputs.append(input)
            onnx_MLPCLassifier.graph.node.remove(node)
    # Get output dimension and clean initializers
    second_last_node = onnx_MLPCLassifier.graph.node[idx_last_node-1]
    for initializer in onnx_MLPCLassifier.graph.initializer:
        if initializer.name in second_last_node.input:
            output_dim = initializer.dims[1]
        if initializer.name in erased_node_inputs:
            onnx_MLPCLassifier.graph.initializer.remove(initializer)
    # Erase original outputs and create new output
    for output in reversed(onnx_MLPCLassifier.graph.output):
        onnx_MLPCLassifier.graph.output.remove(output)
    name_new_output = "probabilities"
    newOutput = onnx.helper.make_tensor_value_info(name_new_output, onnx.TensorProto.FLOAT, shape=(N
one, output_dim))
    onnx_MLPCLassifier.graph.output.append(newOutput)
    # Create new node and connect to new output
    last_node_output = onnx_MLPCLassifier.graph.node[idx_last_node].output
    new_node = onnx.helper.make_node("Identity", last_node_output, [name_new_output], "Identity_Out"
)
    onnx_MLPCLassifier.graph.node.append(new_node)
    return onnx_MLPCLassifier

input_dim = 3
output_dim = 2
n_samples = 100
dummy_input = np.random.random((n_samples,input_dim))
dummy_output = np.random.randint(2, size=(n_samples, output_dim))
model = skl.MLPClassifier(activation ='relu', hidden_layer_sizes=(5))
model.fit(dummy_input,dummy_output)
filename = 'skl_mlp_clf'
initial_type = [('float_input',FloatTensorType([None,input_dim]))]
onx = convert_sklearn(model,initial_types=initial_type, options={type(model): {'zipmap': False}})
onx = modify_onnx_MLPClassifier(onx)
with open(filename+'.onnx','wb') as f:
    f.write(onx.SerializeToString())
```



Beachten Sie die Funktion modify_onnx_MLPClassifier. Diese modifiziert den unteren Teil des ONNX-Graphs, sodass nur Operatoren genutzt werden, die von der TwinCAT Neural Network Inference Engine unterstützt werden. Ohne diese Modifikation werden die hier abgebildeten Operatoren (je nach Dimensionalität des Problems) erzeugt. Nur der grün umrahmte Bereich wird unterstützt.



5.1.2 Stützvektormaschine

Eine Stützvektormaschine (SVM) kann sowohl zur Klassifikation als auch zur Regression [▶ 61] genutzt werden. Gerade hinsichtlich Klassifikationsaufgaben ist die SVM ein häufig eingesetztes Werkzeug.

Grundsätzlich ist das Ziel einer SVM eine Hyperebene in einem N-dimensionalen Raum zu finden, wobei der Abstand des nahegelegensten Datenpunktes zur Ebene maximiert wird. Eine Hyperebene kann den Raum nur linear separieren (auch lineare SVM genannt). Durch den sogenannten Kernel-Trick wird auch eine nichtlineare Trennung möglich (auch Kernel-SVM genannt). Hierbei wird der N-dimensionale Raum in einen höherdimensionalen Raum transformiert. In einem entsprechend hochdimensionalen Raum ist eine lineare Trennung mit einer Hyperebene möglich.

Wenn mehrere Klassen zu unterscheiden sind, werden intern mehrere Stützvektormaschinen erzeugt und es erfolgt eine Einordnung durch Vergleiche. Ebenso ist eine One-Class-SVM trainerbar, welche zur Anomalie-Erkennung eingesetzt werden kann.

Unterstützte Eigenschaften

ONNX-Support

Unterstützt werden die ONNX-Operatoren:

- SVMRegressor

- SVMClassifier

Unterstützte Kernel-Funktionen sind in der folgenden Tabelle aufgeführt:

Kernel-Funktion	Beschreibung
Linear	$K(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}_1^{\mathrm{T}}\mathbf{x}_2$
Radial Basis Function (RBF)	$K(\mathbf{x}_1\mathbf{x}_2) = \exp(-\gamma \left\|\mathbf{x}_1 - \mathbf{x}_2\right\|^2)$
Sigmoid	$K(\mathbf{x}_1, \mathbf{x}_2) = \tanh(a\mathbf{x}_1^{\mathrm{T}}\mathbf{x}_2 + b)$
Polynomial	$K(\mathbf{x}_1, \mathbf{x}_2) = (\mathbf{x}_1^{\mathrm{T}}\mathbf{x}_2 + 1)^d$

Beispiele für den Export von SVMs als ONNX, siehe ONNX Export einer SVM [▶ 29].

Einschränkung bei Klassifikation

Bei Klassifikationsmodellen wird nur der Ausgang der Labels in der SPS abgebildet. Die Scores/ Probabilities sind in der SPS nicht verfügbar.

Unterstützte Datentypen

Es ist zu unterscheiden zwischen „supported datatype“ und „preferred datatype“. Der preferred datatype entspricht der Präzision der Ausführungs-Engine.

Der preferred datatype ist floating point 64 (E_MLLDT_FP64-LREAL).

Bei Verwendung eines supported datatype wird eine effiziente Typ-Konvertierung automatisch in der Bibliothek durchgeführt. Durch die Typ-Konvertierung kann es zu minimalen Performance-Einbußen kommen.

Eine Liste der supported datatypes finden Sie unter ETcMlIDataType [▶ 83].

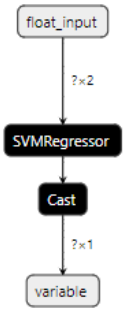
5.1.2.1 ONNX-Export einer SVM

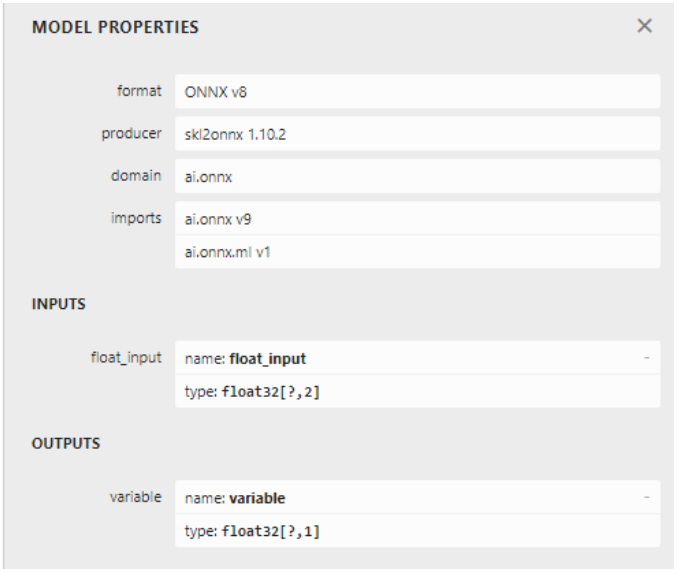
Download Python Samples

Ein Zip-Archiv mit allen Samples finden Sie hier: Beispiele zum ONNX-Export [▶ 63]

SVM Regressor mit Scikit-learn

```
from sklearn.svm import SVR
from skl2onnx import convert_sklearn
from skl2onnx.common.data_types import FloatTensorType


X = [[13,3],[1,16],[1,2]]
Y = [1.0,2.0,3.0]

model = SVR(kernel='rbf',gamma=10)
model.fit(X,Y)

out = model.predict(X)

input_type = [('float_input', FloatTensorType([None,len(X[0])]))]

onnx_filename = 'svr-rbf.onnx'
onx = convert_sklearn(model,initial_types=input_type)
with open(onnx_filename,"wb") as f:
    f.write(onx.SerializeToString())
```

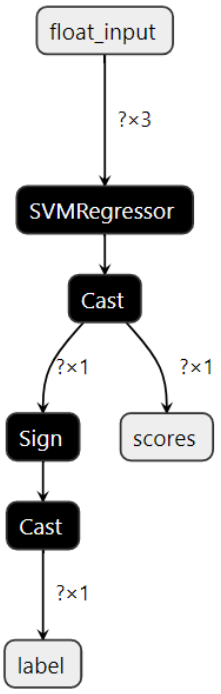


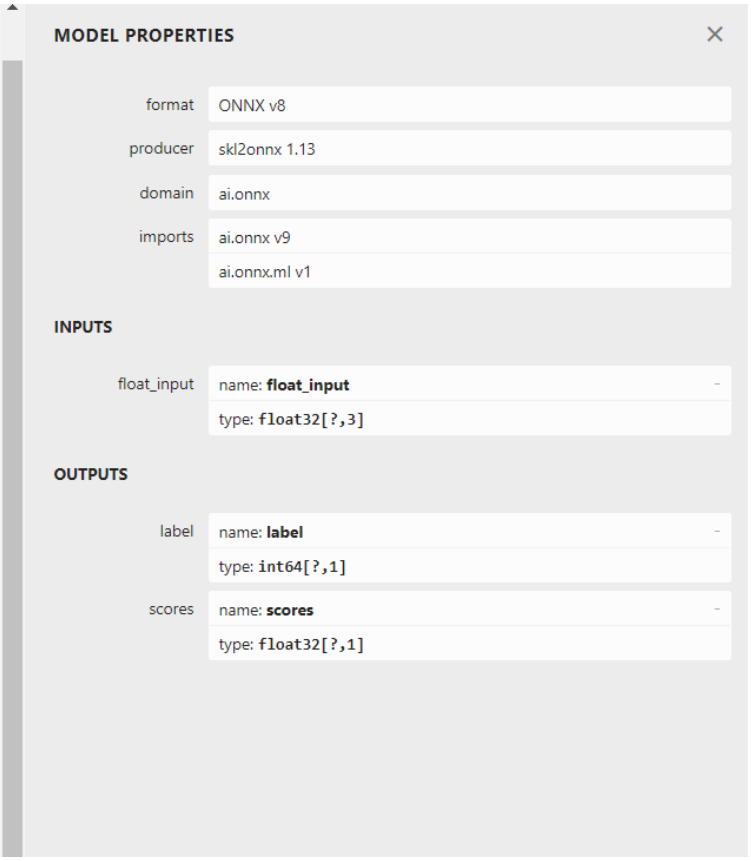


SVM Classifier mit Scikit-learn

One-Class-SVM

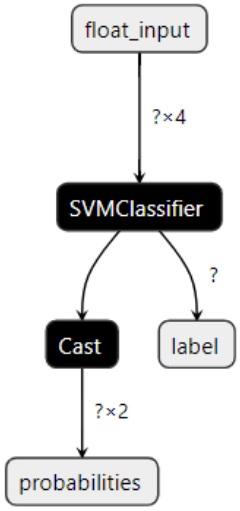
```
from sklearn.datasets import make_blobs
from sklearn.svm import OneClassSVM
from skl2onnx import convert_sklearn
from skl2onnx.common.data_types import FloatTensorType

n_samples = 150
input_dim = 3

X, _ = make_blobs(n_samples=n_samples, n_features=input_dim, centers=1, cluster_std=0.3, shuffle=Tru
e, random_state=42, )
svm = OneClassSVM(kernel='rbf', nu=0.3)
svm.fit(X)
initial_type = [('float_input', FloatTensorType([None, X.shape[1]]))]
onnx_model = convert_sklearn(svm, initial_types=initial_type)
filename = 'skl_oneclass_svm'
with open(filename + '.onnx', "wb") as f:
    f.write( onnx_model.SerializeToString())
f.close()
```

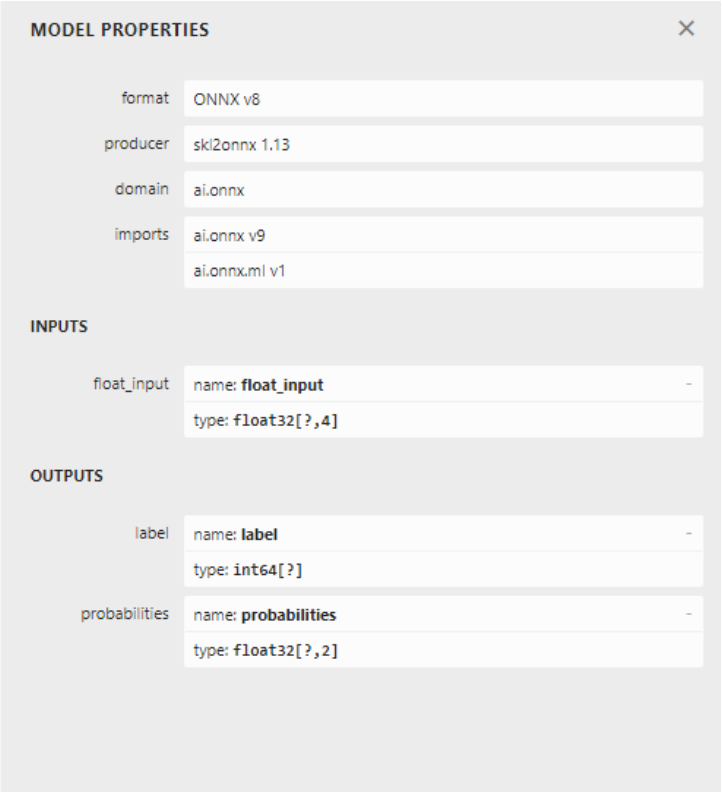




Binary Classification

```
from sklearn.svm import SVC
import numpy as np

# random dataset
n_samples = 150
input_dim = 4
n_classes = 2 # binary classification

rand_in = np.random.random((n_samples, input_dim,))
rand_singleout_multiclass = np.random.randint(n_classes, size=(n_samples, 1))

X = rand_in
y = rand_singleout_multiclass

# train SVC
clr = SVC(kernel='linear', gamma=10) # decision_function_shape option is ignored for binary classifi
cation
clr.fit(X, y)

# Convert into ONNX format
from skl2onnx import convert_sklearn
from skl2onnx.common.data_types import FloatTensorType

initial_type = [('float_input', FloatTensorType([None, X.shape[1]]))]

# # Zipmap should be always turned off as it's not implemented in TF3800
onx = convert_sklearn(clr, initial_types=initial_type, options={type(clr): {'zipmap':False}})
with open("svc_random.onnx", "wb") as f:
    f.write(onx.SerializeToString())
```





Multi-class Classification

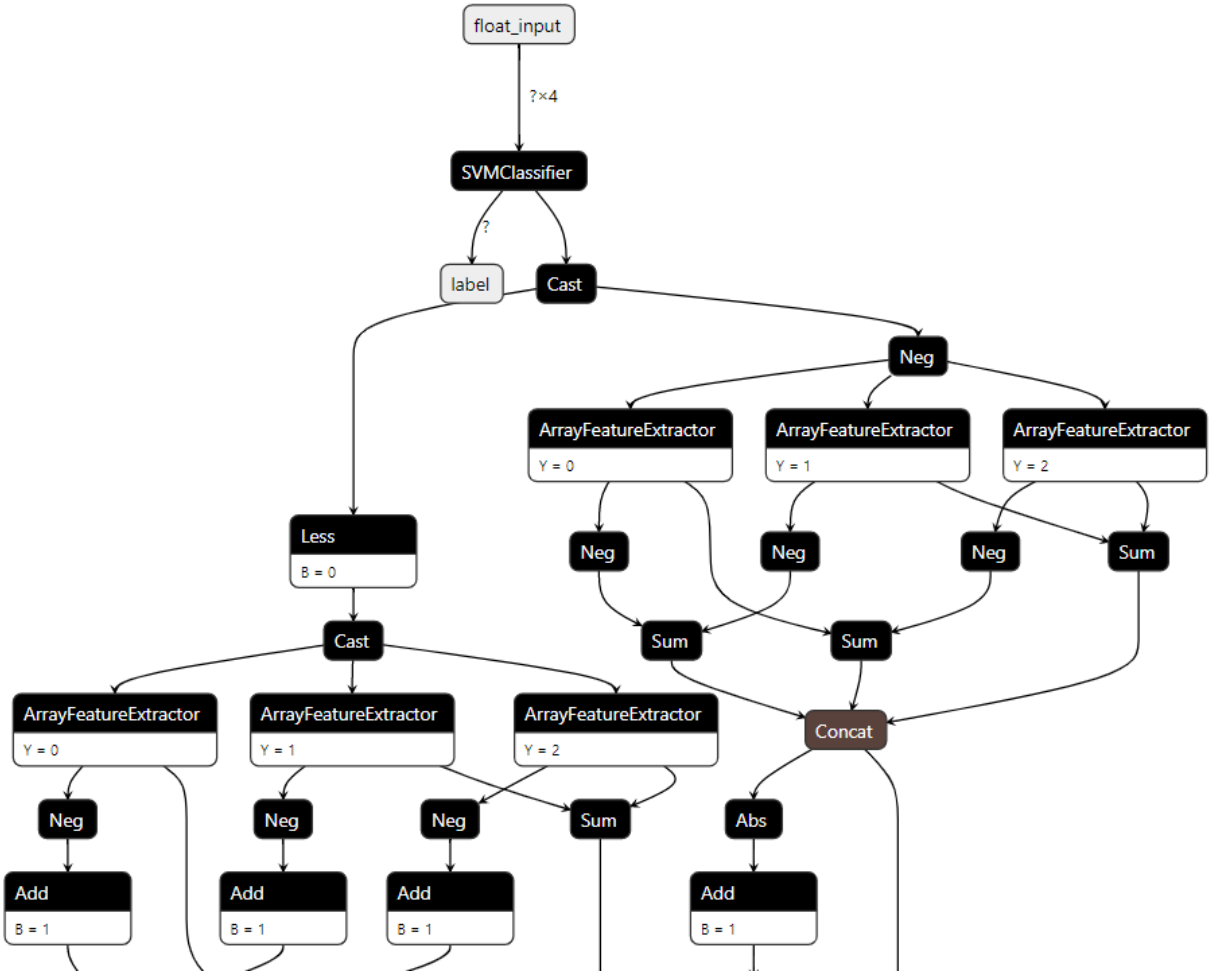
```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC

# data set
iris = load_iris()
X, y = iris.data, iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y)

# train SVC, note that decision_function_shape ovo is mandatory
clr = SVC(kernel='linear', gamma=10,decision_function_shape='ovo')
clr.fit(X_train, y_train)

# Convert into ONNX format
from skl2onnx import convert_sklearn
from skl2onnx.common.data_types import FloatTensorType

initial_type = [('float_input', FloatTensorType([None, 4]))]

onx = convert_sklearn(clr, initial_types=initial_type)
with open("svc_iris.onnx", "wb") as f:
    f.write(onx.SerializeToString())
```

Form des ONNX-Graphen beachten!

Bei Multi-Class-SVC-Modellen muss die Option decision_function_shape “ovo” genutzt werden, damit ein zu TF3800 kompatibler ONNX-Graph erzeugt wird.

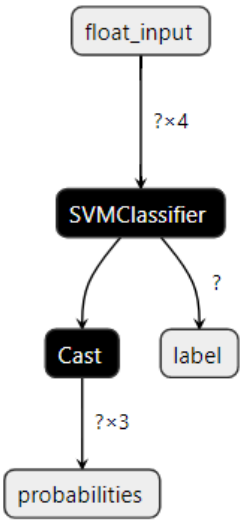
Nicht gültiger ONNX-Graph

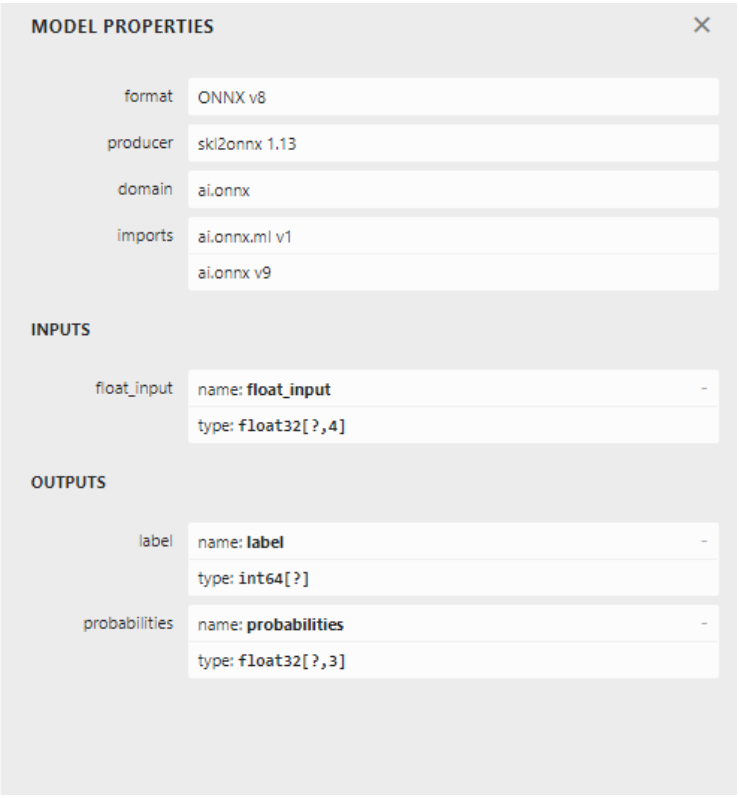
Das folgende Beispiel zeigt den **nicht** gültigen ONNX-Graph: decision_function_shape “ovr”:



Gültiger ONNX-Graph

Das folgende Beispiel zeigt den **gültigen** ONNX-Graph: decision_function_shape “ovo”:





5.1.3 k-Means

Der k-Means-Algorithmus gehört zu den *unsupervised* Lernmethoden und wird zur Clusteranalyse [▶ 61] verwendet. k-Means versucht, eine Stichprobe in k-Cluster gleicher Varianz aufzuteilen, wobei die Anzahl der Cluster k vorab bekannt sein muss. Der Algorithmus lässt sich gut auf eine große Anzahl von Stichproben skalieren und ist einer der am häufigsten eingesetzten Clustering-Verfahren.

Unsupervised heißt, dass der k-Means nicht mit annotierten (gelabelten) Daten trainiert werden muss. Diese Eigenschaft macht den Algorithmus sehr populär. Sobald das Training ausgeführt wurde und die Cluster somit definiert sind, können in der Inferenz neue Daten den bereits bekannten Clustern zugeordnet werden.

Unterstützte Eigenschaften

ONNX-Support

Bislang wird nur der Export aus Scikit-learn unterstützt. Die Angabe des ONNX-Custom Attributes Key: „sklearn_model“ value: „KMeans“ ist für k-Means-Modelle notwendig, damit der Konvertierungsschritt in XML und BML funktioniert.

Einschränkung

Bei Klassifikationsmodellen wird nur der Ausgang der Labels in der SPS abgebildet. Die Scores/ Probabilities sind in der SPS nicht verfügbar.

Ein Beispiel für den Export einer ONNX-Datei aus Scikit-learn zur Nutzung in TwinCAT finden Sie hier: ONNX-Export eines k-Means [▶ 34].

Unterstützte Datentypen

Es ist zu unterscheiden zwischen „supported datatype“ und „preferred datatype“. Der preferred datatype entspricht der Präzision der Ausführungs-Engine.

Der preferred datatype ist floating point 64 (E_MLLDT_FP64-LREAL).

Bei Verwendung eines supported datatype wird eine effiziente Typ-Konvertierung automatisch in der Bibliothek durchgeführt. Durch die Typ-Konvertierung kann es zu minimalen Performance-Einbußen kommen.

Eine Liste der supported datatypes finden Sie unter ETcMllDataType [▶ 83].

5.1.3.1 ONNX-Export eines k-Means

Download Python Samples

Ein Zip-Archiv mit allen Samples finden Sie hier: Beispiele zum ONNX-Export [▶ 63]

k-Means with Scikit-learn

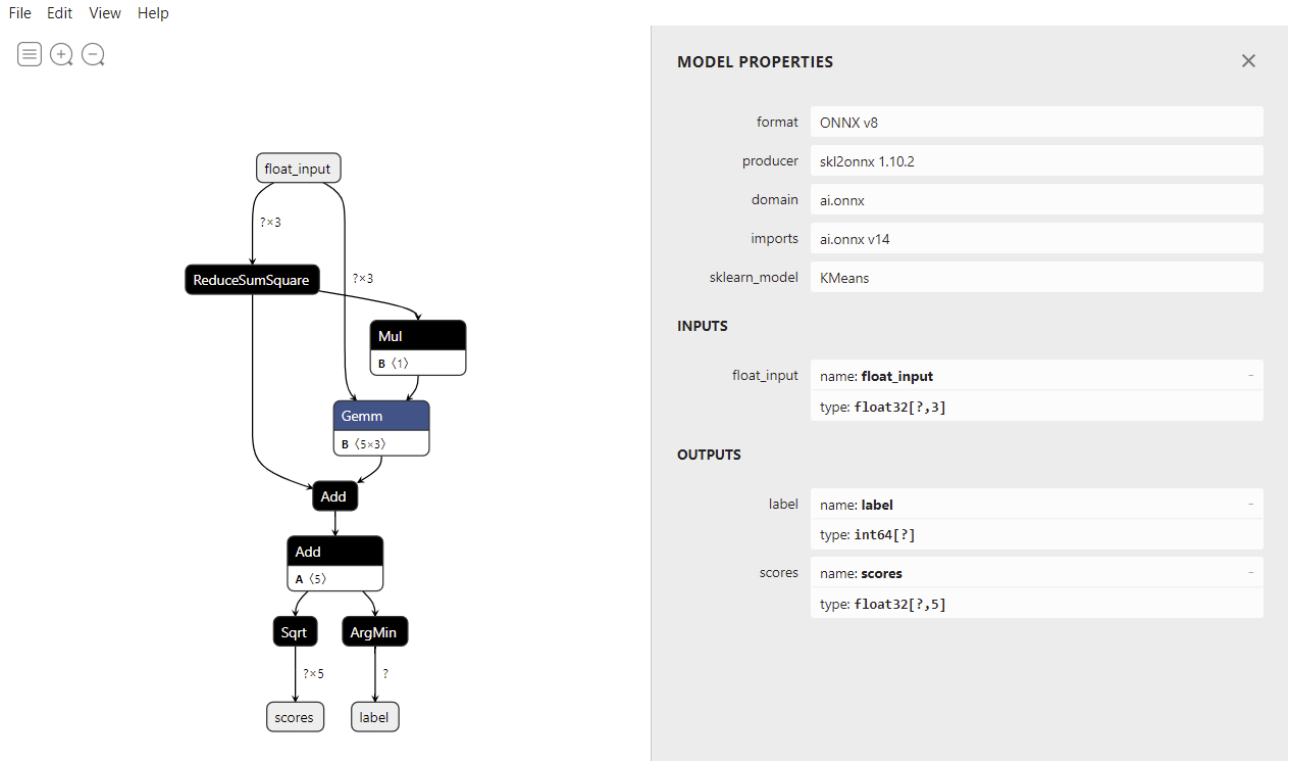
```
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
from skl2onnx import convert_sklearn
from skl2onnx.common.data_types import FloatTensorType

# Generate data for clustering: num of samples, dimensions, num of clusters
X, y, centers = make_blobs(n_samples=50, n_features=3, centers=5, cluster_std=0.5, shuffle=True, ran
dom_state=42,  return_centers=True)
num_features = X.shape[1]
num_clusters = centers.shape[0]

# Define and train K-Means model
km = KMeans(n_clusters=num_clusters, init='k-means+
+', n_init=10, max_iter=500, tol=1e-04, random_state=42)
km.fit(X)

y_km = km.predict(X)

# Export model as ONNX
out_onnx = "kmeans.onnx"
```

```
initial_type = [('float_input', FloatTensorType([None, num_features]))]
onnx_model = convert_sklearn(km, initial_types=initial_type)

# for k-means models this meta info is mandatory. Otherwise convert process to TwinCAT-
specific format will fail!
meta =  onnx_model.metadata_props.add()
meta.key = "sklearn_model"
meta.value = "KMeans"

with open(out_onnx, "wb") as f:
    f.write( onnx_model.SerializeToString())
```

Nach unserem Kenntnisstand ist aktuell nur Scikit-learn mit skl2onnx in der Lage, einen k-Means in ONNX zu konvertieren. Aus diesem Grund beschränkt sich die Beschreibung darauf.

ONNX Custom Attribute notwendig

Die Angabe des Custom Attributes „sklearn_model“ und „KMeans“ ist für k-means Modelle notwendig, damit der Konvertierungsschritt in Beckhoff XML und Beckhoff BML funktioniert.



5.1.4 Principal Component Analysis

Die Principal Component Analysis (PCA) berechnet sogenannte principal components, mit deren Hilfe man das Koordinatensystem eines gegebenen Datensatzes so drehen kann, dass entlang der neuen Hauptachsen die Varianz der Daten, d. h. deren Informationsgehalt, maximiert wird. Die Kovarianzmatrix der transformierten Daten wird diagonalisiert und die Reihenfolge der Hauptkomponenten wird so sortiert, dass die erste Hauptkomponente den größten Informationsanteil des Datensatzes, die zweite dann den zweitgrößten Informationsanteil usw. trägt.

Die letzteren Hauptkomponenten tragen oft nur noch wenig Information zum Datensatz bei, was dazu führt, dass man diese ignorieren kann. Dadurch reduziert sich der Parameterraum bei möglichst wenig Informationsverlust (Dimensionsreduktion [▶ 61]). Die Dimensionsreduktion mit PCA wird häufig als Vorverarbeitung vor der Clusteranalyse oder einer Klassifikation eingesetzt. Ebenso kann die PCA zur Anomaly detection [▶ 61] eingesetzte werden, indem ein großer Raum auf wenige Hauptkomponenten reduziert wird und „von Hand“ Grenzwerte für die wichtigen Hauptkomponenten ermittelt werden.

Unterstützte Eigenschaften

ONNX-Support

Unterstützte ONNX-Operatoren:

- Sub
- MatMul

Ein Beispiel, wie eine PCA zur Dimensionsreduktion aus Scikit-learn exportiert und in TwinCAT eingesetzt werden kann, finden Sie hier: ONNX-Export einer PCA [▶ 36].

Unterstützte Datentypen

Es ist zu unterscheiden zwischen „supported datatype“ und „preferred datatype“. Der preferred datatype entspricht der Präzision der Ausführungs-Engine.

Der preferred datatype ist floating point 64 (E_MLLDT_FP64-LREAL).

Bei Verwendung eines supported datatype wird eine effiziente Typ-Konvertierung automatisch in der Bibliothek durchgeführt. Durch die Typ-Konvertierung kann es zu minimalen Performance-Einbußen kommen.

Eine Liste der supported datatypes finden Sie unter ETcMllDataType [▶ 83].

5.1.4.1 ONNX-Export einer PCA

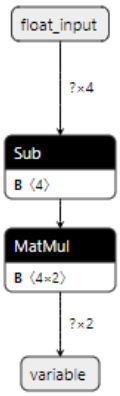
Download Python Samples

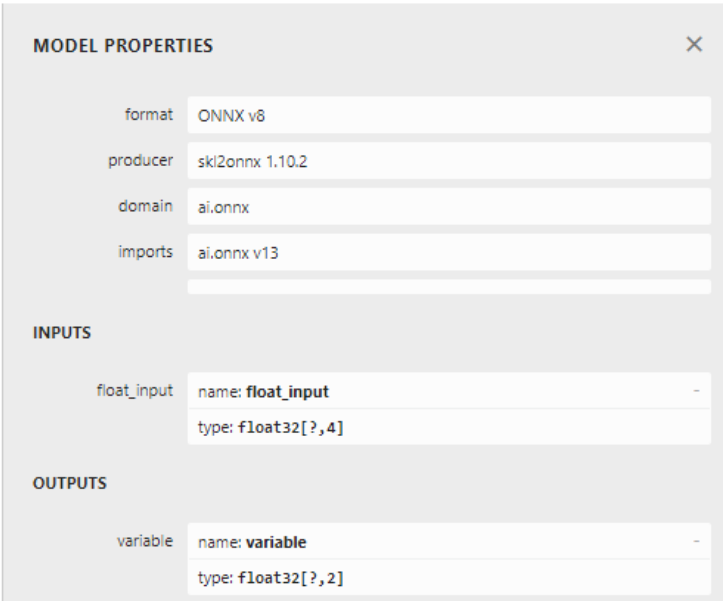
Ein Zip-Archiv mit allen Samples finden Sie hier: Beispiele zum ONNX-Export [▶ 63]

PCA mit Scikit-learn

```
import numpy as np
from sklearn.decomposition import PCA
from skl2onnx import convert_sklearn
from skl2onnx.common.data_types import FloatTensorType, Int64TensorType

#Generate imput data_types
rng = np.random.RandomState(1)
X = np.dot(rng.rand(4, 3), rng.randn(3, 300)).T

# Dimensionality reduction with PCA
pca = PCA(n_components=2)
pca.fit(X)
X_pca = pca.transform(X)
print("original shape:   ", X.shape)
print("transformed shape:", X_pca.shape)

#Convert model to ONNX
initial_type = [('float_input', FloatTensorType([None, X.shape[1]]))]
model_onnx = convert_sklearn(pca, initial_types=initial_type)
meta =  model_onnx.metadata_props.add()
with open("pca.onnx", "wb") as f:
    f.write( model_onnx.SerializeToString())
```

Nach unserem Kenntnisstand ist aktuell nur Scikit-learn mit skl2onnx in der Lage eine PCA in ONNX zu konvertieren. Aus diesem Grund beschränkt sich die Beschreibung darauf.





5.1.5 Decision Tree

Ein Decision Tree (Entscheidungsbaum) ist ein ML-Modell, das eine baumähnliche Struktur verwendet, um Vorhersagen zu treffen. Es handelt sich um ein einfaches, aber leistungsfähiges Werkzeug zur Vorhersage von Werten (Regression) oder Klassen (Klassifikation) [▶ 61] auf der Grundlage mehrerer Eingaben, indem die Daten in immer kleinere Teilmengen unterteilt werden, bis eine endgültige Entscheidung getroffen wird. Die Struktur des Baums ermöglicht eine einfache Interpretation und Visualisierung des Modells.

Unterstützte Eigenschaften

ONNX-Support

- TreeEnsambleClassifier
- TreeEnsambleRegressor

Beispiele für den Export von Decision Tree-Modellen finden Sie hier: ONNX-Export eines Decision Tree [▶ 37].

Einschränkung bei Klassifikation

Bei Klassifikationsmodellen wird nur der Ausgang der Labels in der SPS abgebildet. Die Scores/ Probabilities sind in der SPS nicht verfügbar.

Unterstützte Datentypen

Es ist zu unterscheiden zwischen „supported datatype“ und „preferred datatype“. Der preferred datatype entspricht der Präzision der Ausführungs-Engine.

Der preferred datatype ist floating point 64 (E_MLLDT_FP64-LREAL).

Bei Verwendung eines supported datatype wird eine effiziente Typ-Konvertierung automatisch in der Bibliothek durchgeführt. Durch die Typ-Konvertierung kann es zu minimalen Performance-Einbußen kommen.

Eine Liste der supported datatypes finden Sie unter ETcMlIDataType [▶ 83].

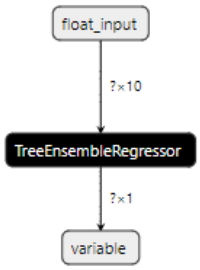
5.1.5.1 ONNX-Export eines Decision Tree

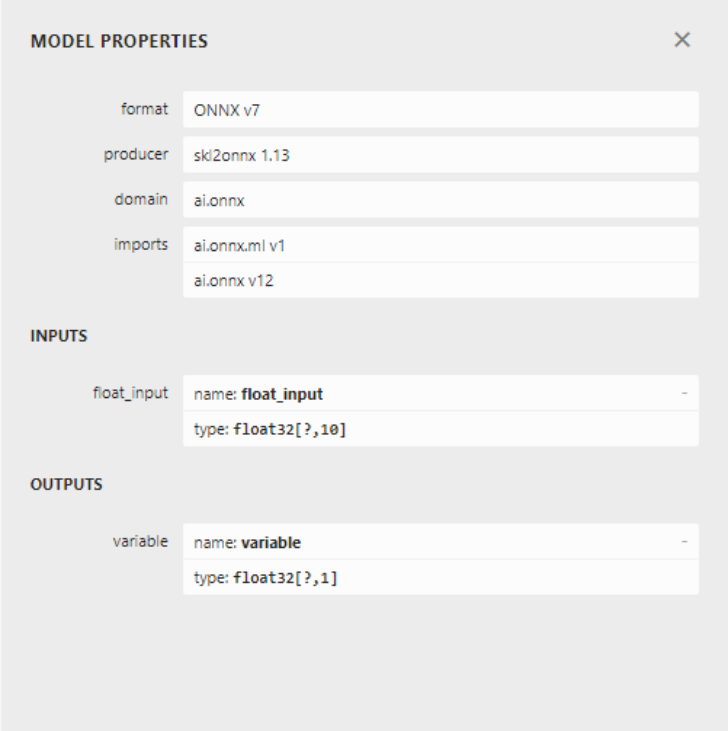
Download Python Samples

Ein Zip-Archiv mit allen Samples finden Sie hier: Beispiele zum ONNX-Export [▶ 63]

Decision Tree Regressor mit Scikit-learn

```
from sklearn.datasets import make_regression
from sklearn.tree import DecisionTreeRegressor
from skl2onnx import convert_sklearn
from skl2onnx.common.data_types import FloatTensorType

# # Generate data for regression
X, y =  make_regression(n_samples=300, n_features=10, n_informative=10, n_targets=1)

# # Construct Decision Tree-Model
model = DecisionTreeRegressor(criterion='squared_error', splitter='best', max_depth=None, min_sample
s_split=2, min_samples_leaf=1)
model.fit(X,y)
# # Convert model to ONNX
onnxfile = 'decisiontree-regressor.onnx'
initial_type = [('float_input', FloatTensorType([None, X.shape[1]]))]
onnx_model = convert_sklearn(model, initial_types=initial_type, target_opset=12)
# # Export to ONNX file
with open(onnxfile, "wb") as f:
    f.write( onnx_model.SerializeToString())
f.close()
exit()
```

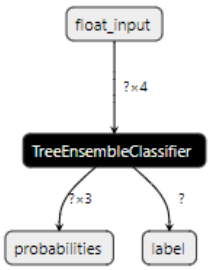


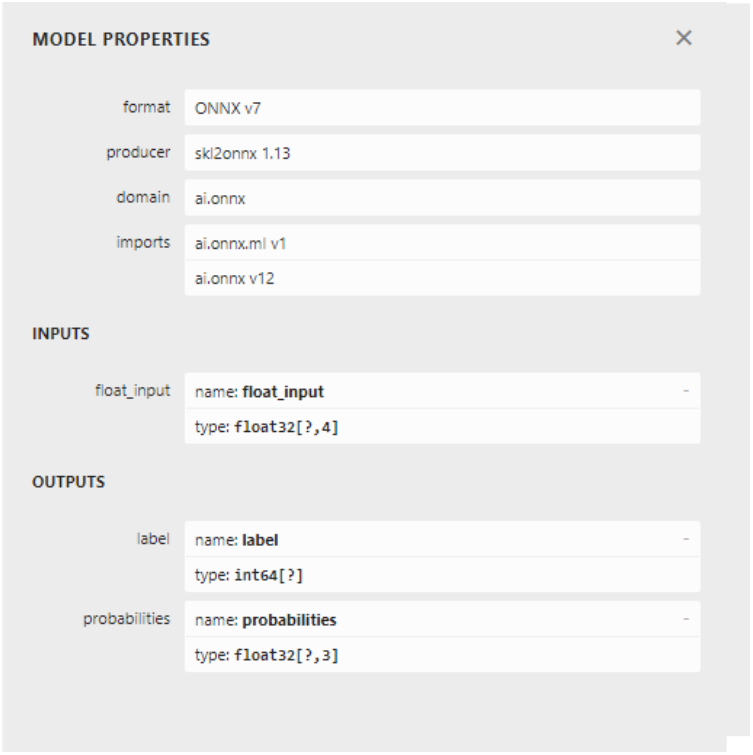


Decision Tree Classifier mit Scikit-learn

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from skl2onnx import convert_sklearn
from skl2onnx.common.data_types import FloatTensorType

# # Load dataset
X, y = load_iris(return_X_y = True)
# # Construct Decision Tree-Model
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1)
model = DecisionTreeClassifier(criterion='gini', splitter='best', max_depth=None, min_samples_split=
2, min_samples_leaf=1)
model.fit(X_train,y_train)
# # Convert model to ONNX
onnxfile = 'decisiontree-classifier.onnx'
initial_type = [('float_input', FloatTensorType([None, X.shape[1]]))]
# # Zipmap should be always turned off as it's not implemented in TF3800
onnx_model = convert_sklearn(model, initial_types=initial_type, options={type(model): {'zipmap':Fals
e}}, target_opset=12)
# # Export to ONNX file
with open(onnxfile, "wb") as f:
    f.write( onnx_model.SerializeToString())
```

```
f.close()
exit()
```





5.1.6 ExtraTree

Ein Extra Tree ist die randomisierte Variante eines Decision Tree [▶ 37]. Er kann ebenfalls zur Vorhersage von Werten (Regression) oder Klassen (Klassifikation) [▶ 61] eingesetzt werden.

Unterstützte Eigenschaften

ONNX-Support

- TreeEnsambleClassifier
- TreeEnsambleRegressor

Beispiele für den Export von Extra Tree-Modellen finden Sie hier: ONNX-Export eines Extra Tree [▶ 40]

Einschränkung bei Klassifikation

Bei Klassifikationsmodellen wird nur der Ausgang der Labels in der SPS abgebildet. Die Scores/ Probabilities sind in der SPS nicht verfügbar.

Unterstützte Datentypen

Es ist zu unterscheiden zwischen „supported datatype“ und „preferred datatype“. Der preferred datatype entspricht der Präzision der Ausführungs-Engine.

Der preferred datatype ist floating point 64 (E_MLLDT_FP64-LREAL).

Bei Verwendung eines supported datatype wird eine effiziente Typ-Konvertierung automatisch in der Bibliothek durchgeführt. Durch die Typ-Konvertierung kann es zu minimalen Performance-Einbußen kommen.

Eine Liste der supported datatypes finden Sie unter ETcMllDataType [▶ 83].

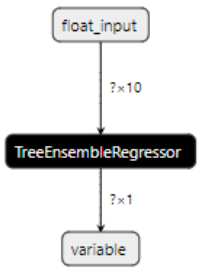
5.1.6.1 ONNX-Export eines Extra Tree

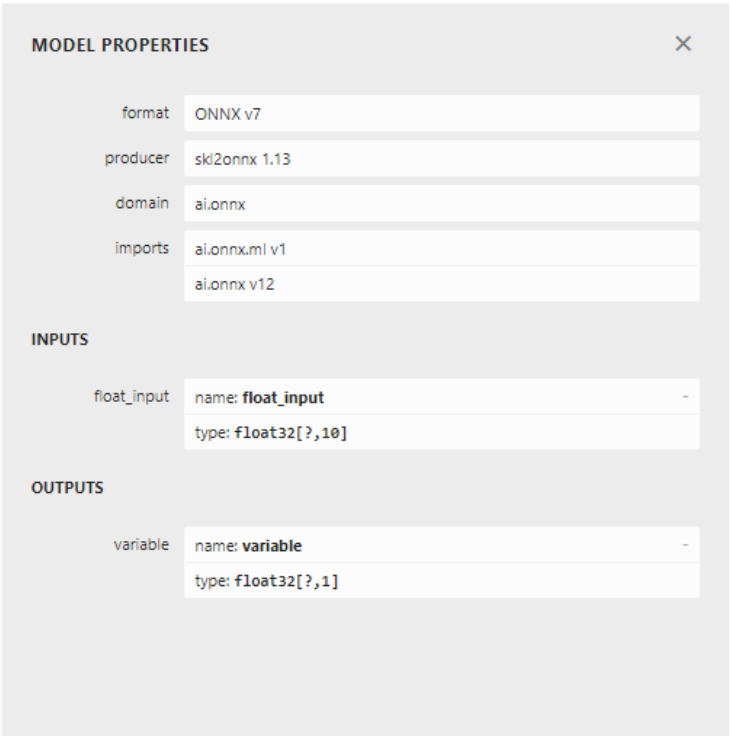
Download Python Samples

Ein Zip-Archiv mit allen Samples finden Sie hier: Beispiele zum ONNX-Export [▶ 63]

Extra Tree Regressor mit Scikit-learn

```
from sklearn.datasets import make_regression
from sklearn.tree import ExtraTreeRegressor
from skl2onnx import convert_sklearn
from skl2onnx.common.data_types import FloatTensorType

# # Generate data for regression
X, y =  make_regression(n_samples=300, n_features=10, n_informative=10, n_targets=1)

# # Construct Extra Tree-Model
model = ExtraTreeRegressor(criterion='squared_error', splitter='random', max_depth=None, min_samples
_split=2, min_samples_leaf=1)
model.fit(X,y)
# # Convert model to ONNX
onnxfile = 'extratree-regressor.onnx'
initial_type = [('float_input', FloatTensorType([None, X.shape[1]]))]
onnx_model = convert_sklearn(model, initial_types=initial_type, target_opset=12)
# # Export to ONNX file
with open(onnxfile, "wb") as f:
    f.write( onnx_model.SerializeToString())
f.close()
exit()
```

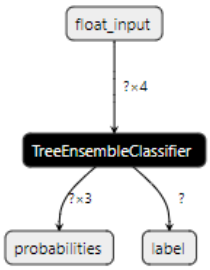


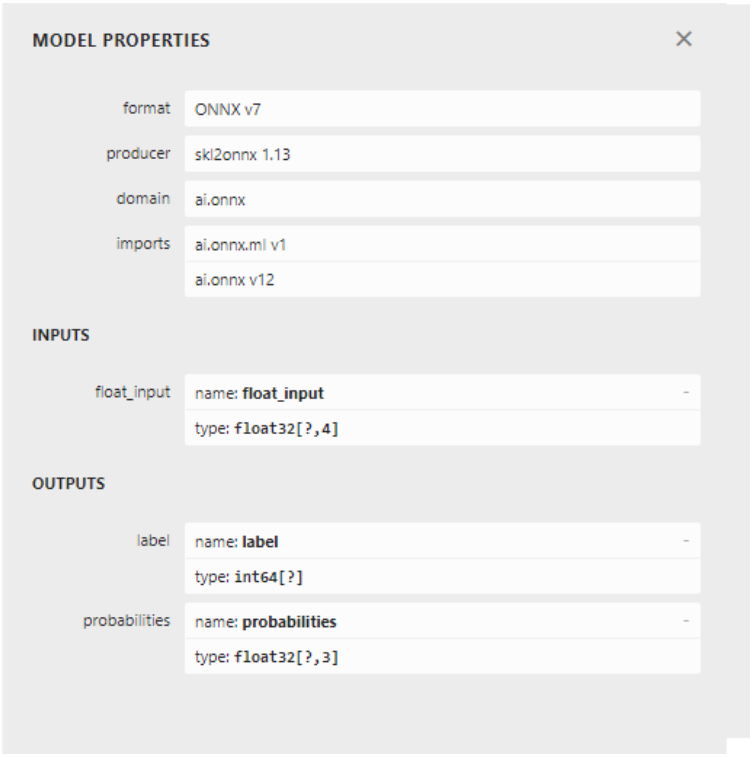


Extra Tree Classifier mit Scikit-learn

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import ExtraTreeClassifier
from skl2onnx import convert_sklearn
from skl2onnx.common.data_types import FloatTensorType

# # Load dataset
X, y = load_iris(return_X_y = True)
# # Construct Decision Tree-Model
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1)
model = ExtraTreeClassifier(criterion='gini', splitter='random', max_depth=None, min_samples_split=2
, min_samples_leaf=1)
model.fit(X_train,y_train)
```

```
# # Convert model to ONNX
onnxfile = 'extratree-classifier.onnx'
initial_type = [('float_input', FloatTensorType([None, X.shape[1]]))]
# # Zipmap should be always turned off as it's not implemented in TF3800
onnx_model = convert_sklearn(model, initial_types=initial_type, options={type(model): {'zipmap':Fals
e}}, target_opset=12)
# # Export to ONNX file
with open(onnxfile, "wb") as f:
    f.write( onnx_model.SerializeToString())
f.close()
exit()
```





5.1.7 Ensemble Tree Methoden

Ensemble Methoden kombinieren mehrere Decision Trees, um eine bessere Prädiktions-Performance zu erreichen. Das Grundprinzip ist, nicht nur ein Modell (einen Baum) zu trainieren, sondern mehrere Bäume -einen Wald von Bäumen- anzulernen und einzelne Ergebnisse zu einem gemeinsamen Ergebnis zu kombinieren.

Es existieren grundsätzlich zwei Technologien, um ein Ensemble von Trees zu erstellen.

Bagging

Zu den Bagging-Methoden gehören:

- Random Forest [▶ 44]
- Extra Trees [▶ 42]

Boosting

Zu den Boosting-Methoden gehören:

- Gradient Boosting [▶ 47]
- Histogram Gradient Boosting [▶ 49]
- XGBoost [▶ 52]
- LightGBM [▶ 55]

Nicht unterstützte weitere Ensemble Tree Methoden

In Scikit-learn stehen auch die Modelle BaggingClassifier, BaggingRegressor, AdaBoostClassifier und AdaBoostRegressor zur Verfügung. Diese erzeugen aktuell beim ONNX-Export einen zur TwinCAT-Bibliothek inkompatiblen Graphen, sodass sie nicht unterstützt werden können.

5.1.7.1 ExtraTrees

Extra Trees erstellt ein Ensemble von randomisierten Decision Trees [▶ 39]. Jeder Baum wird auf einer Submenge des Datensatzes trainiert und die Teilergebnisse gemittelt. Dadurch wird die Accuracy der Prädiktion im Vergleich zum Decision Tree [▶ 37] erhöht und die Neigung zum Overfitting verringert.

Unterstützte Eigenschaften

ONNX-Support

- TreeEnsambleClassifier
- TreeEnsambleRegressor

Beispiele für den Export von Extra Trees finden Sie hier: ONNX-Export von Extra Trees [▶ 42].

Einschränkung bei Klassifikation

Bei Klassifikationsmodellen wird nur der Ausgang der Labels in der SPS abgebildet. Die Scores/ Probabilities sind in der SPS nicht verfügbar.

Unterstützte Datentypen

Es ist zu unterscheiden zwischen „supported datatype“ und „preferred datatype“. Der preferred datatype entspricht der Präzision der Ausführungs-Engine.

Der preferred datatype ist floating point 64 (E_MLLDT_FP64-LREAL).

Bei Verwendung eines supported datatype wird eine effiziente Typ-Konvertierung automatisch in der Bibliothek durchgeführt. Durch die Typ-Konvertierung kann es zu minimalen Performance-Einbußen kommen.

Eine Liste der supported datatypes finden Sie unter ETcMlIDataType [▶ 83].

5.1.7.1.1 ONNX-Export von Extra Trees

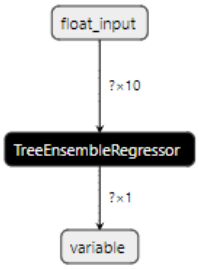
Download Python Samples

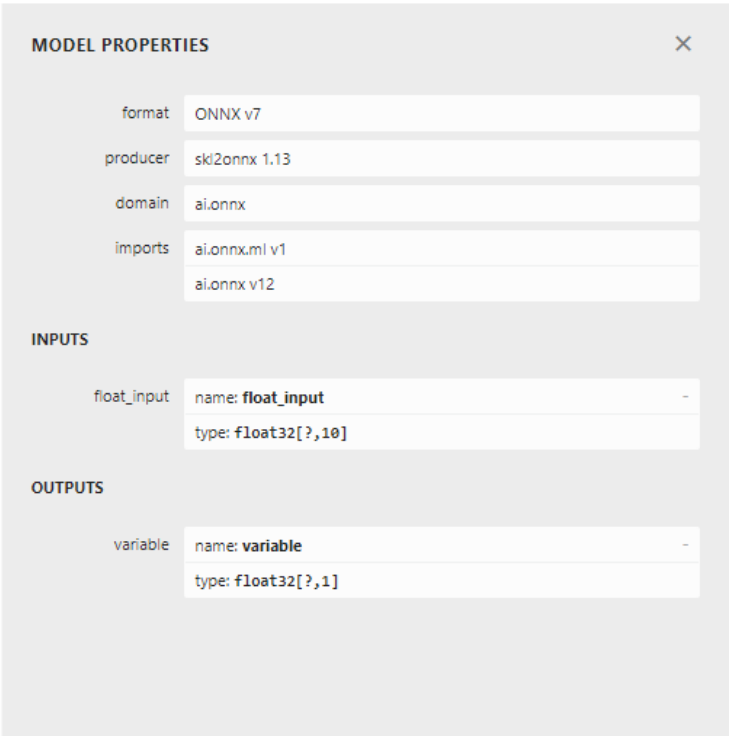
Ein Zip-Archiv mit allen Samples finden Sie hier: Beispiele zum ONNX-Export [▶ 63]

Extra Trees Regressor mit Scikit-learn

```
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.datasets import make_regression
from skl2onnx import convert_sklearn
from skl2onnx.common.data_types import FloatTensorType
# import onnx

# # Generate data for regression
X, y =  make_regression(n_samples=300, n_features=10, n_informative=10, n_targets=1)
# # Construct the model
model = ExtraTreesRegressor(max_depth=None, n_estimators=100)
model.fit(X,y)

# # Convert model to ONNX
onnxfile = 'extratrees-regressor.onnx'
initial_type = [('float_input', FloatTensorType([None, X.shape[1]]))]
onnx_model = convert_sklearn(model, initial_types=initial_type, target_opset=12)
# # Export to ONNX file
# onnx.checker.check_model(onnx_model)
with open(onnxfile, "wb") as f:
    f.write( onnx_model.SerializeToString())
```

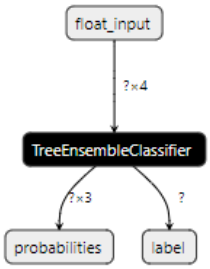
```
f.close()
exit()
```

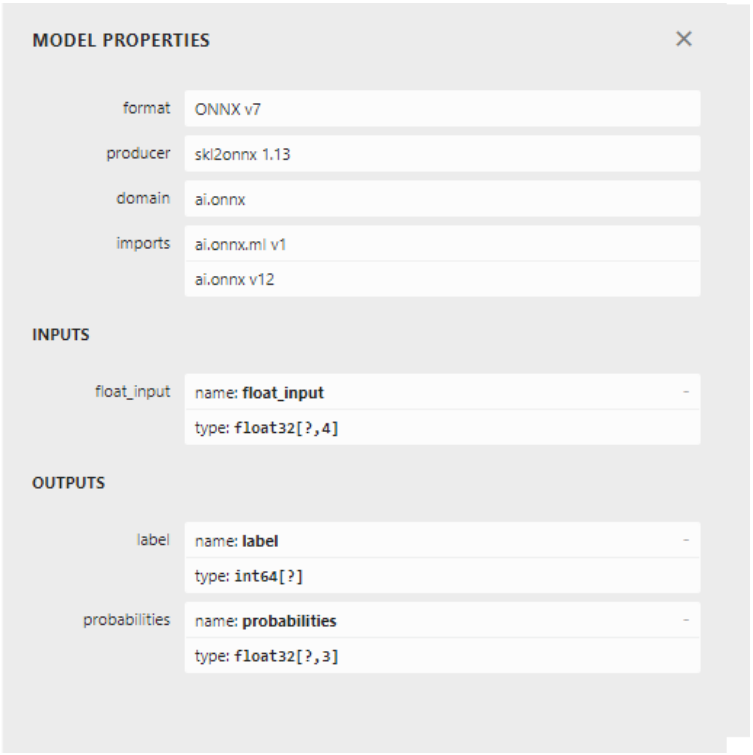




Extra Trees Classifier mit Scikit-learn

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import ExtraTreesClassifier
from skl2onnx import convert_sklearn
from skl2onnx.common.data_types import FloatTensorType

# # Load data for classification
X, y = load_iris(return_X_y = True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1)
# # Construct ExtraTrees-Model
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1)
model = ExtraTreesClassifier(criterion ='entropy', n_estimators=100, max_features=None)
model.fit(X_train,y_train)
# # Export model into ONNX format
onnxfile = 'extratrees-classifier.onnx'
initial_type = [('float_input', FloatTensorType([None, X.shape[1]]))]
# # Zipmap should be always turned off as it's not implemented in TF3800
onnx_model = convert_sklearn(model, initial_types=initial_type, options={type(model): {'zipmap':Fals
e}})
# # Export to ONNX file
with open(onnxfile, "wb") as f:
    f.write( onnx_model.SerializeToString())
f.close()
exit()
```





5.1.7.2 Random Forest

Ein Random Forest kann sowohl zur Klassifikation als auch zur Regression [▶ 61] eingesetzt werden. Der Algorithmus gehört zu den Ensemblemethoden, da eine vom Nutzer definierte Anzahl von unkorrelierten Entscheidungsbäumen aufgebaut und trainiert wird. Die Vorhersage des Ensembles ergibt sich beim Random Forest aus der gemittelten Vorhersage der einzelnen Bäume.

Ein Random Forest hat gegenüber einzelner Entscheidungsbäume [▶ 37] oft eine bessere *Accuracy*, zu dem Preis, dass der Random Forest keine Transparenz bezüglich der getroffenen Vorhersagen zeigt.

Gegenüber einer SVM ist ein Random Forest gerade für hochdimensionale Daten bzgl. der Rechenzeit performanter.

Unterstützte Eigenschaften

ONNX-Support

- TreeEnsambleClassifier
- TreeEnsambleRegressor

Beispiele für den Export von Random Forest Modellen finden Sie hier: ONNX-Export eines Random Forest [▶ 45]

Einschränkung bei Klassifikation

Bei Klassifikationsmodellen wird nur der Ausgang der Labels in der SPS abgebildet. Die Scores/ Probabilities sind in der SPS nicht verfügbar.

Unterstützte Datentypen

Es ist zu unterscheiden zwischen „supported datatype“ und „preferred datatype“. Der preferred datatype entspricht der Präzision der Ausführungs-Engine.

Der preferred datatype ist floating point 64 (E_MLLDT_FP64-LREAL).

Bei Verwendung eines supported datatype wird eine effiziente Typ-Konvertierung automatisch in der Bibliothek durchgeführt. Durch die Typ-Konvertierung kann es zu minimalen Performance-Einbußen kommen.

Eine Liste der supported datatypes finden Sie unter ETcMlIDataType [▶ 83].

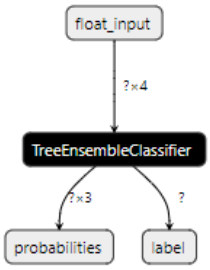
5.1.7.2.1 ONNX-Export eines Random Forest

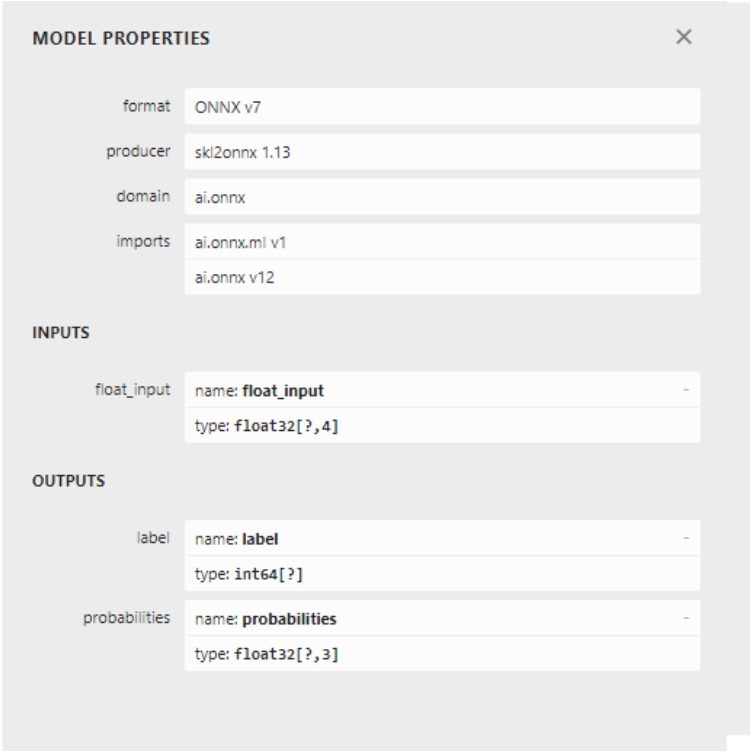
Download Python Samples

Ein Zip-Archiv mit allen Samples finden Sie hier: Beispiele zum ONNX-Export [▶ 63]

Scikit-learn: Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
from skl2onnx import convert_sklearn
from skl2onnx.common.data_types import FloatTensorType

X, y = make_classification(n_samples=1000, n_features=3, n_informative=3, n_redundant=0, random_stat
e=1)

clf = RandomForestClassifier(n_estimators=100)
clf.fit(X, y)

initial_type = [('float_input', FloatTensorType([None, X.shape[1]]))]
# # Zipmap should be always turned off as it's not implemented in TF3800
onnx_model = convert_sklearn(clf, initial_types=initial_type, options={type(clf): {'zipmap':False}})
with open("rf_classifier.onnx", "wb") as f:
    f.write( onnx_model.SerializeToString())
exit()
```

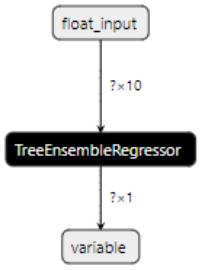


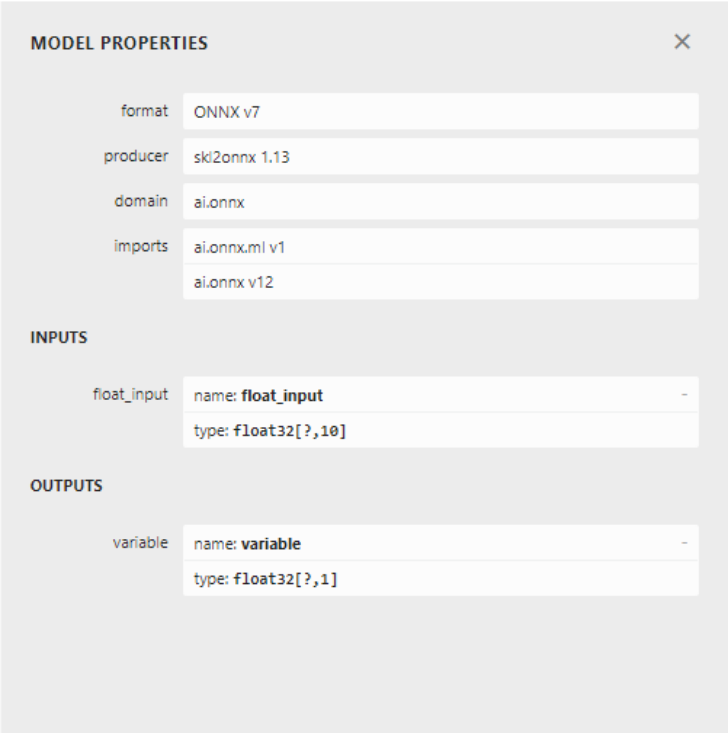


Scikit-learn: Random Forest Regressor

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.datasets import make_regression
from skl2onnx import convert_sklearn
from skl2onnx.common.data_types import FloatTensorType

X, y = make_regression(n_samples=1000, n_features=5, n_informative=5, random_state=2)

clf = RandomForestRegressor(n_estimators=100)
clf.fit(X, y)

initial_type = [('float_input', FloatTensorType([None, X.shape[1]]))]
onnx_model = convert_sklearn(clf, initial_types=initial_type)
```

```
with open("rf_regressor.onnx", "wb") as f:
    f.write( onnx_model.SerializeToString())
```

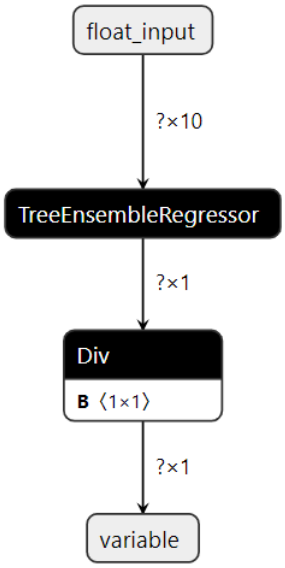


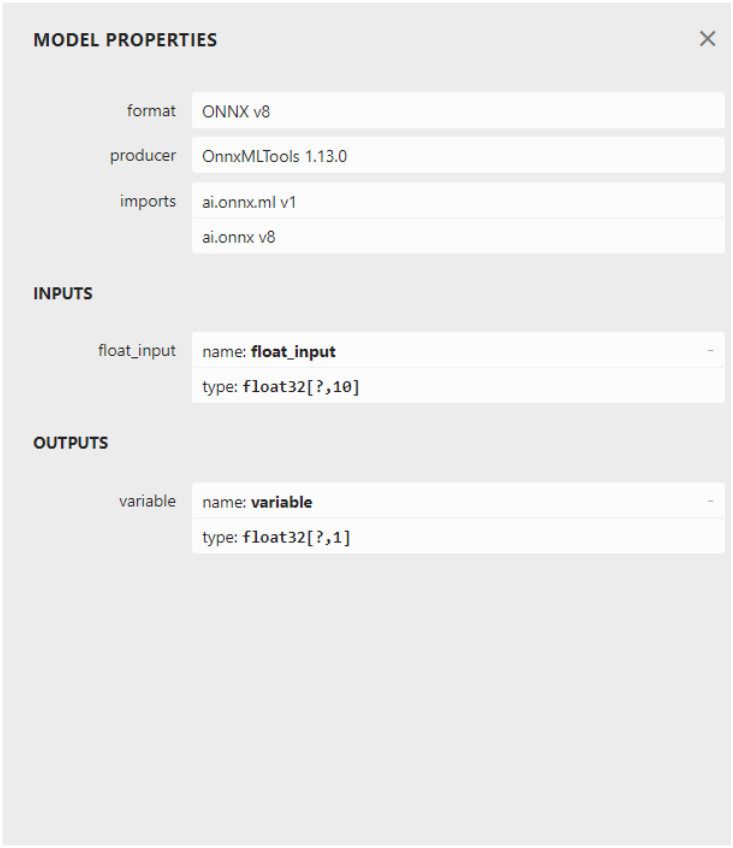


LightGBM: Random Forest Regressor

```
import numpy as np
import onnx_graphsurgeon as gs
import lightgbm as lgb
from lightgbm import LGBMRegressor
from skl2onnx.common.data_types import FloatTensorType
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split
from onnxmltools.convert import convert_lightgbm
import onnxmltools.convert.common.data_types
import onnx
# # Generate data for regression
X, y =  make_regression(n_samples=300, n_features=10, n_informative=10, n_targets=1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1)
# # Construct LightGBM-RandomForest-Model
model = LGBMRegressor(boosting_type='rf', class_weight=None, colsample_bynode=0.3, colsample_bytree=
1.0, importance_type='split', learning_rate=0.05, max_depth=-1, min_child_samples=2, min_child_weigh
t=0.001, min_split_gain=0.0, n_estimators=150, n_jobs=-1, num_class=1, num_leaves=500, objective='re
gression', random_state=None, reg_alpha=0.0, reg_lambda=0.0, silent=True, subsample=0.632, subsample
_for_bin=200000, subsample_freq=1)
model.fit(X_train, y_train,eval_set=[(X_test,y_test),
(X_train,y_train)],eval_metric='rmse', verbose=20)

# # Convert model to ONNX
onnxfile = 'lgbm-regressor-randomforest.onnx'
initial_type = [('float_input', FloatTensorType([None, X.shape[1]]))]
onnx_model = convert_lightgbm(model, initial_types=initial_type, target_opset=12)

# Manipulate ONNX graph

# # Import model to graph object
graph = gs.import_onnx(onnx_model)
graph.name = "LGBM-RandomForest"
# # Modify TreeEnsemble output shape (necessary to meet TwinCAT requirement, working on an update to
 make this step obsolete)
tree_node = [node for node in graph.nodes if node.op == "TreeEnsembleRegressor"][0]
tree_node.outputs[0].shape = [None, 1]
tree_node.outputs[0].dtype = np.float32
# # Modify DIV Node inputs to provide correct averaging (necessary to correct a bug in onnxmltools v
ersion 1.11.1)
div_node = [node for node in graph.nodes if node.op == "Div"][0]
div_node.inputs[1].to_constant(values=np.asarray([[model.n_estimators]], dtype=np.float32))
# # Export graph object to ONNX ProtoModel
graph.cleanup().toposort()
```

```
onnx_model = gs.export_onnx(graph)
# # Add ONNX domain tag to TreeEnsemble Node for proper node recognition (only a reset of the tag as
 it gets lost during onnx manipulation)
tree_node = [node for node in onnx_model.graph.node if node.op_type == "TreeEnsembleRegressor"][0]
tree_node.domain = "ai.onnx.ml"
tree_node.doc_string = "Converted from LGBMRegressor() model with explicit shaping"
# # Export ONNX model to file
onnx.checker.check_model(onnx_model)
with open(onnxfile, "wb") as f:
    f.write( onnx_model.SerializeToString())
f.close()
exit()
```





5.1.7.3 Gradient Boosting

Ein Gradient Boosting Modell kann sowohl zur Klassifikation als auch zur Regression [▶ 61] eingesetzt werden. Das Modell gehört, wie z.B. der Random Forest [▶ 44], zu den Ensemble Tree [▶ 41] Methoden. Gegenüber einem einzelnen Decision Tree kann mit dem Gradient Boosting Modell die Accuracy der Prädiktion verbessert werden, zum Preis, dass das Modell nicht mehr einfach erklärbar ist. Random Forest und Gradient Boosting unterscheiden sich durch die Art wie die einzelnen Trees erzeugt werden.

Unterstützte Eigenschaften

ONNX-Support

- TreeEnsambleClassifier
- TreeEnsambleRegressor

Beispiele für den Export von Gradient Boosting Modellen finden Sie hier: ONNX-Export von Gradient Boosting [▶ 48].

Einschränkung bei Klassifikation

Bei Klassifikationsmodellen wird nur der Ausgang der Labels in der SPS abgebildet. Die Scores/ Probabilities sind in der SPS nicht verfügbar.

Unterstützte Datentypen

Es ist zu unterscheiden zwischen „supported datatype“ und „preferred datatype“. Der preferred datatype entspricht der Präzision der Ausführungs-Engine.

Der preferred datatype ist floating point 64 (E_MLLDT_FP64-LREAL).

Bei Verwendung eines supported datatype wird eine effiziente Typ-Konvertierung automatisch in der Bibliothek durchgeführt. Durch die Typ-Konvertierung kann es zu minimalen Performance-Einbußen kommen.

Eine Liste der supported datatypes finden Sie unter ETcMlIDataType [▶ 83].

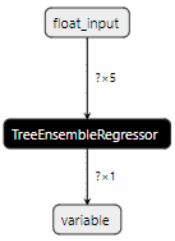
5.1.7.3.1 ONNX-Export von Gradient Boosting

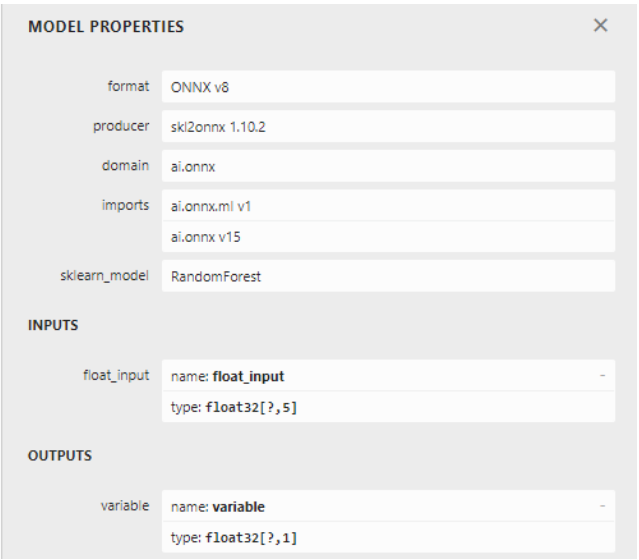
Download Python Samples

Ein Zip-Archiv mit allen Samples finden Sie hier: Beispiele zum ONNX-Export [▶ 63]

Gradient Boosting Regressor mit Scikit-learn

```
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.datasets import make_regression
from skl2onnx import convert_sklearn
from skl2onnx.common.data_types import FloatTensorType
# import onnx

# # Generate data for regression
X, y =  make_regression(n_samples=300, n_features=10, n_informative=10, n_targets=1)
# # Construct the model
model = GradientBoostingRegressor(learning_rate=0.08, max_depth=3, n_estimators=300)
model.fit(X,y)

# # Convert model to ONNX
onnxfile = 'gdb-regressor.onnx'
initial_type = [('float_input', FloatTensorType([None, X.shape[1]]))]
onnx_model = convert_sklearn(model, initial_types=initial_type, target_opset=12)
# # Export to ONNX file
# onnx.checker.check_model(onnx_model)
with open(onnxfile, "wb") as f:
    f.write( onnx_model.SerializeToString())
f.close()
exit()
```

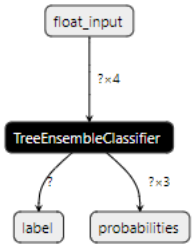


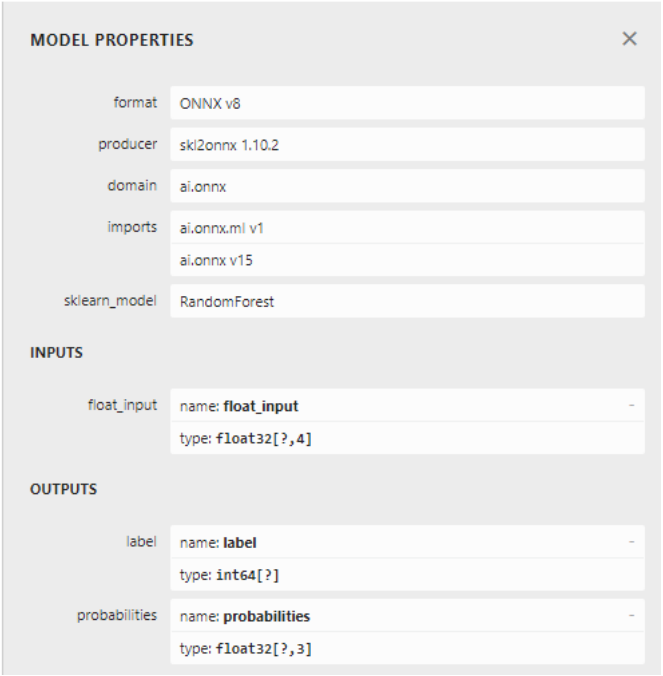


Gradient Boosting Classifier mit Scikit-learn

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from skl2onnx import convert_sklearn
from skl2onnx.common.data_types import FloatTensorType
# import onnx
```

```
# # Load data for classification
X, y = load_iris(return_X_y = True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1)
model = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, max_depth=3)
model.fit(X_train,y_train)

# # Convert model to ONNX
onnxfile = 'gdb-classifier.onnx'
initial_type = [('float_input', FloatTensorType([None, X.shape[1]]))]
# # Zipmap should be always turned off as it's not implemented in TF3800
onnx_model = convert_sklearn(model, initial_types=initial_type, options={type(model): {'zipmap':Fals
e}}, target_opset=12)
# # Export to ONNX file
# onnx.checker.check_model(onnx_model)
with open(onnxfile, "wb") as f:
    f.write( onnx_model.SerializeToString())
f.close()
exit()
```





5.1.7.4 Histogramm-basiertes Gradient Boosting

Ein Histogramm-basiertes Gradient Boosting Modell kann sowohl zur Klassifikation als auch zur Regression [▶ 61] eingesetzt werden.

Das Modell basiert auf dem Gradient Boosting [▶ 47], allerdings werden hier die kontinuierlichen Eingänge mit Hilfe eines Histogramms in Bins diskretisiert. Dadurch wird das Training des Modells, gerade bei sehr großen Datensätzen, massiv beschleunigt.

Unterstützte Eigenschaften

ONNX-Support

- TreeEnsambleClassifier
- TreeEnsambleRegressor

Beispiele für den Export von Hist Gradient Boosting Modellen finden Sie hier: ONNX-Export von Hist Gradient Boosting [▶ 50].

Einschränkung bei Klassifikation

Bei Klassifikationsmodellen wird nur der Ausgang der Labels in der SPS abgebildet. Die Scores/ Probabilities sind in der SPS nicht verfügbar.

Unterstützte Datentypen

Es ist zu unterscheiden zwischen „supported datatype“ und „preferred datatype“. Der preferred datatype entspricht der Präzision der Ausführungs-Engine.

Der preferred datatype ist floating point 64 (E_MLLDT_FP64-LREAL).

Bei Verwendung eines supported datatype wird eine effiziente Typ-Konvertierung automatisch in der Bibliothek durchgeführt. Durch die Typ-Konvertierung kann es zu minimalen Performance-Einbußen kommen.

Eine Liste der supported datatypes finden Sie unter ETcMlIDataType [▶ 83].

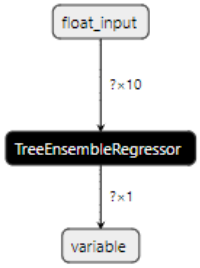
5.1.7.4.1 ONNX-Export von Hist Gradient Boosting

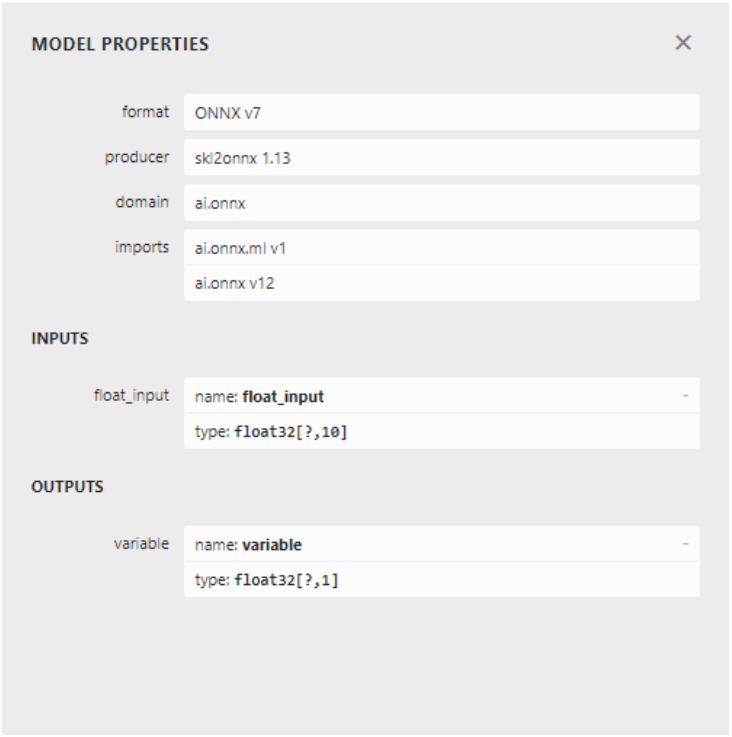
Download Python Samples

Ein Zip-Archiv mit allen Samples finden Sie hier: Beispiele zum ONNX-Export [▶ 63]

Hist Gradient Boosting Regressor mit Scikit-learn

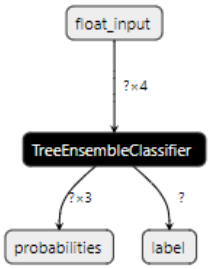
```
from sklearn.ensemble import HistGradientBoostingRegressor
from sklearn.datasets import make_regression
from skl2onnx import convert_sklearn
from skl2onnx.common.data_types import FloatTensorType
# import onnx

# # Generate data for regression
X, y =  make_regression(n_samples=300, n_features=10, n_informative=10, n_targets=1)
# # Construct the model
model = HistGradientBoostingRegressor(learning_rate=0.08, max_depth=3, max_iter=300)
model.fit(X,y)

# # Convert model to ONNX
onnxfile = 'histgdb-regressor.onnx'
initial_type = [('float_input', FloatTensorType([None, X.shape[1]]))]
onnx_model = convert_sklearn(model, initial_types=initial_type, target_opset=12)
# # Export to ONNX file
# onnx.checker.check_model(onnx_model)
with open(onnxfile, "wb") as f:
    f.write( onnx_model.SerializeToString())
f.close()
exit()
```

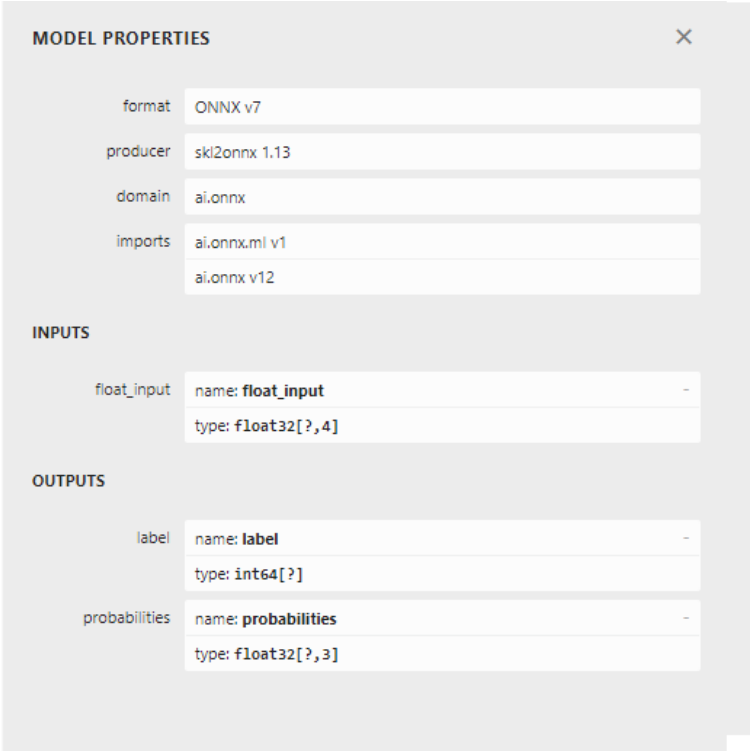




Hist Gradient Boosting Classifier mit Scikit-learn

```
from sklearn.ensemble import HistGradientBoostingClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from skl2onnx import convert_sklearn
from skl2onnx.common.data_types import FloatTensorType
# import onnx

# # Load data for classification
X, y = load_iris(return_X_y = True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1)
# # Construct the model
model = HistGradientBoostingClassifier(learning_rate=0.08, max_depth=3, max_iter=300)
model.fit(X_train,y_train)

# # Convert model to ONNX
onnxfile = 'histgdb-iris.onnx'
initial_type = [('float_input', FloatTensorType([None, X.shape[1]]))]
# # Zipmap should be always turned off as it's not implemented in TF3800
onnx_model = convert_sklearn(model, initial_types=initial_type, options={type(model): {'zipmap':Fals
e}}, target_opset=12)
# # Export to ONNX file
# onnx.checker.check_model(onnx_model)
with open(onnxfile, "wb") as f:
    f.write( onnx_model.SerializeToString())
f.close()
exit()
```





5.1.7.5 XGBoost

Ein XGBoost Modell kann sowohl zur Klassifikation als auch zur Regression [▶ 61] eingesetzt werden.

Der XGBoost hat Vorteile bei der Generalisierung des Modells im Vergleich zum Gradient Boosting [▶ 47]. Der Trainingsdatensatz sollte groß sein – deutlich mehr Samples im Vergleich zur Anzahl der genutzten Feature.

Unterstützte Eigenschaften

ONNX-Support

- TreeEnsambleClassifier
- TreeEnsambleRegressor

Beispiele für den Export von XGBoost Modellen finden Sie hier: ONNX-Export von XGBoost [▶ 53]

Einschränkung bei Klassifikation

Bei Klassifikationsmodellen wird nur der Ausgang der Labels in der SPS abgebildet. Die Scores/ Probabilities sind in der SPS nicht verfügbar.

Unterstützte Datentypen

Es ist zu unterscheiden zwischen „supported datatype“ und „preferred datatype“. Der preferred datatype entspricht der Präzision der Ausführungs-Engine.

Der preferred datatype ist floating point 64 (E_MLLDT_FP64-LREAL).

Bei Verwendung eines supported datatype wird eine effiziente Typ-Konvertierung automatisch in der Bibliothek durchgeführt. Durch die Typ-Konvertierung kann es zu minimalen Performance-Einbußen kommen.

Eine Liste der supported datatypes finden Sie unter ETcMllDataType [▶ 83].

5.1.7.5.1 ONNX-Export von XGBoost

Download Python Samples

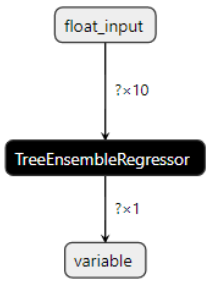
Ein Zip-Archiv mit allen Samples finden Sie hier: Beispiele zum ONNX-Export [▶ 63]

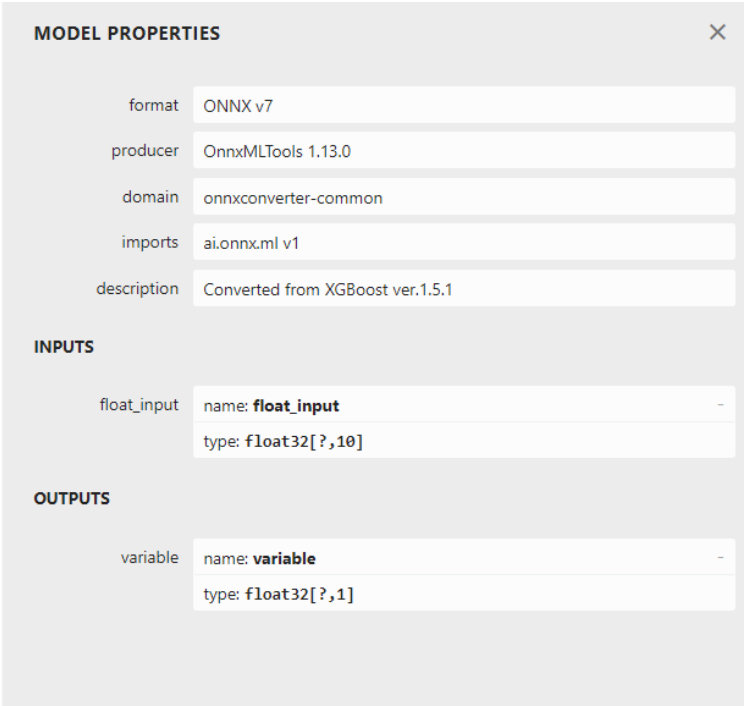
Einschränkung der Version von XGBoost

Die unterstützten Versionen von XGBoost sind auf Grund des ONNX-Export-Verhaltens beschränkt: Version <= 1.5.2 oder >= 1.7.4.

XGB Regressor

```
# # Important Requirement: XGBoost version must be 1.7.4 or higher (otherwise onnxmltools 1.11.1 doe
s not match)
import xgboost as xgb
from xgboost import XGBRegressor
from sklearn.datasets import make_regression
from skl2onnx.common.data_types import FloatTensorType
from onnxmltools.convert import convert_xgboost
import onnx

# # Generate data for regression
X, y =  make_regression(n_samples=300, n_features=10, n_informative=10, n_targets=1)
# # Construct XGB-Model
model = XGBRegressor(objective='reg:squarederror',booster='gbtree', max_depth=3, learning_rate=0.08,
 n_estimators=300)
model.fit(X,y)
# # Convert model to ONNX
onnxfile = 'xgb-regressor.onnx'
initial_type = [('float_input', FloatTensorType([None, X.shape[1]]))]
onnx_model = convert_xgboost(model, initial_types=initial_type, target_opset=12)
onnx_model.graph.doc_string = "Converted from XGBoost ver."+xgb.__version__
# # Export to ONNX file
onnx.checker.check_model(onnx_model)
with open(onnxfile, "wb") as f:
    f.write( onnx_model.SerializeToString())
f.close()
exit()
```

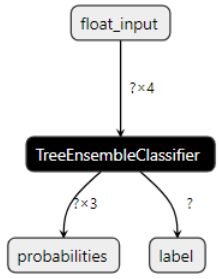


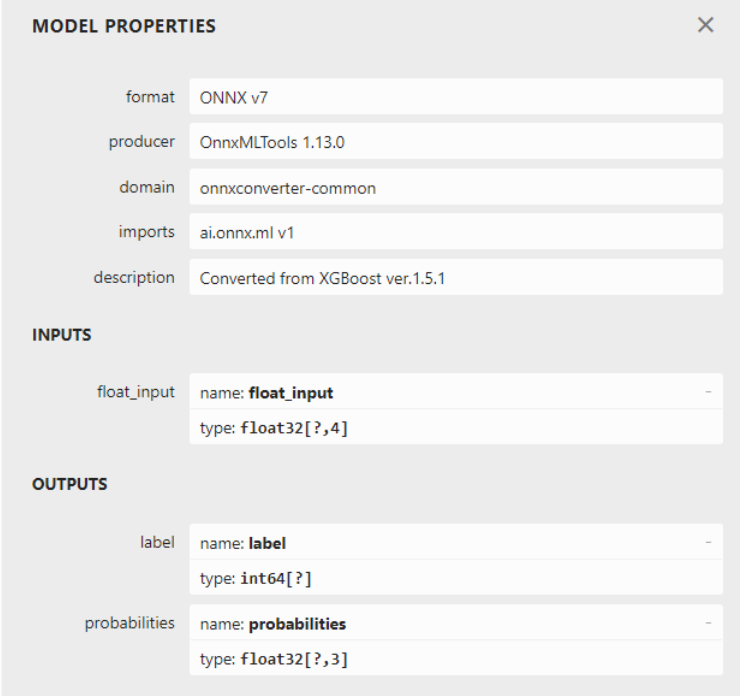


XGB Classifier

```
import onnx
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
import xgboost as xgb # # Important Requirement: XGBoost version must be 1.7.4 or higher (otherwise
```

```
onnxmltools 1.11.1 does not match)
from xgboost import XGBClassifier
from skl2onnx.common.data_types import FloatTensorType
from onnxmltools.convert import convert_xgboost

X, y = load_iris(return_X_y = True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1)
model = XGBClassifier(objective= 'multi:softmax', learning_rate=0.03, max_depth=3, n_estimators=300,
 eval_metric='mlogloss', early_stopping_rounds=20, verbosity=1, use_label_encoder=False)
model.fit(X_train,y_train, eval_set=[(X_train, y_train), (X_test, y_test)], verbose=True)
onnxfile = 'xgb-iris.onnx'
# # Convert model to ONNX
initial_type = [('float_input', FloatTensorType([None, X.shape[1]]))]
onnx_model = convert_xgboost(model, initial_types=initial_type, target_opset=12)
onnx_model.graph.doc_string = "Converted from XGBoost ver."+xgb.__version__
# # Export to ONNX file
onnx.checker.check_model(onnx_model)
with open(onnxfile, "wb") as f:
    f.write( onnx_model.SerializeToString())
f.close()
exit()
```

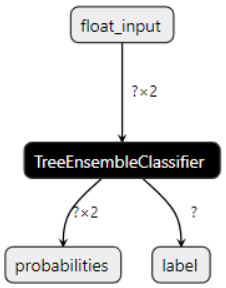


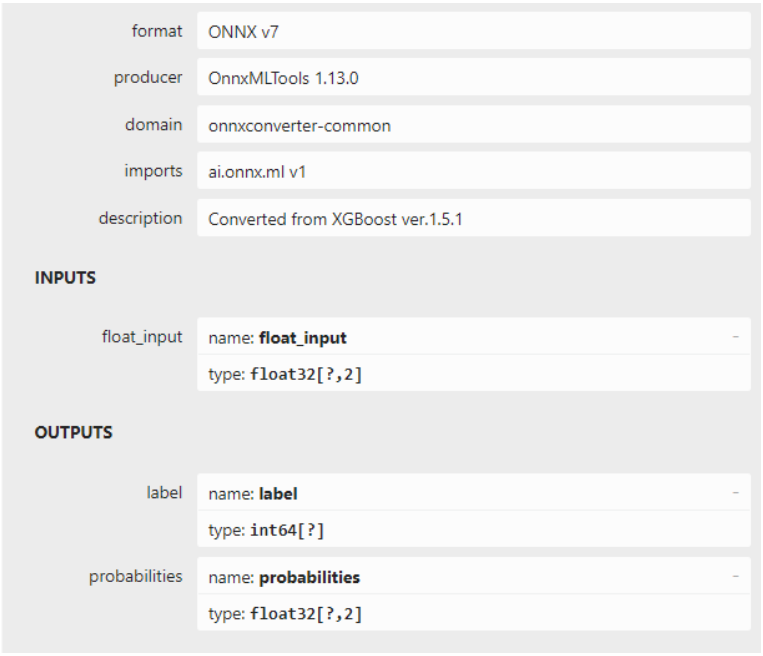


XGB binary classifier

```
from skl2onnx.common.data_types import FloatTensorType
from sklearn.model_selection import train_test_split
from sklearn.datasets import  make_moons, make_circles, make_classification
import onnx
# # Important Requirement: XGBoost version must be 1.7.4 or higher (otherwise onnxmltools 1.11.1 doe
s not match)
import xgboost as xgb
from xgboost import XGBClassifier
from onnxmltools.convert import convert_xgboost

# # Generate data for binary classification
X, y =  make_moons(n_samples=300, noise=0.3, random_state=1)
# X, y = make_circles(n_samples=300, shuffle=True, noise=0.3, random_state=1, factor=0.8)
# # Construct XGB-Model
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1)
model = XGBClassifier(objective= 'binary:logistic', learning_rate=0.03, max_depth=3, n_estimators=30
0, eval_metric='auc', early_stopping_rounds=20, verbosity=1, use_label_encoder=False)
model.fit(X_train,y_train, eval_set=[(X_train, y_train), (X_test, y_test)], verbose=True)
# # Convert model to ONNX
onnxfile = 'xgb-binary.onnx'
initial_type = [('float_input', FloatTensorType([None, X.shape[1]]))]
onnx_model = convert_xgboost(model, initial_types=initial_type, target_opset=12)
onnx_model.graph.doc_string = "Converted from XGBoost ver."+xgb.__version__
# # Export to ONNX file
onnx.checker.check_model(onnx_model)
with open(onnxfile, "wb") as f:
```

```
    f.write( onnx_model.SerializeToString())
f.close()
exit()
```





5.1.7.6 LightGBM

Ein LightGBM Modell kann sowohl zur Klassifikation als auch zur Regression [▶ 61] eingesetzt werden.

LightGBM zählt zu den Histogramm-basierten Gradient Boosting [▶ 49] Methoden. Dadurch wird das Training, gerade bei großen Datensätzen, effizient.

Unterstützte Eigenschaften

ONNX-Support

- TreeEnsambleClassifier
- TreeEnsambleRegressor

Beispiele für den Export von LightGBM Modellen finden Sie hier: ONNX-Export von LightGBM [▶ 56].

Einschränkung bei Klassifikation

Bei Klassifikationsmodellen wird nur der Ausgang der Labels in der SPS abgebildet. Die Scores/ Probabilities sind in der SPS nicht verfügbar.

Unterstützte Datentypen

Es ist zu unterscheiden zwischen „supported datatype“ und „preferred datatype“. Der preferred datatype entspricht der Präzision der Ausführungs-Engine.

Der preferred datatype ist floating point 64 (E_MLLDT_FP64-LREAL).

Bei Verwendung eines supported datatype wird eine effiziente Typ-Konvertierung automatisch in der Bibliothek durchgeführt. Durch die Typ-Konvertierung kann es zu minimalen Performance-Einbußen kommen.

Eine Liste der supported datatypes finden Sie unter ETcMlIDataType [▶ 83].

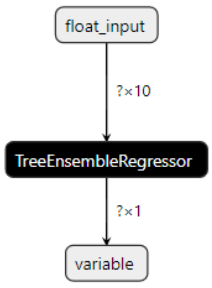
5.1.7.6.1 ONNX-Export von LightGBM

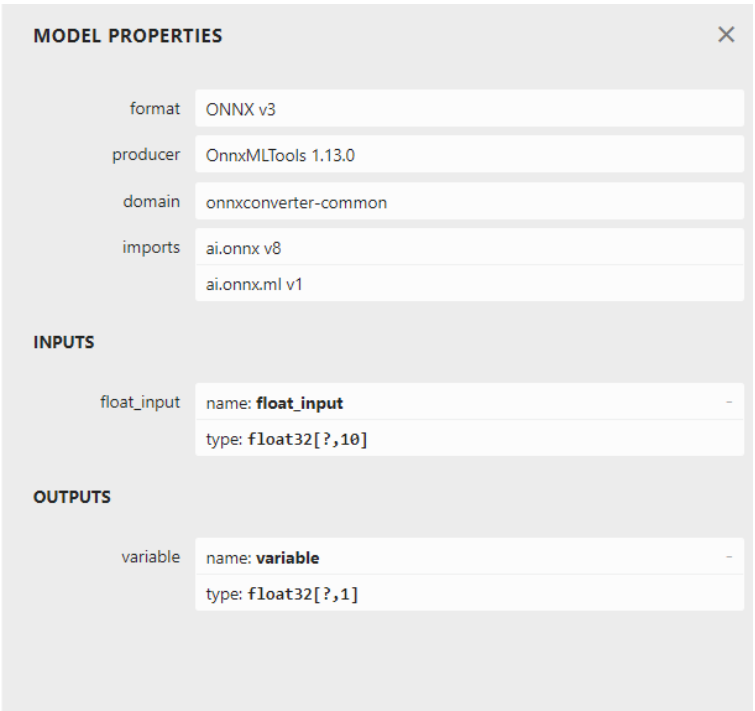
Download Python Samples

Ein Zip-Archiv mit allen Samples finden Sie hier: Beispiele zum ONNX-Export [▶ 63]

LGBM Regressor

```
from lightgbm import LGBMRegressor
from skl2onnx.common.data_types import FloatTensorType
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split
from onnxmltools.convert import convert_lightgbm
import onnx

# # Generate data for regression
X, y =  make_regression(n_samples=300, n_features=10, n_informative=10, n_targets=1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1)
# # Construct LightGBM-Model
model = LGBMRegressor(objective='regression', max_depth=31, learning_rate=0.05, n_estimators=300)
model.fit(X_train, y_train,eval_set=[(X_test,y_test),
(X_train,y_train)],eval_metric='rmse', verbose=20)

# # Convert model to ONNX
onnxfile = 'lgbm-regressor.onnx'
initial_type = [('float_input', FloatTensorType([None, X.shape[1]]))]
onnx_model = convert_lightgbm(model, initial_types=initial_type, target_opset=12)

onnx.checker.check_model(onnx_model)
with open(onnxfile, "wb") as f:
    f.write( onnx_model.SerializeToString())
f.close()
exit()
```

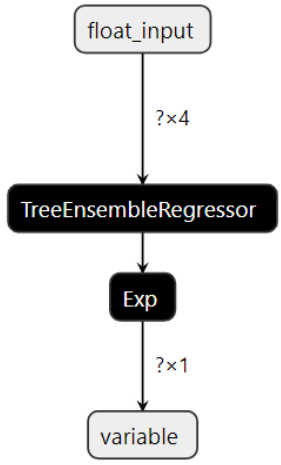


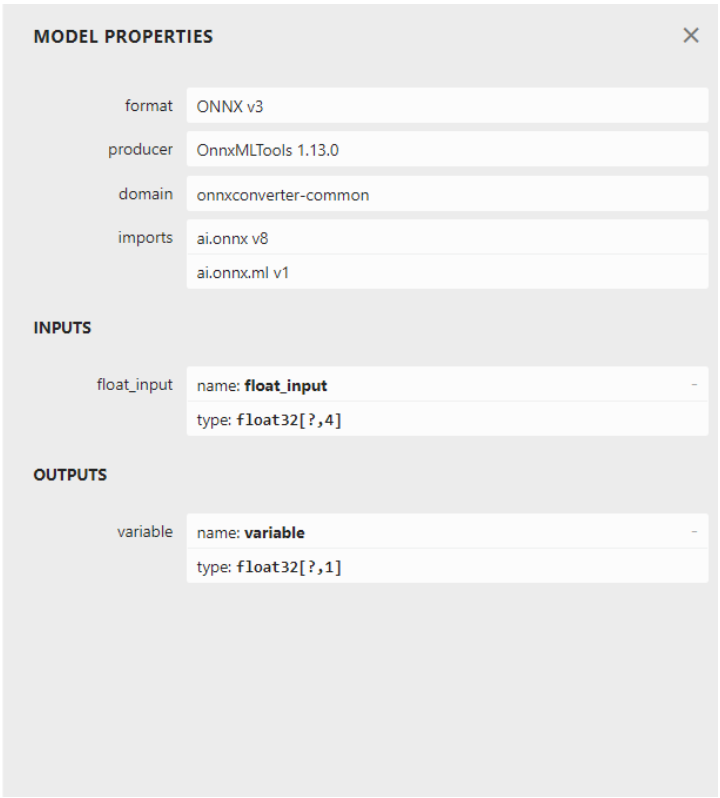


LGBM Regressor (gamma objective)

```
import numpy as np
from lightgbm import LGBMRegressor
from skl2onnx.common.data_types import FloatTensorType
from sklearn.model_selection import train_test_split
from onnxmltools.convert import convert_lightgbm
import onnx

# # Generate data for regression
N_ROWS = 1000
N_COLS = 4
```

```
X = np.random.randn(N_ROWS, N_COLS)
# # For 'poisson' and 'gamma' objective, all target values need to be non-negative
y = abs(np.random.randn(N_ROWS))
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1)
# # Construct LightGBM-Model
model = LGBMRegressor(objective='gamma', max_depth=-1, learning_rate=0.05, n_estimators=300)
model.fit(X_train, y_train,eval_set=[(X_test,y_test),
(X_train,y_train)],eval_metric='rmse', verbose=20)

# # Convert model to ONNX
onnxfile = 'lgbm-regressor-gamma.onnx'
initial_type = [('float_input', FloatTensorType([None, X.shape[1]]))]
onnx_model = convert_lightgbm(model, initial_types=initial_type, target_opset=12)
onnx.checker.check_model(onnx_model)
with open(onnxfile, "wb") as f:
    f.write( onnx_model.SerializeToString())
f.close()
exit()
```

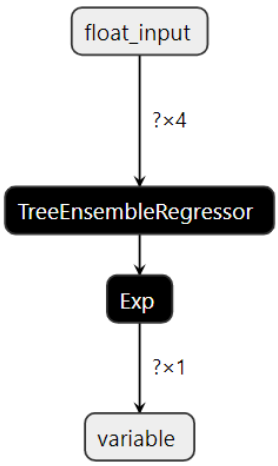


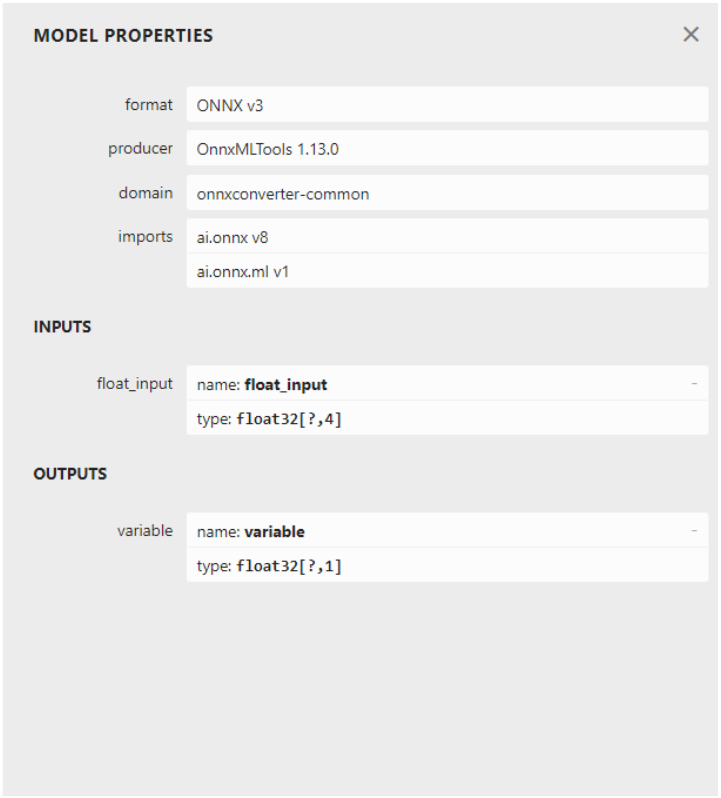


LGBM Regressor (poisson objective)

```
import numpy as np
from lightgbm import LGBMRegressor
from skl2onnx.common.data_types import FloatTensorType
from sklearn.model_selection import train_test_split
from onnxmltools.convert import convert_lightgbm
import onnx

# # Generate data for regression
N_ROWS = 1000
N_COLS = 4
X = np.random.randn(N_ROWS, N_COLS)
# # For 'poisson' and 'gamma' objective, all target values need to be non-negative
y = abs(np.random.randn(N_ROWS))
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1)
# # Construct LightGBM-Model
model = LGBMRegressor(objective='poisson', max_depth=-1, learning_rate=0.05, n_estimators=300)
model.fit(X_train, y_train,eval_set=[(X_test,y_test),
(X_train,y_train)],eval_metric='rmse', verbose=20)

# # Convert model to ONNX
onnxfile = 'lgbm-regressor-poisson.onnx'
initial_type = [('float_input', FloatTensorType([None, X.shape[1]]))]
onnx_model = convert_lightgbm(model, initial_types=initial_type, target_opset=12)
onnx.checker.check_model(onnx_model)
```

```
with open(onnxfile, "wb") as f:
    f.write( onnx_model.SerializeToString())
f.close()
exit()
```

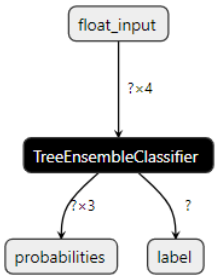


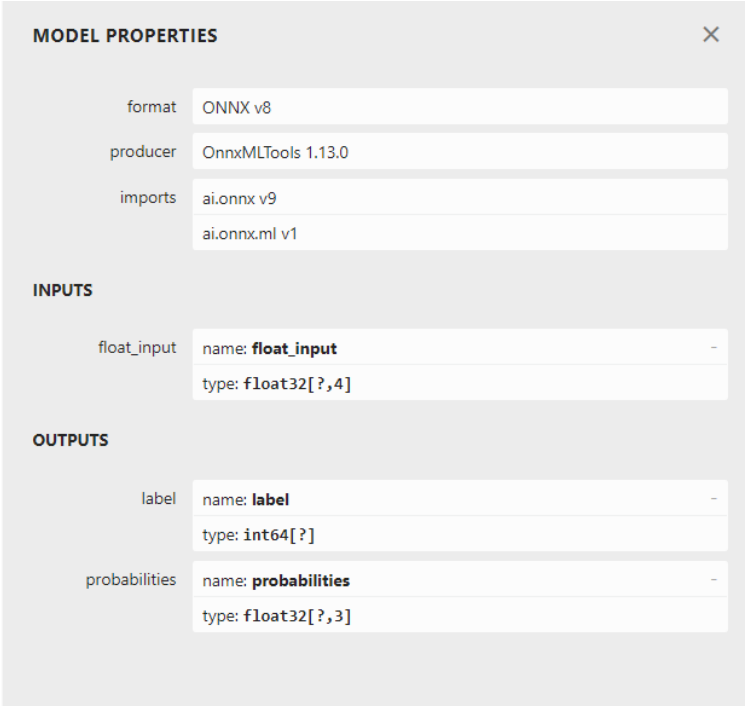


LGBM Classifier

```
import onnx
import onnx_graphsurgeon as gs
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from lightgbm import LGBMClassifier
from skl2onnx.common.data_types import FloatTensorType
from onnxmltools.convert import convert_lightgbm

# # Load data for classification
X, y = load_iris(return_X_y = True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1)
# # Construct LightGBM Model
model = LGBMClassifier(objective='multiclass', learning_rate=0.05, max_depth=-1, n_estimators=100, r
andom_state=42)
model.fit(X_train,y_train,eval_set=[(X_test,y_test),
(X_train,y_train)],verbose=20,eval_metric='logloss')

# # Convert model to ONNX
onnxfile = 'lgbm-iris.onnx'
initial_type = [('float_input', FloatTensorType([None, X.shape[1]]))]
# # Zipmap should be always turned off as it's not implemented in TF3800
onnx_model = convert_lightgbm(model, initial_types=initial_type, zipmap=False)

# # Manipulate graph to force its ONNX conformity (necessary to correct a bug in onnxmltools version
 1.11.1)
graph = gs.import_onnx(onnx_model)
graph.name = "LGBMClassifier"
tree_node = [node for node in graph.nodes if node.op == "TreeEnsembleClassifier"][0]
tree_node.name = "TreeEnsembleClassifier"
tree_node.outputs = graph.outputs
tree_node.outputs[0].shape = [None]
tree_node.outputs[1].shape = [None, model.n_classes_]
# # Collect 2 artifacts of the converter
cast_node = [node for node in graph.nodes if node.op == "Cast"][0]
mul_node = [node for node in graph.nodes if node.op == "Mul"][0]
# # Clear outputs of these two nodes
mul_node.outputs.clear()
cast_node.outputs.clear()
```

```
# # Remove these nodes from the graph
graph.cleanup().toposort()
onnx_model = gs.export_onnx(graph)
nodes = onnx_model.graph.node
for node in nodes:
    # # Modify node attributes.
    if node.op_type == "TreeEnsembleClassifier":
        node.domain = "ai.onnx.ml" # # Domain info is required for ONNX conformity
        node.doc_string = "Converted from LGBMClassifier() model"
# # Export to ONNX file
onnx.checker.check_model(onnx_model)
with open(onnxfile, "wb") as f:
    f.write( onnx_model.SerializeToString())
f.close()
exit()
```





5.2 Machine Learning Cheat Sheet: Auswahl von Modellen

Welches der unterstützten Modelle passt auf meine Problemstellung? Diese Frage wird häufig gestellt. Folgende Ausführungen sollen bei der Auswahl geeigneter Algorithmen behilflich sein.

Art der Eingangsdaten (Input-Daten) des Modells

Eine erste wesentliche Fragestellung betrifft die Art der Input-Daten des Modells: Bilddaten, Zeitreihen oder *Tabular Data*?

Die unterstützten Modelle eignen sich hauptsächlich für *Tabular Data*. Damit ist gemeint, dass der Input des Modells ein Array von Werten bildet.

Bilddaten

Für die direkte Verarbeitung von Bilddaten werden in der Regel **C**onvolutional **N**eural **N**etworks (CNNs) genutzt. Diese werden voraussichtlich ab Q3/2023 unterstützt. Für einen eingeschränkten Anwendungsbereich können auch MLPs ausreichende Ergebnisse liefern. Dabei werden die Bildpixel als Vektor in das Model gegeben.

Zielführend ist es darüber hinaus, aus den Bilddaten zunächst Features zu extrahieren und diese Features als Input-Daten eines KI-Modells zu verwenden. Zur Bildaufnahme, Vorverarbeitung und Feature-Generierung steht mit TwinCAT Vision eine leistungsfähige Bibliothek zur Verfügung. Die Features können dann als Array zusammengefasst und das Problem als Tabular Data Problem aufgefasst werden.

Zeitreihen

Für die direkte Verarbeitung von Zeitreihen, d. h. Reihen von Datenpunkten, bei denen die zeitliche Abfolge der Samples wesentliche Informationen tragen, werden in der Regel rekurrente Neuronale Netze, wie das LSTM genutzt. Diese werden voraussichtlich ab Q3/2023 unterstützt. Für einen eingeschränkten Anwendungsbereich können auch MLPs ausreichende Ergebnisse liefern. Dabei werden N-Samples als Vektor in das Model gegeben.

Zielführend ist darüber hinaus, aus den Zeitreihen Signal-Feature zu extrahieren und diese Features als Input-Daten eines KI-Modells zu verwenden. Zur Feature-Generierung von Zeitreihen eignen sich SPS-Bibliotheken, wie die Condition Monitoring Bibliothek oder die Analytics Bibliothek. In der Praxis hat es sich als sehr effizient herausgestellt, z. B. statistische Größen, wie Mittelwert, Standardabweichung, Maximal- und Minimalwert usw. über einen definierten Zeitausschnitt zu bilden. Der Zeitausschnitt kann dabei die Länge eines Prozessschritts sein, beispielsweise vom Beginn bis zum Ende eines Schnitts oder vom Beginn bis zum Ende eines Biegeprozesses. Neben statistischen Größen haben sich, insbesondere bei rotierenden Prozessen, frequenzbasierte Feature, wie die Signalleistung in definierten Frequenzbändern, als nützlich erwiesen. Die generierten Features werden dann in einem Array zusammengefasst und als Input für das KI-Modell verwendet. Entsprechend kann das Problem als Tabular Data Problem interpretiert werden.

Tabular Data

Tabular Data kann direkt als Input für die meisten KI-Modelle verwendet werden. Situationen, bei denen direkt ein Array von Input-Daten bereitsteht, können sein: Es wird mit unterschiedlichen Messmitteln die Länge, Breite, Masse sowie dessen optische Komponenten in R-, G-, und B-Werten, gemessen. Die Werte können direkt als Array von 6 Elementen zusammengefasst und als Input für ein KI-Modell verwendet werden – zum Beispiel zur Klassifikation in OK oder nicht OK.

Beschreibung der Zielstellung

Ist die Art der Eingangsdaten definiert, stellt sich die Frage, was genau das KI-Modell leisten soll bzw. unter gegebenen Bedingungen leisten kann. Sind annotierte (gelabelte) Daten vorhanden und von welchem Typ sind die Label?

Clustering

Beim Clustering wird die innere Struktur der Eingangsdaten analysiert. Bei der Cluster-Analyse sind keine annotierten Daten notwendig, allerdings muss die Anzahl k der erwarteten Cluster bekannt sein.

Anomaly detection

Eine beliebte Anwendung, ebenfalls für den Fall, dass keine annotierten Daten vorhanden sind. In der Trainingsphase werden dem Modell nur Daten präsentiert, welche als „Normal“ zu bezeichnen sind. In der Inferenzphase kann das Modell unterscheiden zwischen bekannter Eingangsdatenstruktur und unbekannter Eingangsdatenstruktur. In letzterem Fall wird von einer Anomalie ausgegangen. Herausforderung bei der Anomaliy detection ist das preprocessing der Trainingsdaten, sodass möglichst nur der Normal-Fall im Training genutzt wird sowie die limitierte Aussagekraft des Ergebnisses.

Dimensionality reduction

Menschen können sich bildlich den 2- und auch 3-dimensionalen Raum gut vorstellen. Punktewolken in einem 3D-Plot sind handhabbar und können das Verständnis über Prozesse verbessern. Sind mehrere Dimensionen beteiligt, wird es schnell unübersichtlich. Bei der Dimensionsreduktion geht es darum, einen N-dimensionalen Eingangsvektor auf einen kleineren Vektor abzubilden, wobei möglichst wenig Informationen verloren gehen sollen: Zum Beispiel wird ein 10-dimensionaler Eingang auf 3 Dimensionen reduziert und dabei bleiben 95% der Informationen erhalten. Ausgenutzt werden redundante Informationen der Eingangsdaten. Die Dimensionsreduktion eignet sich gut als Feature-Generierungsschritt, z. B. vor einem Klassifikator.

Regression

Ein Regressionsproblem setzt voraus, dass ein annotierter Datensatz vorliegt. In der Regel wird ein Problem beschrieben mit N REAL oder LREAL als Eingang des Modells und M REAL oder LREAL als Ausgang. Beispiel: Bei einem Verformungsprozess werden N Features erstellt (z. B.: Maximum, Standardabweichung, Schiefe des Servomotorstroms). Zu diesen Features ist jeweils bekannt der resultierende Durchmesser des verformten Produkts in Längs- und Querrichtung. Aus den 3 Features werden entsprechend 2 Werte geschätzt.

Soll der Verlauf einer Zeitreihe modelliert werden, kann man als Eingangsvektor die N vergangenen Zeit-Werte nutzen und den N+1 Wert als Label verwenden. Somit entfällt das händische Labeln der Daten.

Classification

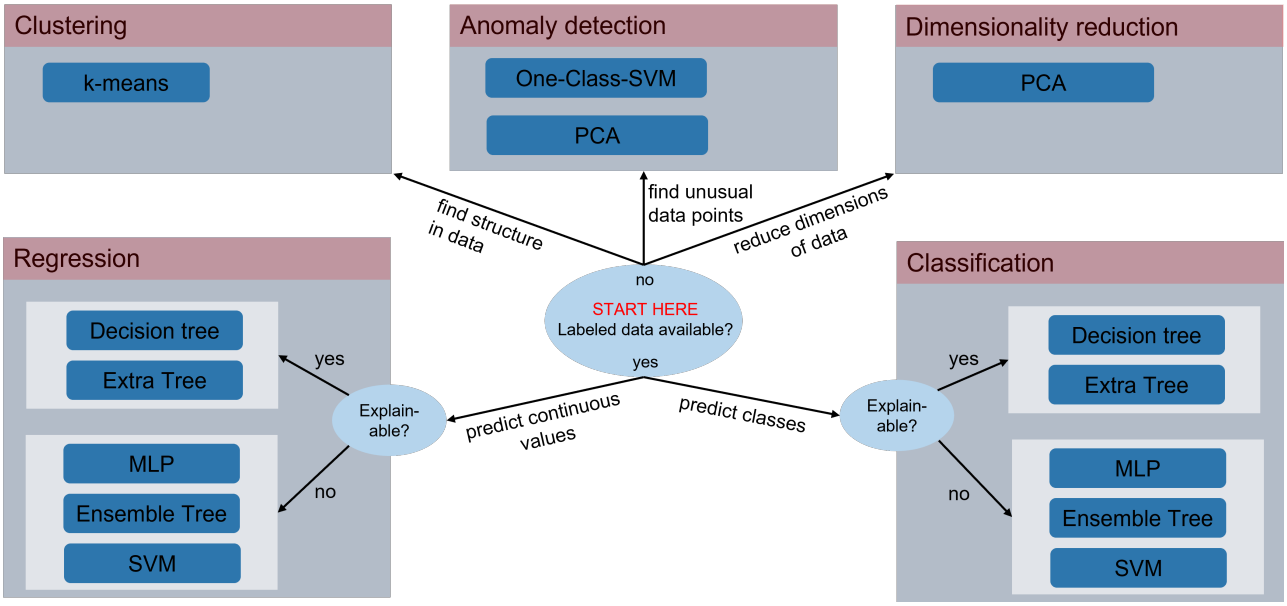
Ein Klassifikationsproblem setzt einen annotierten Datensatz voraus. Hier werden in der Regel N REAL- oder LREAL-Werte auf eine Kategorie, in TwinCAT in der Regel als INT dargestellt, abgebildet. Beispielsweise wird aus N Features berechnet, ob ein gefertigtes Produkt der Qualitätsklasse A, B oder C entspricht.

Erklärbarkeit eines KI-Modells

In manchen Situationen ist es von größter Bedeutung, die Ergebnisse eines KI-Modells erklären zu können, also bspw. die Frage zu beantworten, warum ein Modell ein Produkt als fehlerhaft eingestuft hat. Leider arbeiten die meisten Algorithmen wie Black Boxes, und die Ergebnisse sind nur schwer bis gar nicht zu erklären – auch wenn sie sehr genau sind. Entscheidungsbäume sind gut erklärbare Modelle, da man den Pfad durch den Baum mit den einzelnen Grenzwerten der Abzweigungen nachvollziehen kann. Allerdings ist die Accuracy dieser Modelle oft nicht so überzeugend wie bei anderen nicht erklärbaren Modellen.

KI-Modell Cheat Sheet

Folgende Abbildung gibt einen einfachen Leitfaden zur Wahl eines geeigneten KI-Modells. Sie veranschaulicht die Einordnung von KI-Modellen für unterschiedliche Anwendungszwecke; Vorausgesetzt, es handelt sich um Tabular Data als Modell-Input.



5.3 ONNX erstellen und konvertieren

Der Lernprozess eines Machine Learning-Modells findet außerhalb der Echtzeit, in der Regel in einer Skriptsprache, wie Python oder R, statt. Das gelernte Modell ist aus dem gewählten ML-Framework als ONNX-Datei zu exportieren. Um die Beschreibung des Modells in der TwinCAT Laufzeit zu laden, muss die ONNX-Datei zuvor in ein proprietäres, TwinCAT 3- spezifisches, Format konvertiert werden.

Die Beschreibungsformate für Machine Learning-Modelle im Überblick:

- Open Neural Network Exchange Format (ONNX) [▶ 62]
- Beckhoff ML XML [▶ 75]
- Beckhoff ML BML [▶ 78]

Direkt von der Machine Learning Runtime lesbar sind die Beckhoff-eigenen Formate XML und BML. Das ONNX-Datenformat ist über bereitgestellte Konverter [▶ 65] in ein Beckhoff-eigenes Format zu überführen.

Während ONNX und XML offen einsehbare Formate darstellen, ist das BML ein Binärformat und damit vor allem charakterisiert durch eine kleine Dateigröße und ein effizienteres Ladeverhalten (Ausführzeit der Configure-Methode) in der XAR.

5.3.1 Open Neural Network Exchange (ONNX)

Was ist ONNX?

ONNX ist ein offenes Dateiformat zur Repräsentation von Machine Learning-Modellen und wird als Community-Projekt verwaltet. Homepage der ONNX Community: onnx.ai

Das ONNX-Format definiert Gruppen von Operatoren in einem standardisierten Format, sodass gelernte Modelle interoperabel mit diversen Frameworks, Runtimes und weiteren Werkzeugen genutzt werden können.

ONNX unterstützt sowohl Beschreibungen Neuronaler Netze als auch klassischer Machine Learning Algorithmen und ist damit sowohl für die TwinCAT Machine Learning Inference Engine als auch für die TwinCAT Neural Network Inference Engine das passende Format.

Warum ONNX?

Durch die Unterstützung von ONNX integriert Beckhoff die TwinCAT Machine Learning-Produkte in offener Art und Weise und **garantiert so flexible Workflows**. Während der Automatisierungsspezialist in TwinCAT 3 arbeiten kann, kann der Data Scientist mit seinen gewohnten Werkzeugen (PyTorch, Scikit-learn, …) arbeiten.

Der Einsatz von ONNX erleichtert das **arbeitsgruppenübergreifende Arbeiten** sowohl unternehmensintern als auch unternehmensübergreifend mit Partnern. Der Automatisierungsspezialist stellt dem Data Scientist aufgenommene Daten zur Verfügung. Der Data Scientist erstellt ein ML-Modell und übergibt sein Werk als ONNX-Datei an den Automatisierungsspezialist. Diese Datei enthält bereits alle Informationen, um das erstellte Modell in TwinCAT auszuführen.

Der **Offline-Test** von Modellen wird vereinfacht, da alle gängigen KI-Frameworks die ONNX-Datei laden und ebenso ausführen können.

Welche Software unterstützt ONNX?

Unterstützte Werkzeuge der ONNX Community können hier eingesehen werden: onnx.ai/supported-tools.

Darunter z. B. die **Frameworks:**

- PyTorch
- Keras/TensorFlow
- MXNet
- Scikit-learn
- …

Graph **Optimierer**

- ONNX Optimizer (https://github.com/onnx/onnx/blob/enable_noexpception_build/docs/Optimizer.md)

Graph **Visualisierer**

- Netron (https://github.com/lutzroeder/Netron)
- …

5.3.2 Beispiele zum ONNX-Export

Wie erstelle ich ONNX-Dateien?

Im Folgenden werden mehrere Varianten **exemplarisch** aufgezeigt, wie Sie aus unterschiedlichen Frameworks bestimmte Modelle als ONNX exportieren können. Die Beispiele haben keinen Anspruch auf Vollständigkeit und dienen nur der ersten Orientierung. Für eine ausführliche Dokumentation sei auf jene der entsprechenden Frameworks verwiesen.

Die aufgeführten Beispiele beschränken sich auf das Erstellen einer ONNX-Datei. Beispiele zur Konvertierung, um sie in TwinCAT nutzbar zu machen, finden Sie hier: Konvertieren von ONNX in XML und BML [▶ 65] sowie im verlinkten Samples ZIP-Archiv (siehe unten) im Ordner PythonAPI_mllib.

Übersicht verfügbarer Beispiele

Python Package	Modell-Typ	Option	Kommentar	Sample-Link
PyTorch	MLP Regressor			GoToPage [▶ 23]
Keras	MLP Regressor			GoToPage [▶ 24]
Scikit-learn	MLP Regressor			GoToPage [▶ 25]
Scikit-learn	MLP Classifier		ONNX-Graph muss adaptiert werden	GoToPage [▶ 26]
Scikit-learn	SVR			GoToPage [▶ 29]
Scikit-learn	SVC	decision_function_shape='ovo'		GoToPage [▶ 30]
Scikit-learn	k-means		Meta Key muss im ONNX eingetragen werden.	GoToPage [▶ 34]
Scikit-learn	PCA			GoToPage [▶ 36]
Scikit-learn	Decision Tree Classifier			GoToPage [▶ 38]
Scikit-learn	Decision Tree Regressor			GoToPage [▶ 38]
Scikit-learn	Extra Tree Classifier			GoToPage [▶ 40]
Scikit-learn	Extra Tree Regressor			GoToPage [▶ 40]
Scikit-learn	Extra Trees Classifier			GoToPage [▶ 43]
Scikit-learn	Extra Trees Regressor			GoToPage [▶ 42]
Scikit-learn	Random Forest Classifier			GoToPage [▶ 45]
Scikit-learn	Random Forest Regressor			GoToPage [▶ 45]
LightGBM	Random Forest Regressor		ONNX-Graph muss adaptiert werden	GoToPage [▶ 46]
Scikit-learn	Gradient Boosting Classifier			GoToPage [▶ 48]
Scikit-learn	Gradient Boosting Regressor			GoToPage [▶ 48]
Scikit-learn	Hist Gradient Boosting Classifier			GoToPage [▶ 51]
Scikit-learn	Hist Gradient Boosting Regressor			GoToPage [▶ 50]
XGBoost	XGBClassifier	Nicht alle Konfigurationen lassen einen ONNX-Export zu	Package Version <= 1.5.2 oder >= 1.7.4 notwendig	GoToPage [▶ 53]
XGBoost	XCBRegressor	Nicht alle Konfigurationen lassen einen ONNX-Export zu	Package Version <= 1.5.2 oder >= 1.7.4 notwendig	GoToPage [▶ 53]
LightGBM	LGBMRegressor	Nicht alle Konfigurationen lassen einen ONNX-Export zu		GoToPage [▶ 56]
LightGBM	LGBMClassifier		ONNX-Graph muss adaptiert werden	GoToPage [▶ 58]

Alle Samples können Sie hier als ZIP-Archiv herunterladen: https://infosys.beckhoff.com/content/1031/tf38x0_tc3_ML_NN_Inference_Engine/Resources/13668699915.zip

5.3.3 Konvertieren von ONNX in XML und BML

Im Folgenden wird beschrieben, wie Sie eine oder mehrere ONNX-Dateien in das Beckhoff-spezifische XML- bzw. BML-Format konvertieren können.

Warum ONNX in XML oder BML konvertieren?

Das ONNX-Format kann von der ML-Runtime auf der Zielplattform nicht direkt geladen werden. Daher ist im Engineering ein Konvertierungsschritt notwendig, bevor die ML-Beschreibungsdatei auf das Zielsystem übertragen wird. Die Erstellung einer XML-Datei hat den Vorteil, dass sie offen lesbar ist. So können während der Engineering-Phase alle Informationen in der XML über einen einfachen XML-Editor eingesehen werden. Die BML-Datei ist binär und damit der Inhalt nicht mehr einfach einsehbar. Außerdem wird eine BML-Datei von der ML-Runtime deutlich schneller geladen, sodass die *BML-Datei für die Auslieferung empfohlen* wird.

Welche TwinCAT-spezifischen Informationen können in der XML- bzw. BML-Datei eingetragen werden?

Erstellen von Multi-Engines

Es ist möglich, mehr als ein ML-Modell in eine Instanz des FB_MllPrediction [▶ 84] zu laden. Ein Wechsel zwischen den Modellen, im Folgenden in diesem Zusammenhang Engines genannt, ist ohne Latenz möglich. So kann zum Beispiel ein Set an Modellen für unterschiedliche Arbeitsbereiche der Maschine trainiert werden und während der Inferenzphase latenzfrei die korrekte Engine angesprochen werden.

Voraussetzung für das Zusammenführen von mehreren Modellen ist, dass die Modellstruktur, in der Beckhoff XML der Bereich <Configuration> [▶ 77], für alle Modelle identisch ist. Beispielsweise bedeutet das, dass nur MLPs zusammengeführt werden können, wenn die Struktur (Anzahl der Layer, Anzahl der Neuronen pro Layer und Aktivierungsfunktionen) gleich sind. Nur die Modellparameter, im Beispiel des MLPs also die Gewichte, dürfen unterschiedlich sein.

Vergleiche in der Beckhoff ML XML-Beschreibung: XML Tag Parameters [▶ 77].

Beim Zusammenführen werden also mehrere KI-Modelle mit identischer Struktur, aber unterschiedlichen Parametern in einer Beschreibungsdatei zusammengefasst. Die einzelnen Modelle (=Engines) werden über eine einzige Beschreibungsdatei in der ML Runtime geladen. Der Predict-Methode [▶ 92] ist beim Aufruf in der SPS jeweils die Engine-ID zu übergeben, welche adressiert werden soll. Über die PredictRef-Methode [▶ 93] kann die Engine über einen String adressiert werden. Ebenso steht eine GetEngineIdFromRef-Methode [▶ 90] zur Verfügung, um aus der Referenz die zugehörige ID zu finden.

Multi-Engines sind als Organisationseinheit zu verstehen. Es ist natürlich auch möglich, mehrere Instanzen eines FB_MllPrediction [▶ 84] in der SPS zu instanziieren und in jeden FB eine eigene Beschreibungsdatei zu laden.

Mindestversion des Treibers der ML-Runtime

Beim Konvertieren der Beschreibungsdatei werden automatisch zwei Einträge in die XML- bzw. BML-Datei gesetzt. Zum einen wird die Version der konvertierenden Komponente eingetragen und zum anderen wird eine „required version“ des ML-Runtime Treibers hinzugefügt. Beim Laden der Modelldatei in TwinCAT wird die „required version“ geprüft und bei Nicht-Bestehen der Abfrage eine Warnung ausgegeben.

Vergleiche in der Beckhoff ML XML-Beschreibung: XML Tag AuxilliarySpecifictaions [▶ 76].

Welche Anwendungsspezifischen Informationen können in der XML- bzw. BML-Datei eingetragen werden?

Eingangs- und Ausgangsskalierungen

Häufig werden die Eingänge des KI-Modells für den Trainingsprozess skaliert. Für die Inferenz ist diese Skalierung dann ebenfalls durchzuführen. Diese kann entweder von Hand in der SPS implementiert, oder direkt als Information in die XML- bzw. BML-Datei eingetragen werden. Sind die Skalierungseinträge gesetzt, wird die Skalierung automatisch durchgeführt. Zur Skalierung sind ein Scaling und ein Offset anzugeben. Es gilt:

```
y = x * Scaling + Offset
```

Werden skalierte Inputs für ein Modell verwendet, ist in der Regel auch eine Rückskalierung der Modell-Outputs notwendig. Daher ist neben der Input-Skalierung auch eine Output-Skalierung verfügbar.

Output Transformation nur für ausgesuchte Modelle

Während für alle KI-Modelle eine Input-Transformation möglich ist, können Output-Transformations nur für Modelle vom Typ Regression verwendet werden.

Vergleiche in der Beckhoff ML XML-Beschreibung: XML Tag AuxilliarySpecifictaions [▶ 76].

Modellname, Modellversion, Modellbeschreibung, …

Zur eindeutigen Identifizierung einer Modellbeschreibungsdatei ist es möglich, diverse Beschreibungen eines Modells hinzuzufügen. Diese sind jeweils als freier String verwendbar:

- Eine Modellversion
- Ein Modellname
- Eine Modellbeschreibung
- Ein Modell Autor
- Weitere optionale Tags

Vergleiche in der Beckhoff ML XML-Beschreibung: XML Tag AuxilliarySpecifictaions [▶ 76].

Freier Custom Attributes Bereich

Custom Attributes sind optional und können vom Anwender frei genutzt werden. Die Anzahl der Attribute und die Anzahl der XML-Tags sind nicht begrenzt. Die Attribute werden in der XML/BML typisiert, sodass die Einträge in der SPS wieder ausgelesen werden können. Dafür stehen die Methoden GetCustomAttribute_array, GetCustomAttribute_fp64, GetCustomAttribute_int64 und GetCustomAttribute_str zur Verfügung. Siehe auch Ausführliches Beispiel [▶ 96].

Beispiele für die Anwendung von Custom Attributes können sein:

- Angabe einer internen Version oder Identifikationsnummer
- Angabe des Input-Bereichs der einzelnen Features
- Beschreibung der Eingänge und Ausgänge
- …

Vergleiche in der Beckhoff ML XML-Beschreibung: XML Tag CustomAttributes [▶ 76].

Wie kann ich Dateien konvertieren und Informationen ergänzen?

Zur einfachen Integration des Konvertier- und Informationsmodellierungs-Schritts in Ihren Arbeitsablauf werden unterschiedliche Interfaces angeboten:

- Eine GUI im TwinCAT XAE „Machine Learning Model Manager [▶ 67]“
- Ein CLI „mllib_toolbox.exe [▶ 71]“
- Eine API als Python Package „beckhoff.toolbox [▶ 74]“

Nicht jedes Interface bietet alle oben beschriebenen Aktionen an. Das CLI ist limitiert auf grundlegende Anwendungen – hauptsächlich Konvertieren. Die Python API und das GUI bieten den größten Funktionsumfang.

5.3.3.1 GUI

Der TwinCAT 3 Machine Learning Model Manager ist das zentrale UI zum Bearbeiten von ML-Modellbeschreibungsdateien. Das Werkzeug ist in Visual Studio integriert und kann über die Menüleiste unter **TwinCAT > Machine Learning** geöffnet werden.

Voraussetzung an Visual Studio Version

Die graphische Oberfläche des TwinCAT 3 Machine Learning Model Managers ist kompatibel mit Visual Studio 2017 und 2019 sowie der TcXaeShell. Sollten Sie eine andere Version nutzen, können Sie die Oberfläche als Standalone Executable ausführen. Diese liegt unter *<TwinCATInstallDir>\3.1\Components\TcMachineLearning\ML_VS_Extension\ModelManagerStandalone.exe*.

Alternativ zur Bearbeitung der ML-Modellbeschreibungsdateien über die Oberfläche des TwinCAT 3 Machine Learning Model Manager können Sie auch ein Command Line Tool, siehe CLI [▶ 71], oder eine Python Library, siehe Python API [▶ 74] nutzen.

Konvertieren von ML-Modelldateien

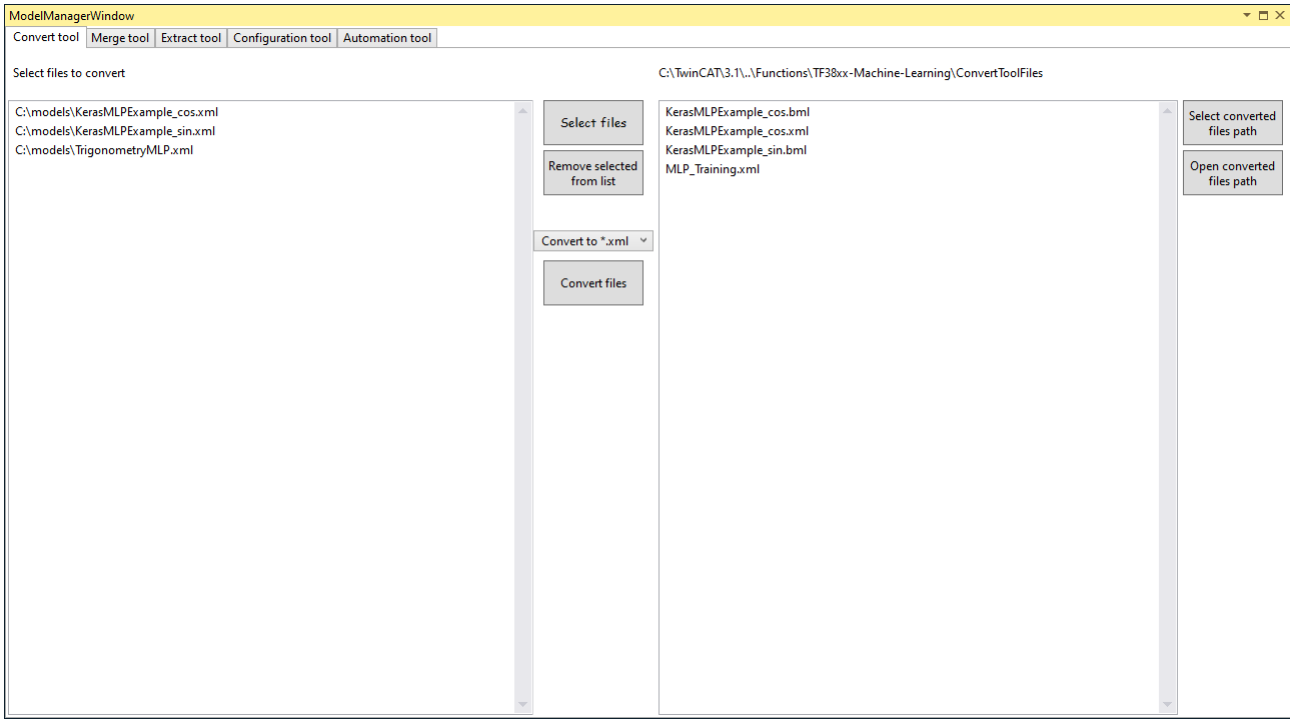
Unter dem Reiter **Convert tool** befindet sich ein Konvertierungswerkzeug für ML-Modellbeschreibungsdateien. Es können XML [▶ 75] und ONNX [▶ 62] Dateien selektiert und in XML oder BML [▶ 78] Format konvertiert werden.

HINWEIS

Konvertieren von Beckhoff BML zurück zu XML ist nicht vorgesehen

Ziel der Beckhoff BML ist es, den Inhalt als Binärdatei nicht frei lesbar zu repräsentieren. Daher ist der Konvertierungsvorgang von Beckhoff BML zu Beckhoff XML nicht vorgesehen.

Über **Select files** öffnet sich der File Browser und es können ML-Modellbeschreibungsdateien ausgewählt werden (multi-select möglich durch Strg-Click). Selektierte ML-Modelldateien werden auf der linken Seite mit ihrem Pfad und Dateinamen gelistet. Dateien können durch **Remove selected from list** wieder aus dieser entfernt werden.



Gelistete ML-Modellbeschreibungsdateien können in der linken Liste ausgewählt (auch hier ist ein multi-select mit Strg-Click möglich) und mit **Convert files** in das, über das Drop-Down-Menü, gewählte Format konvertiert werden. Die konvertierten Dateien werden im *converted file path* abgelegt. Default ist der Pfad *<TwinCATPath>\Functions\TF38xx-Machine-Learning\ConvertToolFiles*. Durch Click auf **Open converted file path** kann der *converted file path* im File Browser geöffnet werden.

Mit **Select converted files path** kann der Pfad verändert werden. Die Änderung bleibt auch nach Neustart des PCs erhalten.

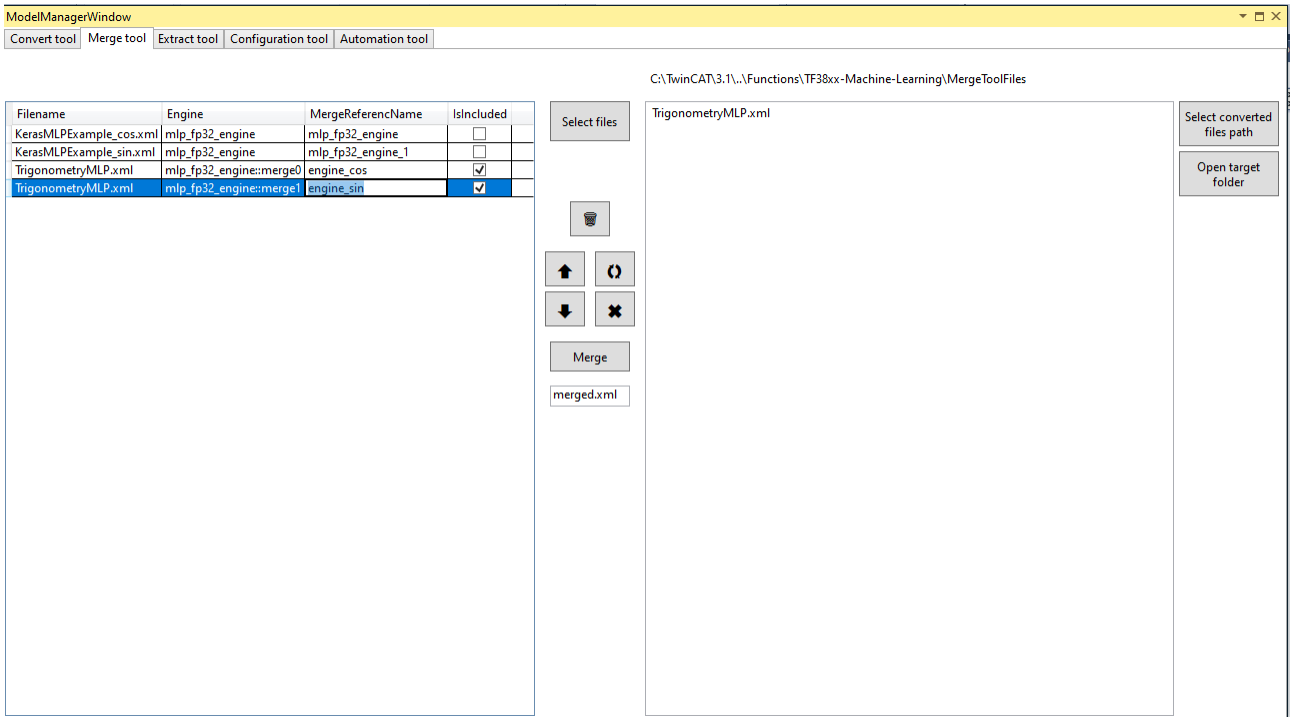
Erstellen einer Multi-Engine Beschreibung

Zum Erstellen einer Multi-Engine sind mit **Select files** mehrere ML-Modellbeschreibungen zu laden. In der Liste auf der linken Seite werden dann Engine-basiert die Einträge sichtbar. Sind in einer Beschreibungsdatei bereits mehrere Engines vorhanden, werden diese einzeln in der Liste aufgeführt.

Das Feld **MergeReferenceName** ist frei editierbar. Hier kann ein Referenzname der selektierten Engine eingetragen werden, sodass diese Engine in der SPS über diese Referenz adressierbar ist, vgl. PredictRef [▶ 93] und GetEngineIdFromRef [▶ 90]. Wird in der SPS nicht mit dem Referenznamen, sondern mit der EngineId gearbeitet gilt, dass die oberste Engine in der Liste die ID = 0 trägt und die folgenden entsprechend 1, 2, 3 usw.

Über die Checkbox **IsIncluded** ist einstellbar, ob eine selektierte Engine in die zusammengeführte Beschreibungsdatei übernommen werden soll oder nicht.

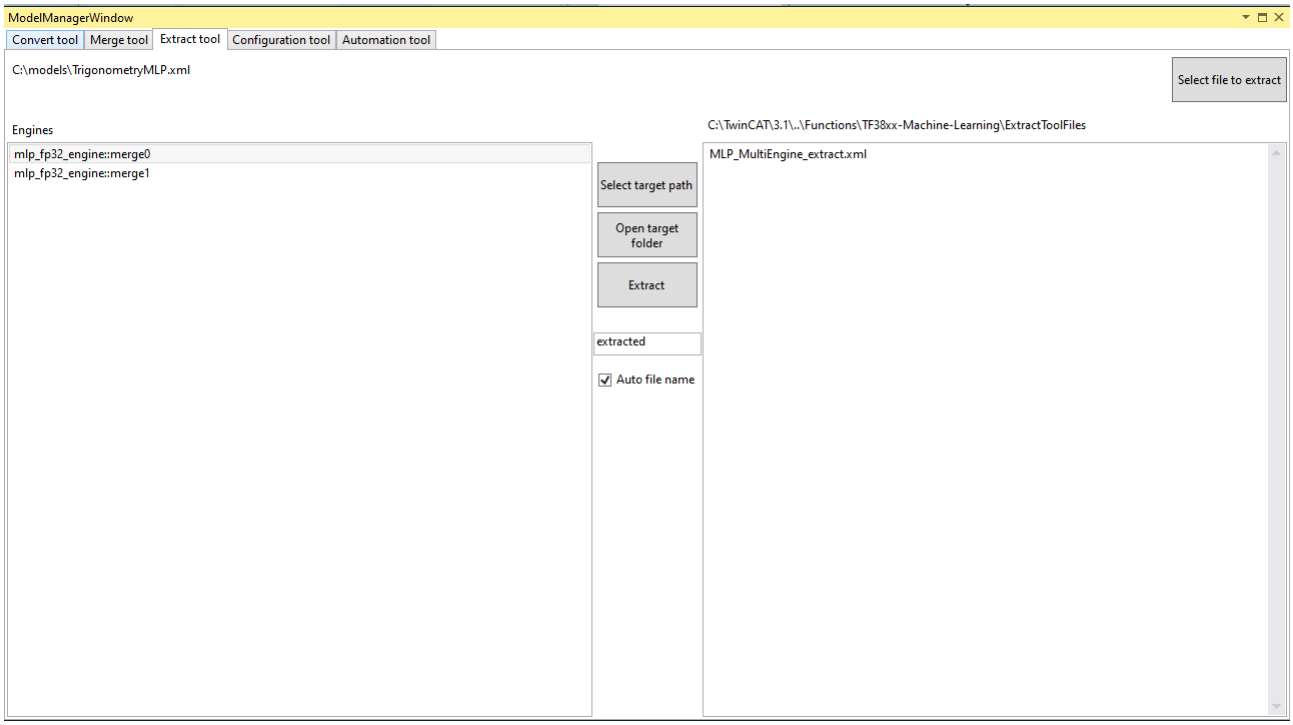
Im Mittelteil der Benutzeroberfläche können Engines in der Liste hinsichtlich ihrer Position manipuliert werden. Selektierte Engines können nach oben oder unten in der Liste verschoben werden (Pfeil hoch und runter). Hier sind beliebig viele Engines gleichzeitig auswählbar. Ebenso kann über das Kreuz-Symbol eine beliebige Anzahl von Engines gleichzeitig bzgl. der Checkbox **IsIncluded** manipuliert werden. Sind zwei Engines selektiert, können diese den Platz tauschen, indem das kreisförmige Doppelpfeil-Symbol verwendet wird. Engines können aus der Liste mit dem Papierkorb-Symbol entfernt werden.



In dem Textfeld im Mittelteil kann der Name der zusammengeführten ML-Beschreibungsdatei eingetragen werden. Mit **Merge** wird die Datei erzeugt und im Dateipfad *<TwinCATPath>\Functions\TF38xx-Machine-Learning\MergeToolFiles* abgelegt. Mit **Select target path** kann der Pfad verändert werden.

Extrahieren von Multi-Engine Beschreibungen

Mit dem Extract tool ist es möglich, zusammengeführte Beschreibungsdateien wieder zu trennen. Über **Select file to extract** kann eine ML-Modellbeschreibungsdatei geladen werden. In der Liste auf der linken Seite erscheinen alle darin enthalten Engines. Wird eine Engine selektiert, kann diese durch **Extract** in eine eigenständige ML-Beschreibungsdatei überführt werden. Der Name der generierten neuen Datei ist im Textfeld einzutragen. Ist die Checkbox **Auto file name** aktiv, wird der Originaldateiname um den String im Textfeld ergänzt. Ist die Checkbox inaktiv, wird nur der String im Textfeld als neuer Dateiname genutzt. Die neu erzeugte Datei wird im Dateipfad *<TwinCATPath>\Functions\TF38xx-Machine-Learning\ExtractToolFiles* abgelegt. Mit **Select target path** kann der Pfad verändert werden.

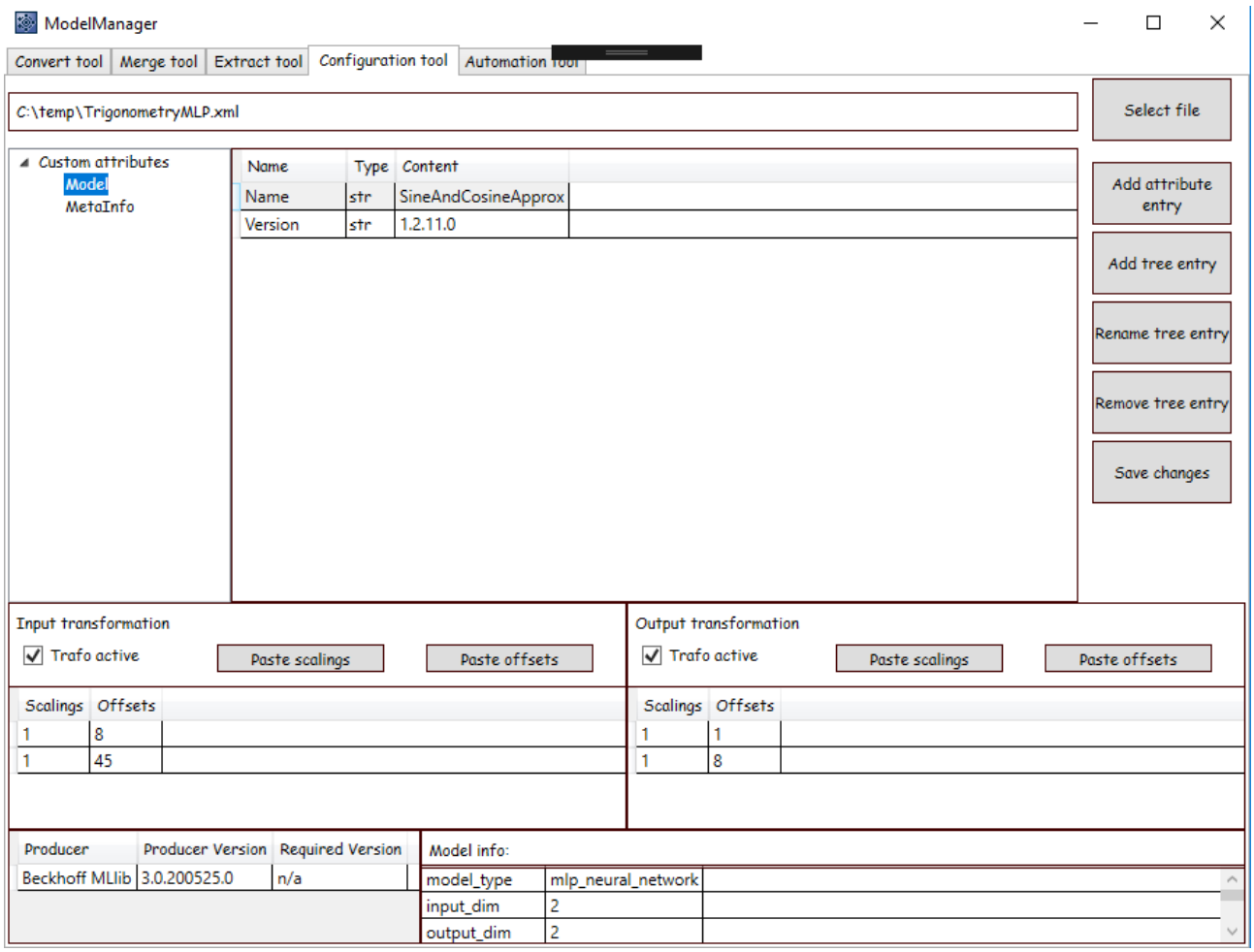


Metadaten zum Modell erstellen

Über den Reiter **Configuration tool** wird der Konfigurator angezeigt für:

- Custom attributes [▶ 76]
- Product Version und Target Version Information [▶ 76]
- Input- und Output-Skalierung [▶ 76]

Eine ML-Beschreibungsdatei kann über **Select file** ausgewählt und dann bearbeitet werden. Nach Bearbeitung wird mit **Save changes** die Originaldatei überschrieben.



Die Bearbeitung der **Custom Attributes** erfolgt über die Buttons:

- **Add attribute entry**: Fügt dem ausgewählten Tree item ein Attribut hinzu. Das Tree item ist in der linken Liste zu selektieren.
  - Wird ein Attribut angelegt, ist der Name, der Datentyp und der Wert bzw. die Werte anzugeben.
  - Löschen von Attributen erfolgt durch Selektieren des Attributes und Drücken der Taste **Entfernen**.
- **Add tree entry**: Fügt ein Tree item unter dem selektierten Tree item (als Sub Tree Item) an.
- **Rename tree entry**: Das selektierte Tree Item kann umbenannt werden.
- **Remove tree entry**: Das selektierte Tree Item, inkl. der Sub Tree Items, wird gelöscht.

Durch Aktiveren der Checkbox **Trafo active** lässt sich die Editierungsfläche für **Input- und Output Transformationen** freischalten. Je nach Anzahl der Ein- und Ausgänge wird eine entsprechende Anzahl von Zeilen angeboten, in denen jeweils Scaling und Offset einzutragen ist, vgl. XML Tag AuxilliarySpecifictaions [▶ 76]. Über die Buttons **Paste Scaling** und **Paste Offset** kann aus einer Liste von Zahlen aus der Zwischenablage ein Wert als Offset oder Scaling eingefügt werden. Die Zahlenfolge kann dabei durch Komma, Semikolon oder Leerzeichen getrennt sein. Lediglich die Anzahl der Zahlen der Liste muss zu der Anzahl der Ein- bzw. Ausgänge passen.

Die **Producer und Target Version Information** wird automatisch vom TwinCAT 3 ML Model Manager gesetzt. Die Target Version wird auf Basis des verwendeten Feature Sets der Modellbeschreibungsdatei automatisch ermittelt. Wird eine ältere ML Runtime Version genutzt, um diese Modelldatei zu laden, erfolgt eine Warnmeldung bei der Ausführung der Configure-Methode.

Im unteren rechten Teil des Fensters wird die Anzahl der **Ein- und Ausgänge** des Modells sowie der **Modell-Typ** angezeigt. Dies kann nicht editiert werden und dient nur zur Information.

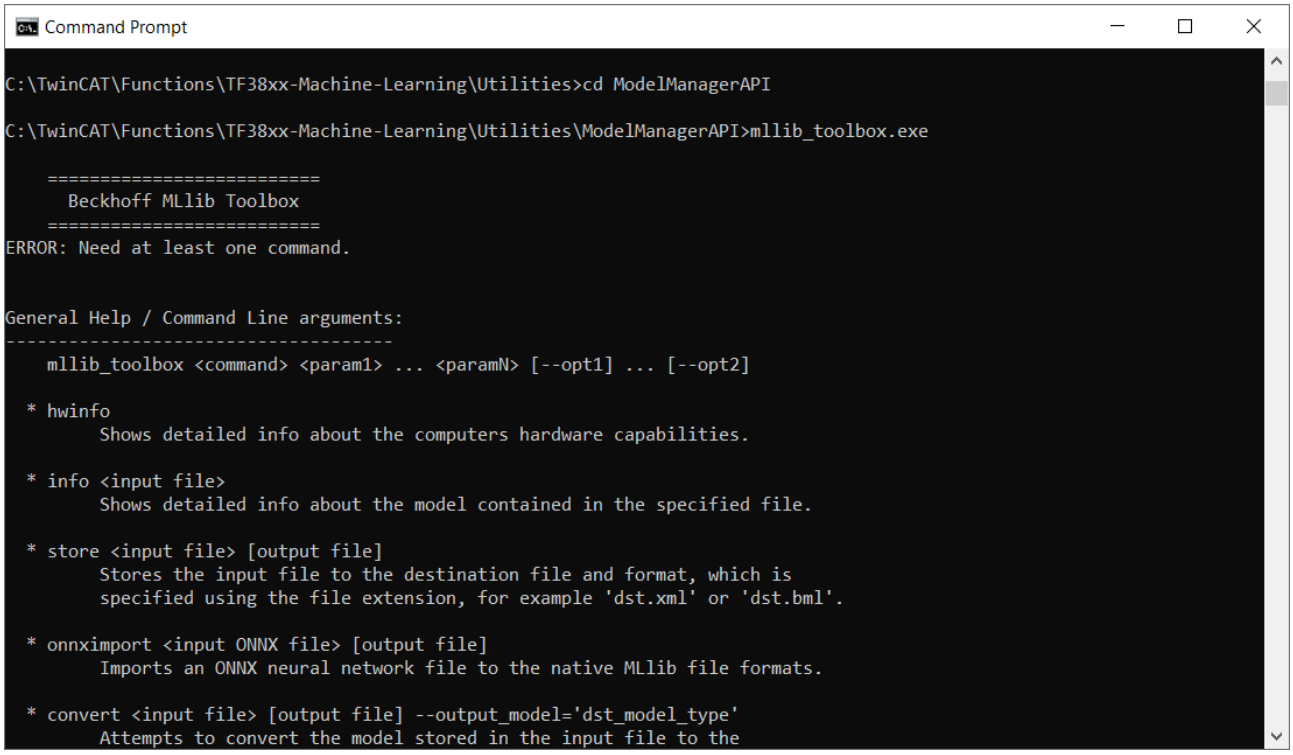
5.3.3.2 CLI

Zur programmatischen Bearbeitung von ML-Beschreibungsdateien, z. B. dem Konvertieren von ONNX zu XML oder BML, steht neben dem GUI und dem Python Package ein Commandline Tool zur Verfügung: **mllib_toolbox.exe**

Die Executable ist unter <TwinCatInstallDir>\Functions\TF38xx-Machine-Learning\Utilities\ModelManagerAPI zu finden.

Nutzung der Executable

Die Nutzung der `mllib_toolbox.exe` kann z. B. aus dem Command Prompt heraus erfolgen. Die eingebaute Hilfe wird durch Ausführung der exe ohne Argument ausgegeben.



Die mllib_toolbox.exe können Sie auf ihrem PC auch an eine beliebige andere Stelle verschieben/kopieren. Die exe verwendet die Path-Umgebungsvariable, welche auf die mllib_um.dll verweist (default <TwinCatInstallDir>\3.1\Components\Base\Addins\TcMLExtension).

Konvertieren von ONNX-Dateien

Das Konvertieren von ONNX-Dateien erfolgt durch die Methode `onnximport`

```
mllib_toolbox.exe onnximport ".\decision tree\decisiontree-classifier.onnx" ".
\decision tree\decisiontree-classifier.bml"
```

Alternativ kann auch ein Argument verwendet werden (--xml oder –bml).

```
Mllib_toolbox onnximport myOnnxFile.onnx --xml
```

```
Mllib_toolbox onnximport myOnnxFile.onnx -bml
```

Der Konvertier-Befehl schreibt automatisch die „required Version“ (Mindestversion des ML-Runtime Treibers) in die generierte XML bzw. BML.

Merge und Extract von Engines

Zum Erstellen von Multi-Engines nutzen Sie den `merge`-Befehl. Folgender Befehl führt die beiden genannten XML-Dateien zusammen zu einer Multi-Engine mit 2 Engines. Dabei wird die letzte genannte Datei überschrieben.

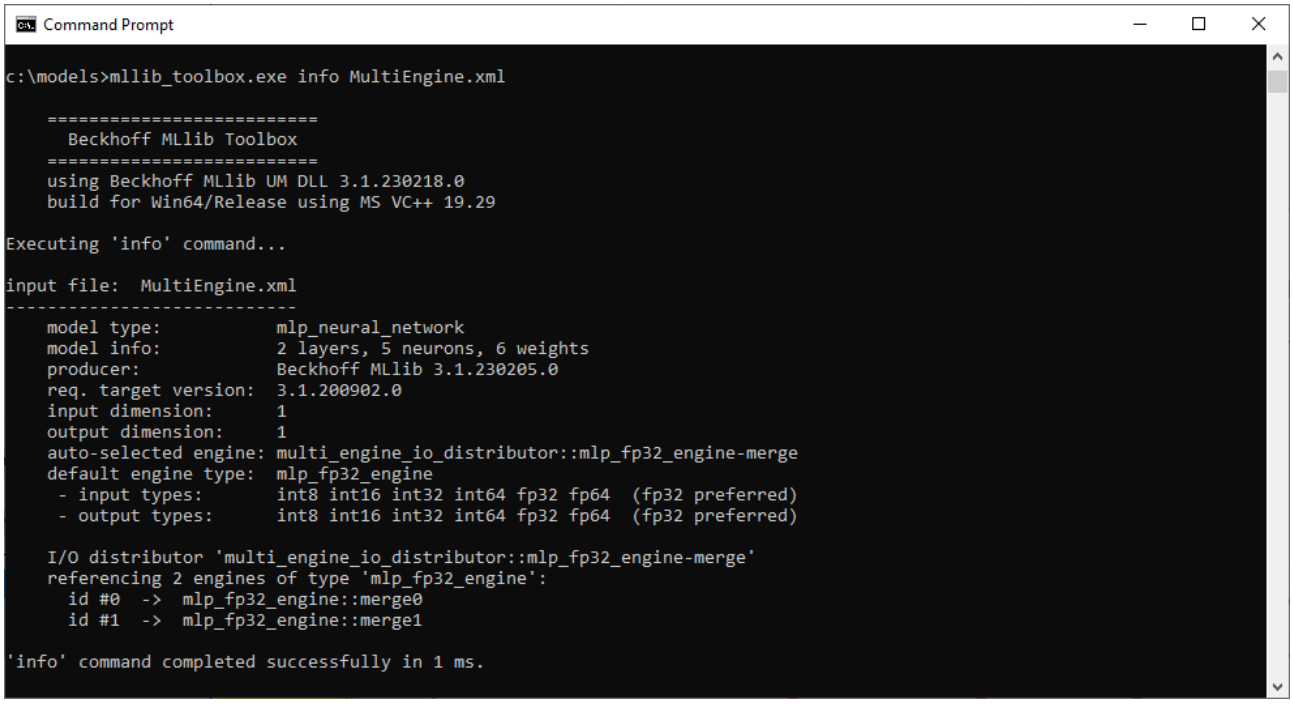
```
mllib_toolbox.exe merge KerasMLPExample_cos.xml KerasMLPExample_sin.xml
```

Es kann auch eine neue Zieldatei angegeben werden. Diese kann sowohl eine XML- als auch eine BML-Datei sein.

```
mllib_toolbox.exe merge KerasMLPExample_cos.xml KerasMLPExample_sin.xml MultiEngine.bml
```

Die Anzahl der mit einem Kommando zu kombinierenden Dateien ist nicht begrenzt.

Um Engines aus einer Beschreibungsdatei mit mehreren Engines zu extrahieren, nutzen Sie die Extract Methode. Prüfen Sie zunächst mit dem Info-Befehl, wie viele Engines in der Datei vorhanden- und wie sie benannt sind.



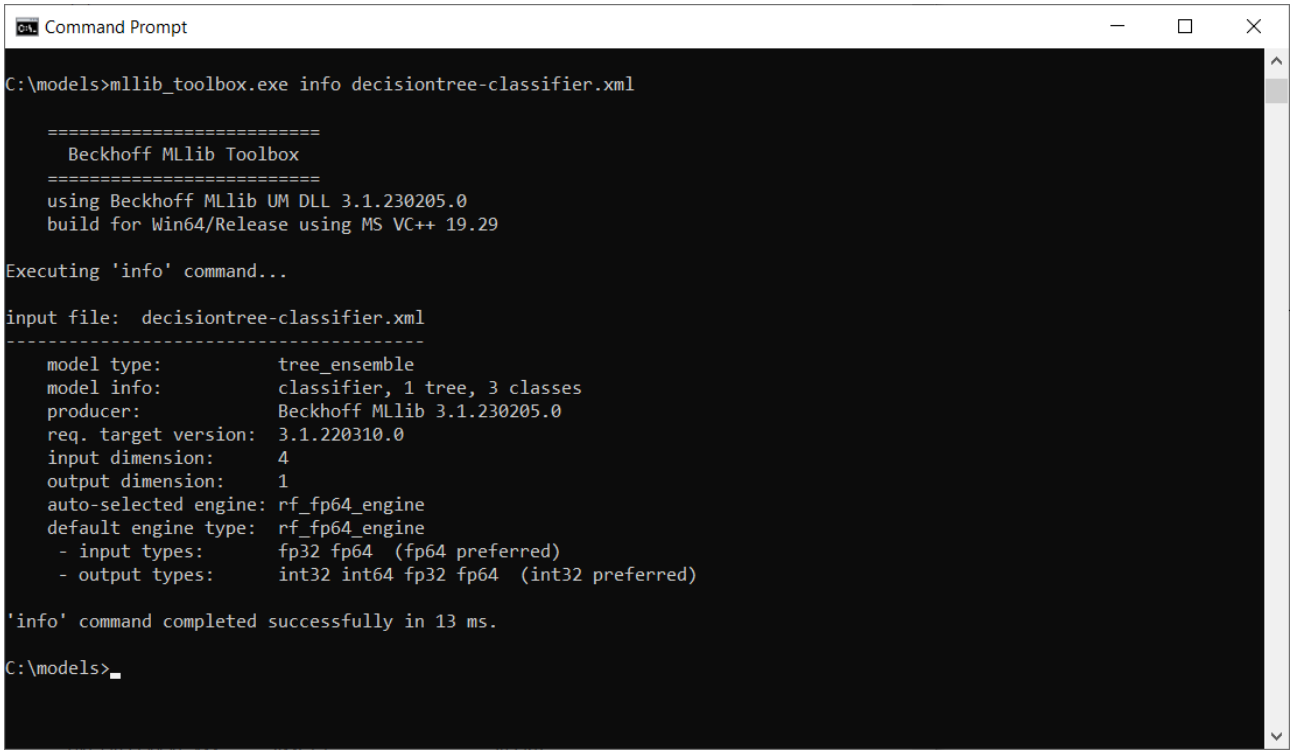
In obiger Abbildung sehen Sie zwei Engines mit den Bezeichnungen mlp_fp32_engine::merge0 bzw. merge1. Zum Extrahieren der ersten Engine in die Zieldatei Extracted.xml rufen Sie auf:

```
mllib_toolbox.exe extract MultiEngine.xml?eng='mlp_fp32_engine::merge0' Extract.xml
```

Informationen einer ML-Modellbeschreibungsdatei anzeigen

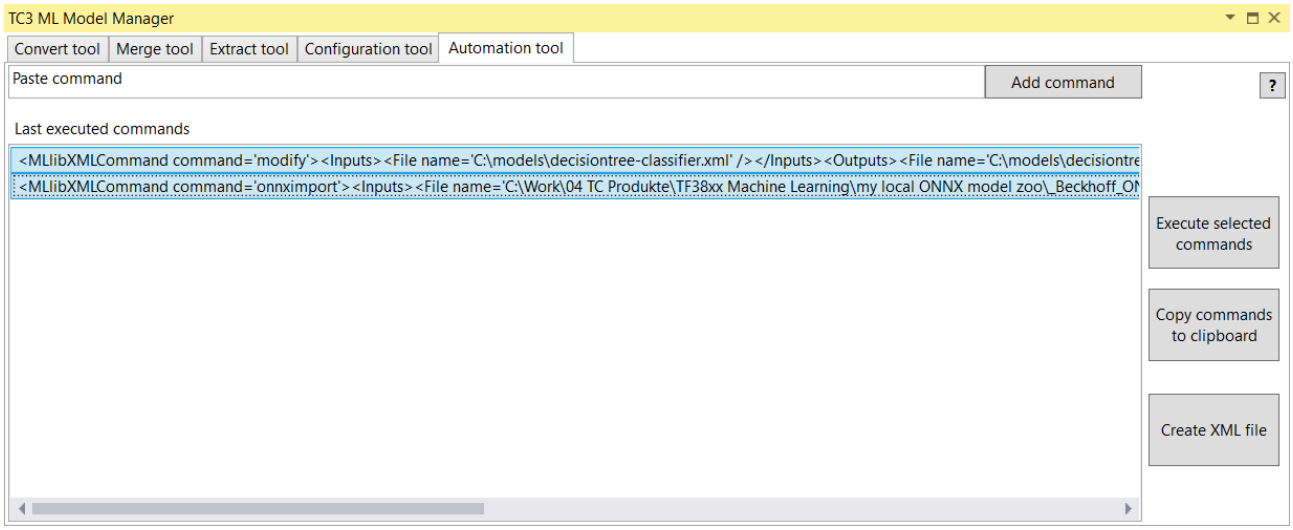
Um schnell zu prüfen, was für ein Modell in einer Beschreibungsdatei beschrieben ist, kann der `info`-Befehl genutzt werden. Dieser kann sowohl ONNX-Dateien als auch XML- und BML-Dateien analysieren.

```
mllib_toolbox.exe info decisiontree-classifier.xml
mllib_toolbox.exe info decisiontree-classifier.bml
mllib_toolbox.exe info decisiontree-classifier.onnx
```



Ausführen einer Anweisungsliste

Sie können im Machine Learning Model Manager Operationen graphisch durchführen. Jeder Operation, die Sie in einer Session durchführen, wird im Tab „Automation Tool“ mitgeschrieben als XML-Kommando.



In obiger Abbildung ist beispielsweise ein Konvertier-Befehl und ein Befehl zum Eintragen von Custom Attributes zu sehen. Markieren Sie die Befehle, die Sie exportieren möchten, und wählen Sie **Create XML file**. Achten Sie dabei auf die Reihenfolge, in der Sie die Befehle im Machine Learning Model Manager selektieren. Die Selektionsreihenfolge bestimmt die Reihenfolge der Befehle in der exportierten Datei.

Das erzeugte XML-File können Sie zum Reproduzieren der Befehlsfolge nutzen über

```
mllib_toolbox.exe rawxml AutomationToolExport.xml
```

Custom Attributes, Scalings und Model description

Diese Eigenschaften sind im CLI nicht verfügbar. Nutzen Sie dafür das Python Package [▶ 74] oder den TwinCAT Machine Learning Model Manager [▶ 67]. Das CLI beschränkt sich auf Basisfunktionalitäten.

5.3.3.3 Python API

Installation des Python Package

Das Python Package liegt als whl-Datei im Ordner <TwinCatInstallDir>\Functions\TF38xx-Machine-Learning\Utilities\ModelManagerAPI\PythonPackage.

Um das Package zu installieren, nutzen Sie `pip install` <TwinCatInstallDir>\Functions\TF38xx-Machine-Learning\Utilities\ModelManagerAPI\PythonPackage\<whl-file-name>. Der Ordner enthält ggf. unterschiedliche Versionen des Package (nur, wenn Sie ein neues Setup über ein altes TwinCAT Machine Learning Setup installiert haben).
Achten Sie darauf, immer die aktuelle Version zu nutzen.

```
pip  install  "C:\TwinCAT\Functions\TF38xx-Machine-
Learning\Utilities\ModelManagerAPI\PythonPackage\beckhoff_toolbox-3.1.230205-py3-none-any.whl"
```

Verwenden der Beckhoff Toolbox

Zur Beschreibung der einzelnen Punkte wie Custom Attributes und Model Description siehe: Konvertieren von ONNX in XML und BML [▶ 65].

Folgender Quellcode ist auch als py-File verfügbar. Siehe dazu: https://infosys.beckhoff.com/content/1031/tf38x0_tc3_ML_NN_Inference_Engine/Resources/13668699915.zip im Verzeichnis PythonAPI_mllib.

Das Package wird geladen mit

```
import  beckhoff.toolbox as  tb
```

Konvertieren einer ONNX-Datei in XML

```
tb.onnximport("../decision tree/decisiontree-regressor.onnx", "../decision tree/decisiontree-
regressor.xml")
```

```
tb.onnximport("../decision tree/decisiontree-regressor.onnx", "../decision tree/decisiontree-
regressor.bml")
```

Anzeigen von **Modellinformationen**

```
tb.info("../decision  tree/decisiontree-regressor.onnx")
```

```
tb.info("../decision  tree/decisiontree-regressor.xml")
```

Custom Attributes hinzufügen

```
new_ca = { 'nID' : -34234, 'bTested' : True, 'fNum' : 324.3E-12, 'AnotherTreeItem' : { 'fPi' : 3.134
12, 'bFalseFlag' : False }}
tb.modify_ca("../decision tree/decisiontree-regressor.xml","../decision tree/decisiontree-regressor-
custom.xml", new_ca)
```

Model Description (Name, Version, Autor usw. eines Modells) hinzufügen

```
model_description = {
    "new_version" : "2.3.1.0",
    "new_name" : "CurrentPreControlAxis42",
    "new_desc":"This is the most awesome model to control Axis42",
    "new_author":"Max",
    "new_tags":"awesome, ingenious, astounding",
}
tb.modify_md("../decision tree/decisiontree-regressor-custom.xml", "../decision tree/decisiontree-
regressor-md2.xml",
             **model_description)
```

Input- und Output Transformations hinzufügen

> **Output Transformations nur für ausgesuchte Modelle**
> Während für alle KI-Modelle eine Input-Transformation möglich ist, können Output-Transformations nur für Modelle vom Typ Regression verwendet werden.

```
# add input / output scalings (input-output-transformations)
tb.modify_iot("../decision tree/decisiontree-regressor-md.xml", "../decision tree/decisiontree-
regressor-iot.xml",
              tb.iot_scaled_offset([1.2,3.4,1.0,1.1,1.0,1.0,1.0,1.0,1.0,1.0],
```

```
[1.2,3.4,1.0,1.1,1.0,1.0,1.0,1.0,1.0,1.0]),
              None)   # no output transformation
```

Die Funktion `iot_scaled_offset` erwartet eine Liste von offsets und sclaings. Für jeden Input bzw. Output ist ein Wert anzugeben.

```
def iot_scaled_offset(offsets : list, scalings : list):
```

Im obigen Beispiel ist die Anzahl der Eingänge 10, daher wird eine Liste von 10 Elementen übergeben. Für decision trees ist keine Output Transformation verfügbar, daher wird hier ein None übergeben.

Multi-Engines erzeugen

Führen Sie mehrere Modelle zusammen zu einem XML-File.

```
# # Create a Multi-Engine
# merge two XML files - output file is Merged.xml
tb.merge(['KerasMLPExample_sin.xml','KerasMLPExample_cos.xml'],'Merged.xml')

# merge two specific engines from two XML files
tb.merge([tb.input_engine('Merged.xml','mlp_fp32_engine::merge1'),
          tb.input_engine('KerasMLPExample_cos.xml','mlp_fp32_engine')],
         'DoubleMerged.xml')

# merge two specific engines from two XML files and provide a reference name for both engines in the
 target file
tb.merge([tb.input_engine('KerasMLPExample_sin.xml','mlp_fp32_engine', 'sine'),
          tb.input_engine('KerasMLPExample_cos.xml','mlp_fp32_engine', 'cosine')],
         'MergedRef.xml')

# extract a specific engine from a Multi-Engine file
tb.extract(tb.input_reference('MergedRef.xml', 'sine'),'Extract.xml')
```

Test-Predict der ML-Runtime in Python

Erzeugen einer XML und anschließend Predict-Aufruf der ML-Runtime. Die ML-Runtime ist als DLL kompiliert und wird aus Python heraus als User-Mode Prozess aufgerufen. Es ist keine TwinCAT Runtime erforderlich.

```
tb.onnximport("../decision tree/decisiontree- regressor.onnx", "../decision tree/decisiontree-
 regressor.xml")
inp_file_xml = "../decision tree/decisiontree-regressor.xml"
# define input and output format and input values (10 in, 1 out)
prediction = [{'input_type': 'fp64', 'output_type': 'fp64', 'input': [-1.0,-1.0,1.0,1.4,-1.0,-1.0,1.
0,1.4,4.2,4.2]}]
# call predict method of Python-ML-Runtime
out = tb.predict(inp_file_xml,prediction)
print(out)
```

Weiteres Beispiel für einen Classifier.

```
tb.onnximport("../decision tree/decisiontree-classifier.onnx", "../decision tree/decisiontree-
classifier.xml")
inp_file_xml = "../decision tree/decisiontree-classifier.xml"
# define input and output format and input values (4 in, 1 out)
prediction = [{'input_type': 'fp64', 'output_type': 'int32', 'input': [-1.0,-1.0,1.0,1.4]}]
# call predict method of Python-ML-Runtime
out = tb.predict(inp_file_xml,prediction)
print(out)
```

5.3.3.4 Beschreibung des Beckhoff-spezifischen XML- und BML-Formats

5.3.3.4.1 Beckhoff ML XML

Einführung zur Beckhoff ML XML

Das Beckhoff-spezifische XML-Format zur Repräsentation von trainierten Machine Learning Modellen bildet eine Kernkomponente der TwinCAT Machine Learning Inference Engine und TwinCAT Neural Network Inference Engine. Die Datei wird aus einer ONNX-Datei [▶ 62] über den TC3 Machine Learning Model Manager [▶ 67] oder die Machine Learning Toolbox [▶ 71] oder das bereitgestellte Python Package [▶ 74] erstellt.

Im Gegensatz zum ONNX, kann die XML-basierte Beschreibungsdatei TwinCAT-spezifische Eigenschaften abbilden. Die XML gewährleistet einen erweiterten Funktionsumfang des TwinCAT Machine Learning-Produkts – siehe bspw. das Konzept der Multi-Engines [▶ 77]. Andererseits stellt sie ein nahtloses Zusammenarbeiten zwischen Erzeuger und Anwender der Beschreibungsdatei sicher – vgl. Ein- und Ausgangstransformationen [▶ 76] und Custom Attributes [▶ 76].

Im Folgenden werden wesentliche Bereiche der Beckhoff ML XML beschrieben. Dies dient dem Verständnis der darin bereitgestellten Funktionen.

XML Tag <MachineLearningModel>

Obligatorisches Tag mit 2 obligatorischen Attributen. Das Tag wird automatisch erzeugt und darf nicht manipuliert werden.

Beispiel:

```
<MachineLearningModel modelName="Support_Vector_Machine" defaultEngine="svm_fp64_engine">
```

Das Attribut `modelName` kann in der PLC ausgelesen werden über die Methode GetModelName [▶ 91]. Über den Modellnamen wird der Modelltyp identifiziert, welcher geladen werden soll. Das Attribut kann z. B. die Werte `support_vector_machine` oder `mlp_neural_network` annehmen.

Das Attribut modelName in diesem Tag ist nicht zu verwechseln mit dem Attribut str_modelName aus <ModelDescription>.

XML Tag <CustomAttributes>

Das Tag CustomAttributes ist optional und kann vom Anwender frei genutzt werden. Die Tiefe des Baums und die Anzahl der Attribute ist nicht begrenzt. Die Erstellung kann über den TC3 Machine Learning Model Manager erfolgen. Ebenso kann die XML in diesem Bereich manuell editiert werden.

Attribute können in der PLC ausgelesen werden über die Methoden GetCustomAttribute_array [▶ 87], GetCustomAttribute_fp64 [▶ 88], GetCustomAttribute_int64 [▶ 89] und GetCustomAttribute_str [▶ 89]. In der XML ist die Typisierung durch die Prefixe str_, int64_, fp64_ usw. gegeben.

Beispiel:

```
<CustomAttributes>
   <Model str_Name="TempEstimator" str_Version="1.2.11.0" />
   <MetaInfo arrfp64_InputRange="0.10000000000000001,0.90000000000000002" int64_TheAnswer="42" />
</CustomAttributes>
```

Hier wird ein Modell mit dem Namen „TempEstimator“ in der Version 1.2.11.0 angelegt. Als weiterführende Information wird ein Array und ein Integer Wert bereitgestellt. Exemplarischer Code zum Auslesen der CustomAttributes kann im Bereich Beispiele [▶ 96] heruntergeladen werden.

XML Tag <AuxilliarySpecifications>

Der Bereich AuxilliarySpecifications ist optional und untergliedert in die Childs <PTI> und <IOModification**>.**

Beispiel:

```
<AuxiliarySpecifications>
   <PTI str_producer="Beckhoff MLlib Keras Exporter" str_producerVersion="3.0.200525.0 str_requiredV
ersion="3.0.200517.0d"/>
   <ModelDescription str_modelVersion="2.3.1.0" str_modelName="CurrentPreControlAxis42" str_modelDes
c="This is the most awesome model to control Axis42" str_modelAuthor="Max" str_modelTags="awesome,in
genious,astounding" />
   <IOModification>
      <OutputTransformation str_type="SCALED_OFFSET" fp64_offsets="0.48288949404162623" fp64_scaling
s="1.4183887951105305"/>
   </IOModification>
</AuxiliarySpecifications>
```

<PTI>

PTI steht für „Product Version und Target Version Information“. Hier wird angegeben, mit welchem Werkzeug die XML erstellt wurde und welche Version das erstellende Werkzeug zum Zeitpunkt der XML-Erzeugung trug.

Ebenso kann, über das Attribut str_requiredVersion, eine Mindestversion der ausführenden ML Runtime angegeben werden. Ist das Attribut nicht gesetzt, gilt die Abfrage als bestanden. Ist das Attribut gesetzt, gilt die Abfrage als bestanden, wenn die ML Runtime Version größer oder gleich der required Version ist. Ist die Abfrage nicht bestanden, d. h. ist die genutzte Version der ML Runtime kleiner als die vorausgesetzte Version, wird bei Ausführung der Configure-Methode eine Warnung ausgegeben.

<IOModification>

Werden Eingänge oder Ausgänge des gelernten Modells in der Trainingsumgebung skaliert, können die genutzten Skalierungsparameter direkt in die XML-Datei integriert werden, sodass TwinCAT die Skalierung automatisch in der ML Runtime durchführt.

Die Skalierung erfolgt durch y = x * Scaling + Offset.

<ModelDescription>

Attribute an diesem optionalen Tag beschreiben

- die Modell-Version str_modelVersion
- den Modell-Namen str_modelName
- die Modell-Beschreibung str_modelDesc
- den Autor des Modells str_modelAuthor
- weitere optionale Tags str_modelTags

XML Tag <Configuration>

Der obligatorische Bereich *Configuration* beschreibt die Struktur des geladenen Modells.

Beispiel SVM

```
<Configuration str_operationType="SVM_TYPE_NU_REGRESSION" fp64_cost="0.1" fp64_nu="0.3" str_kernelFu
nction="KERNEL_FN_RBF" fp64_gamma="1.0" int64_numInputAttributes="1"/>
```

Beispiel MLP

```
<Configuration int_numInputNeurons="1" int_numLayers="2" bool_usesBias="true">
   <MlpLayer1 int_numNeurons="3" str_activationFunction="ACT_FN_TANH"/>
   <MlpLayer2 int_numNeurons="1" str_activationFunction="ACT_FN_IDENTITY"/>
</Configuration>
```

Eine Konfiguration existiert nur einmalig und wird automatisch generiert.

XML Tag <Paramaters>

Der obligatorische Bereich Parameters konkretisiert das geladene Modell mit der beschriebenen <Configuration>. Hier werden die gelernten Parameter des Modells abgelegt, z. B. die Gewichte der Neuronen.

Im Standardfall, d. h. es ist ein gelerntes Modell in einer XML beschrieben, ist der Tag <Parameters> nur einmal in der XML vorhanden.

```
<Parameters str_engine="mlp_fp32_engine" int_numLayers="2" bool_usesBias="true">
```

Über den Machine Learning Model Manager können mehrere Modelle mit identischer <Configuration> zusammengeführt werden, sodass beide Modelle in einer einzelnen XML beschrieben sind. Die Parametersätze können dann durch die *Engines* unterschieden werden, welche als Attribut für jeden Parameter-Tag angegeben ist.

Beispiel:

```
<Parameters str_engine="mlp_fp32_engine::merge0" int64_numLayers="2" bool_usesBias="true">
   …
</Parameters>
<Parameters str_engine="mlp_fp32_engine::merge1" int64_numLayers="2" bool_usesBias="true">
   …
</Parameters>
<IODistributor str_distributor="multi_engine_io_distributor::mlp_fp32_engine-
merge" str_engine_type="mlp_fp32_engine" int64_engine_count="2">
   <Engine0 str_engine_name="merge0" str_reference="sin_engine" />
   <Engine1 str_engine_name="merge1" str_reference="cos_engine" />
</IODistributor>
```

Hier wurden zwei MLPs mit identischer *Configuration* zusammengeführt. Die erste Engine trägt die ID 0, den internen Namen „mlp_fp32_engine::merge0“ und kann vom Nutzer über die Referenz „sin_engine“ angesprochen werden. Die zweite Engine trägt die ID 1, den internen Namen „mlp_fp32_engine::merge1“ und die Referenz „cos_engine“.

Die ID der Engines wird von Null angefangen fortlaufend um den Wert Eins erhöht. Die Referenz ist ein String, welcher im Model Manager beim *Merge* angegeben werden kann.

Werden mehrere Engines in einer XML zusammengefasst, werden alle Engines in der ML Runtime geladen und stehen zur Inferenz zur Verfügung. Der Predict-Methode [▶ 92] ist beim Aufruf die Engine-ID zu übergeben, welche genutzt werden soll. Über die PredictRef-Methode [▶ 93] kann die Referenz der Engine übergeben werden. Ebenso steht eine GetEngineIdFromRef-Methode [▶ 90] zur Verfügung, um aus der Referenz die zugehörige ID zu finden. Ein Wechsel zwischen den Engines ist ohne Latenz möglich.

Ein Beispiel zur Nutzung von Multi-Engines in der PLC ist im Bereich Beispiele vorhanden.

5.3.3.4.2 Beckhoff ML BML

Das BML-Format ist eine binäre Repräsentation der XML-basierten ML Beschreibungsdatei. Dadurch ist das Format nicht offen einsehbar und die Dateigröße ist im Vergleich zu ONNX und XML geringer. Dadurch ist der Ladevorgang des Modells auch deutlich schneller als beim Laden einer XML.

Eine BML-Datei kann über den TC3 Machine Learning Model Manager [▶ 67] aus einer XML- oder einer ONNX-Datei erzeugt werden. Der Weg zurück von einer BML-Datei zu einer XML-Datei ist **nicht** vorgesehen.

5.4 Dateimanagement der ML-Beschreibungsdateien

Dateimanagement auf dem Engineering PC (XAE)

Konvertierung, Bearbeitung der ONNX, XML und BML

Der Machine Learning Model Manager [▶ 67] dient als zentrales Werkzeug zur Bearbeitung und Konvertierung von Machine Learning Modellen. Wird eine Datei geladen und bearbeitet, so wird die resultierende Datei im Standardfall in den folgenden Ordnern abgelegt – je nach durchgeführter Aktion:

- \Functions\TF38xx-Machine-Learning\ConvertToolFiles
- \Functions\TF38xx-Machine-Learning\ExtractToolFiles
- \Functions\TF38xx-Machine-Learning\MergeToolFiles

Der Default-Ordner kann jeweils im Model Manager angepasst werden.

Wird statt des GUI-basierten Machine Learning Model Managers die Machine Learning Toolbox [▶ 71] genutzt, wird die neu erzeugte Datei im aktiven Pfad abgelegt, solange kein anderer Pfad konkret benannt wurde.

Einbinden eines Modells in eine TwinCAT Solution

Wird eine BML- oder XML-Beschreibungsdatei in einer TwinCAT Solution genutzt, muss bezüglich des Dateimanagements unterschieden werden zwischen TcCOM API [▶ 81] und der PLC API [▶ 83].

TcCOM API

Nach Einbindung einer Beschreibungsdatei im TcCOM TcMachineLearningModelCycal wird die entsprechende Beschreibungsdatei in das Visual Studio Projektverzeichnis kopiert und ist somit Bestandteil des Projekts: <VS Projekt>_MLInstall.

Bei Aktivieren der Konfiguration wird die Datei aus dem Visual Studio Projektverzeichnis in den Boot-Ordner auf dem Zielsystem kopiert: \TwinCAT\3.1\Boot\ML_Boot.

PLC API

Bei Nutzung der PLC API wird im SPS-Code der Dateiname und Pfad der Machine Learning Modelldatei als `T_MaxString` angegeben – entsprechend ist vom Anwender zu gewährleisten, dass vom Zielsystem ausgehend eine entsprechende Datei existiert. Das heißt, die Beschreibungsdatei wird weder Bestandteil des Visual Studio Projektverzeichnis noch wird sie automatisch auf das Zielsystem übertragen.

Übertragung der ML-Beschreibungsdateien auf das Zielsystem bei Aktivierung der Konfiguration

TcCOM API

Bei Nutzung des TcCOM Object **TcMachineLeraningModelCycal** wird die ML-Beschreibungsdatei vom XAE-System automatisch auf das XAR-System übertragen.

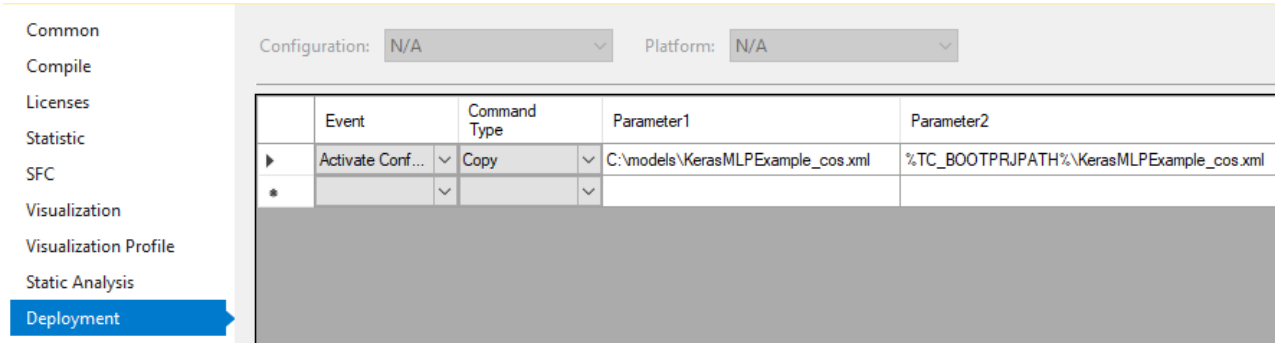
Die Datei wird aus dem Visual Studio Projektordner <VS Projekt>_MLInstall in den Boot-Ordner auf dem XAR übertragen TwinCAT\3.1\Boot\ML_Boot.

PLC API

Wird die **PLC API** genutzt, ist der Nutzer für die Übertragung der ML-Beschreibungsdatei zuständig. Dadurch erhöht sich einerseits die Flexibilität der Anwendung, andererseits sind entsprechende Schritte vom Anwender zu implementieren.

Die Übertragung der ML-Beschreibungsdatei auf das Zielsystem kann auf vielen Wegen erfolgen, nachfolgend wird einer beispielhaft genannt.

- Über die Eigenschaften des PLC-Projekts unter „Deployment“ kann angegeben werden, welche Dateien bei einem bestimmten Event, zum Beispiel Aktivierung der Konfiguration, auf das Zielsystem übertragen werden sollen.



HINWEIS

Schreibrechte auf dem Zielsystem

Es sind die Schreibrechte seitens des Betriebssystems und die Write Filter-Einstellungen zu beachten.

ML-Beschreibungsdateien im Feld aktualisieren

Ein wichtiges Szenario beim maschinellen Lernen ist die Aktualisierung von datenbasierten Algorithmen im Feld während der Laufzeit einer Maschine. Auch hier ist wieder zu unterscheiden zwischen Nutzung der TcCOM [▶ 81] API und der PLC API [▶ 83].

TcCOM API

Bei der TcCOM API verhält sich das Updateverhalten wie bei anderen Änderungen im TcCOM-Bereich. Es ist ein XAE-System notwendig mit einer ADS Route auf das Zielsystem. Im XAE kann im TwinCAT-Projekt eine neue ML-Beschreibungsdatei eingebunden werden. Durch Aktivieren der Konfiguration wird dann das neue Projekt auf das Zielsystem übertragen. Hier ist also ein Neustart der TwinCAT-Laufzeit notwendig.

PLC API

Wird die PLC API genutzt, ist ein Aktualisieren der ML-Beschreibungsdatei auf dem Zielsystem ohne Neustart der TwinCAT-Laufzeit möglich. Dazu ist lediglich ein Update der ML-Beschreibungsdatei auf dem Zielsystem sowie ein erneutes Triggern der Configure-Methode notwendig.

Sie können dazu, z. B. per ADS, die SPS-Variable, die den FullPath der ML-Beschreibungsdatei hält, auf den Wert der neuen, zuvor auf das Dateisystem übertragenen, Datei setzen, und dann den State der State-Maschine wieder auf „laden“ setzen.

6 API

Zur Programmierung in TwinCAT 3 stehen dem Anwender zwei Wege zur Verfügung. Es können statische TcCOM-Objekte erstellt und über einen zyklischen Task getriggert oder auch aus der PLC heraus eine Instanz erstellt und über die PLC konfiguriert werden.

Die Programmier-Schnittstelle der PLC bietet dabei deutlich mehr Flexibilität als die Nutzung einer TcCOM Instanz, hingegen ist letztere sehr einfach und in viele Fällen ausreichend.

6.1 TcCOM

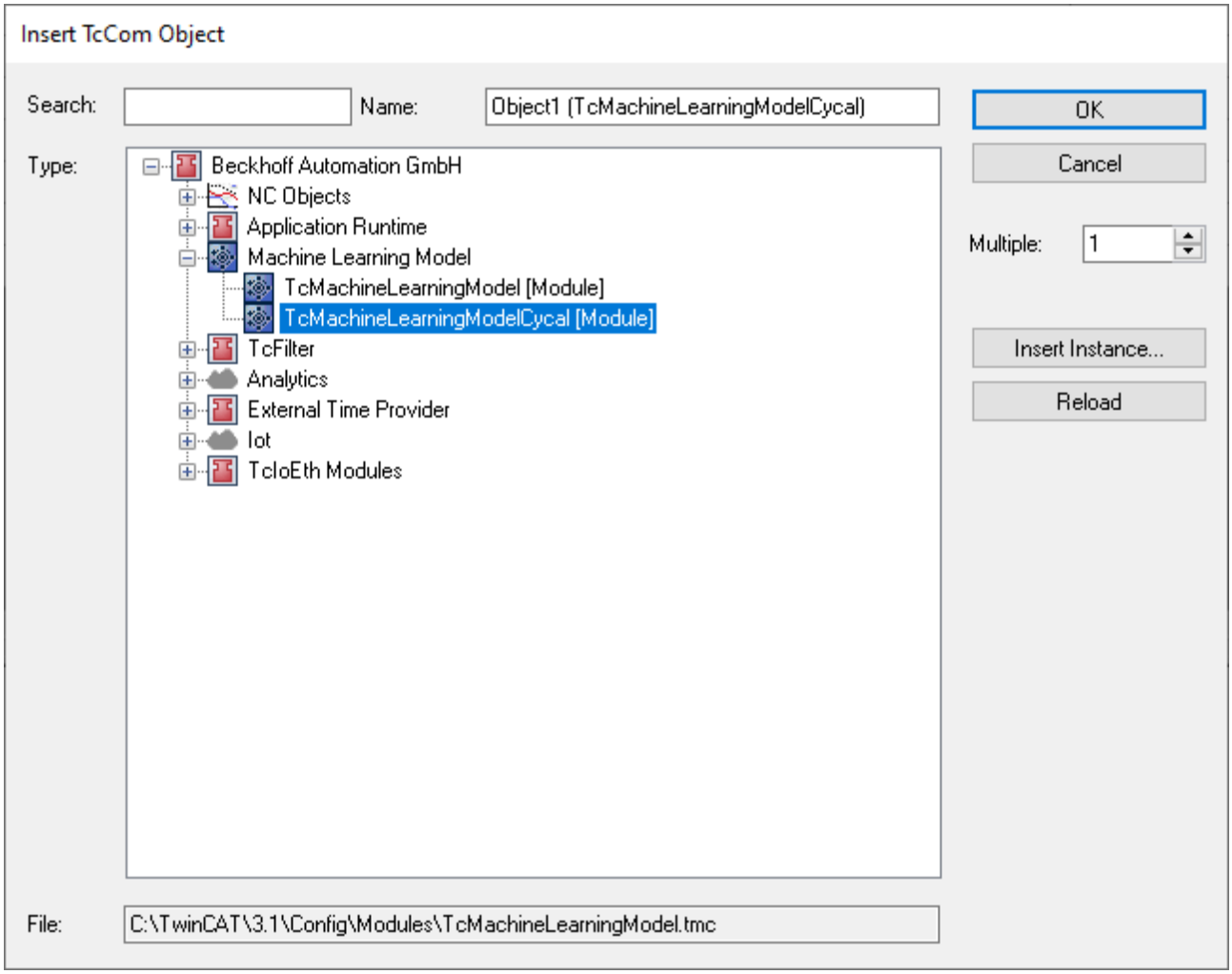
Die Nutzung eines TcCOM zur Inferenz eines ML-Modells in TwinCAT liefert eine sehr einfache Möglichkeit, trainierte Modelle in der TwinCAT-XAR auszuführen. Prinzipiell ist das gesamte Vorgehen bereits im Schnellstart dokumentiert, sodass im Folgenden zunächst die dort beschriebenen Schritte wiederholt und nachfolgend ein paar weitere Details gegeben werden.

Einbeziehung eines Modelles mittels TcCOM-Objekt

In diesem Abschnitt wird die Ausführung von Modellen des Maschinellen Lernens mittels eines vorbereiteten TcCOM-Objektes behandelt. Diese Schnittstelle bietet eine einfache und übersichtliche Möglichkeit, Modelle zu laden, in Echtzeit auszuführen und mittels des Prozessabbilds entsprechende Verknüpfungen in die eigene Applikation zu erzeugen.

Vorbereitetes TcCOM-Objekt TcMachineLearningModelCycal erzeugen

1. Selektieren Sie dazu den Knoten **TcCOM Objects** mit der rechten Maustaste und wählen Sie **Add New Item…**

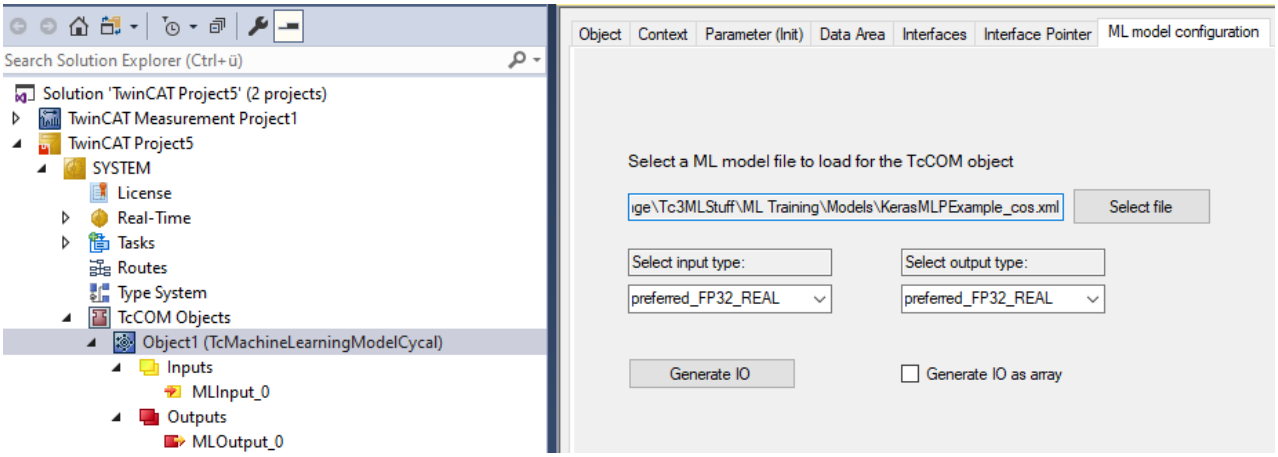


Unter Tasks eine neue TwinCAT Task erzeugen und der neu erzeugten Instanz des TcMachineLearningModelCycal diesen Task-Kontext zuweisen

2. Gehen Sie dazu auf den Reiter **Context** des erzeugten Objekts.
3. Wählen Sie im Drop-Down-Menü Ihre erzeugte Task aus.

⇨ Die Instanz des TcMachineLearningModelCycal verfügt über einen Reiter **ML model configuration**. Hier können Sie die Beschreibungsdatei des ML Algorithmus (XML oder BML) laden und bekommen im Anschluss daran verfügbare Datentypen für Ein- und Ausgänge des selektierten Modells angezeigt.

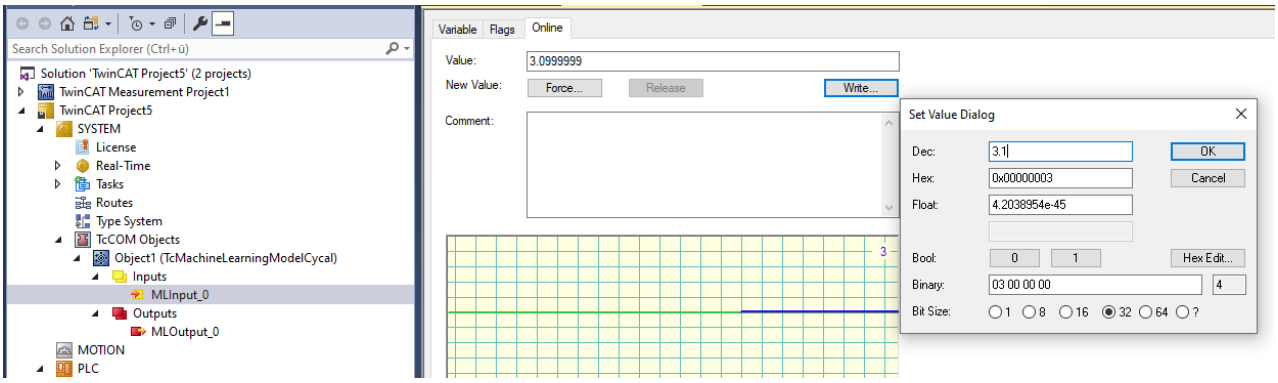
- Die Datei muss sich nicht auf dem Zielsystem befinden. Es kann vom Entwicklungssystem aus selektiert werden und wird dann beim Aktivieren der Konfiguration auf das Zielsystem überspielt.
  - Es wird zwischen präferierten und unterstützen Datentypen unterschieden. Der Unterschied ist lediglich, dass zur Laufzeit eine Konvertierung des Datentyps erfolgt, sollte ein nicht präferierter ausgewählt werden.
    Dies kann bei Verwendung nicht-präferierter Datentypen zu minimalen Performance-Einbußen führen.
- Automatisch werden die Datentypen für Ein- und Ausgänge zunächst auf den präferierten Datentypen gesetzt. Durch einen Klick auf **Generate IO** wird das Prozessabbild des selektierten Modells erstellt. Entsprechend erhalten Sie durch das Laden der *KerasMLPExample_cos.xml* ein Prozessabbild mit einem Input vom Typ REAL und ein Output vom Typ REAL.



Projekt auf Target aktivieren

1. Bevor Sie das Projekt auf einem Target aktivieren, müssen Sie im Projektbaum unter System>License im Tab **Manage Licenses** die TF3810 Lizenz manuell anwählen, da Sie ein Mehrlagiges Perzeptron laden möchten.
2. Aktivieren Sie die Konfiguration.

⇨ Sie können nun das Modell durch händisches Schreiben auf dem Input testen.



Ist das Prozessabbild größer, d.h. existieren viele Ein- oder Ausgänge, kann es hilfreich sein, nicht jeden Eingang einzeln als PDO zu erzeugen, sondern einen Eingang oder einen Ausgang als Array-Typ zu definieren. Aktvieren Sie dazu die Checkbox **Generate IO as array** und klicken Sie auf **Generate IO**.

Modelle mit mehreren Engines, vgl. XML Tag Parameters [▶ 77], können geladen werden, jedoch wird nur EngineId = 0 verwendet. Ein Umschalten zwischen den EngineIds ist mit der TcCOM API nicht vorgesehen.

Die verwendete ML-Beschreibungsdatei wird automatisch vom Engineering System auf das Runtime System bei Aktivieren der Konfiguration übertragen. Details zum Dateimanagement sind im Abschnitt Dateimanagement der ML-Beschreibungsdateien [▶ 78] beschrieben.

6.2 PLC API

6.2.1 Datatypes

6.2.1.1 ETcMlIDataType

Syntax

Definition:

```
TYPE ETcMllDataType :
(
    E_MLLDT_UNDEFINED  := 0,
    E_MLLDT_INT8_SINT  := 10,
    E_MLLDT_INT16_INT  := 20,
    E_MLLDT_INT32_DINT := 30,
    E_MLLDT_INT64_LINT := 40,
    E_MLLDT_FP16       := 50,
    E_MLLDT_FP16B      := 55,
    E_MLLDT_FP32_REAL  := 60,
    E_MLLDT_FP64_LREAL := 70,
    E_MLLDT_SPECIAL    := 99
)BYTE;
END_TYPE
```

Values

Name	Description
E_MLLDT_UNDEFINED	invalid / undefined data type
E_MLLDT_INT8_SINT	8-bit signed integer number (SINT / char)
E_MLLDT_INT16_INT	16-bit signed integer number (INT / short)
E_MLLDT_INT32_DINT	32-bit signed integer number (DINT / long)
E_MLLDT_INT64_LINT	64-bit signed integer number (LINT / long long)
E_MLLDT_FP16	16-bit IEEE floating point number (future usage)
E_MLLDT_FP16B	16-bit "bfloat16" floating point number (future usage)
E_MLLDT_FP32_REAL	32-bit IEEE floating point number (REAL / float)
E_MLLDT_FP64_LREAL	64-bit IEEE floating point number (LREAL / double)
E_MLLDT_SPECIAL	Function-specific byte stream

6.2.1.2 ST_MlIPredictionParameters

Syntax

Definition:

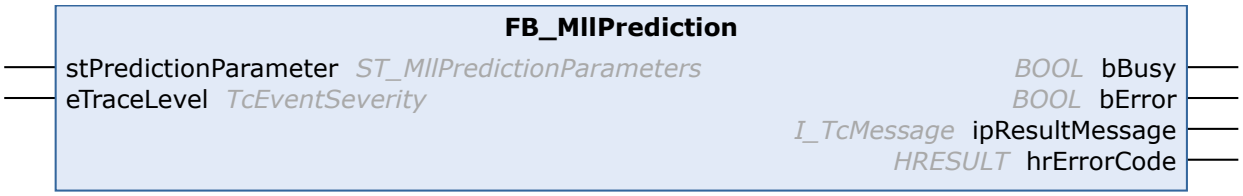
```
TYPE ST_MllPredictionParameters :
STRUCT
    MlModelFilepath : STRING(255);
    MaxConcurrency  : UINT;
END_STRUCT
END_TYPE
```

Parameters

Name	Type	Default	Description
MlModelFilepath	STRING(255)		File path of the loaded model. Either *.xml or *.bml file
MaxConcurrency	UINT	1	Maxium number of threads calling predict on the FB in the same time.

6.2.2 Function Blocks

6.2.2.1 FB_MlPrediction



Syntax

Definition:

```
FUNCTION_BLOCK FB_MlPrediction
VAR_INPUT
    stPredictionParameter : ST_MlPredictionParameters;
    eTraceLevel           : TcEventSeverity;
END_VAR
VAR_OUTPUT
    bBusy                 : BOOL;
    bError                : BOOL;
    ipResultMessage       : I_TcMessage;
    hrErrorCode           : HRESULT;
END_VAR
```

Inputs

Name	Type	Description
stPredictionParameter	ST_MlPredictionParameters [▶ 83]	General Prediction parameters
eTraceLevel	TcEventSeverity	Eventlogger trace level. Default = Critical. Must be set before Configure method is called

Outputs

Name	Type	Description
bBusy	BOOL	True if a asynchronous action is taking place
bError	BOOL	Indicates error in method
ipResultMessage	I_TcMessage	Contains the last invoked error
hrErrorCode	HRESULT	Unique error code

Methods

Name	Description
CheckPreferredIODataTypes [▶ 86]	
CheckSupportedIODataTypes [▶ 86]	
Configure [▶ 87]	
GetCustomAttribute_array [▶ 87]	
GetCustomAttribute_fp64 [▶ 88]	
GetCustomAttribute_int64 [▶ 89]	
GetCustomAttribute_str [▶ 89]	
GetEngineIdFromRef [▶ 90]	
GetInputDim [▶ 90]	
GetMaxConcurrency [▶ 91]	
GetModelName [▶ 91]	
GetOutputDim [▶ 92]	
Predict [▶ 92]	
PredictRef [▶ 93]	
Reset [▶ 94]	
SetActiveEngineOptions [▶ 95]	

General information

The `FB_MllPrediction` is a central Function Block for the usage of TC3 Machine Learning in the PLC. The Function Block offers a variety of Methods as described above. Basically, the `FB_MllPrediction` offers the functionality to load and to execute ML models. Hence, it is an interface to the TwinCAT 3 integrated inference engine (ML Runtime).

Error handling

Note that all methods of `FB_MllPrediction` return a `BOOL` which indicates if the execution on the method caused any error, e.g.

```
bFailed := fbprediction.GetInputDim(nInputDim);
```

The evaluation of the return value of the methods is equivalent to the evaluation of the Output `bError` of `FB_MllPrediction`.

Further, `FB_MllPrediction` references the TwinCAT 3 EventLogger and thus ensures that information (events) is provided via the standardized interface I_TcMessage. The trace level can be adjusted using TcEventSeverity.

Sample Code

Sample code for the usage of the Function Block is available here [▶ 96].

System Requirements

6.2.2.1.1 CheckPreferredIODataTypes

Syntax

Definition:

```
METHOD CheckPreferredIODataTypes : BOOL
VAR_INPUT
    fmtInputType  : ETcMllDataType;
    fmtOutputType : ETcMllDataType;
    IsPreferred   : Reference To BOOL;
END_VAR
```

Inputs

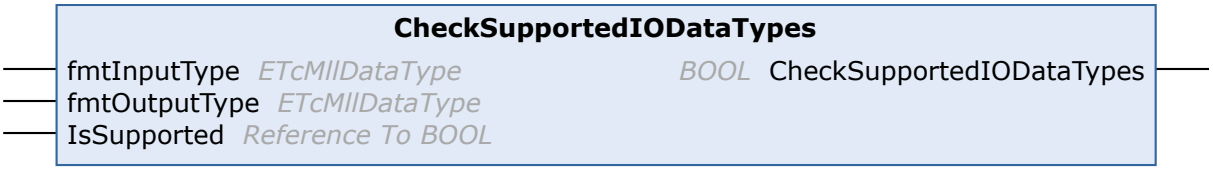
Name	Type	Description
fmtInputType	ETcMllDataType [▶ 83]	Input type to check
fmtOutputType	ETcMllDataType [▶ 83]	Output type to check
IsPreferred	Reference To BOOL	Return true if preferred typo

Return value

BOOL

A distinction is made between preferred and supported data types. The only difference is that a conversion of the data type takes place at runtime if a non-preferred type is selected. This may lead to slight losses in performance when using non-preferred data types.

6.2.2.1.2 CheckSupportedIODataTypes



Syntax

Definition:

```
METHOD CheckSupportedIODataTypes : BOOL
VAR_INPUT
    fmtInputType  : ETcMllDataType;
    fmtOutputType : ETcMllDataType;
    IsSupported   : Reference To BOOL;
END_VAR
```

Inputs

Name	Type	Description
fmtInputType	ETcMllDataType [▶ 83]	Input type to check
fmtOutputType	ETcMllDataType [▶ 83]	Output type to check
IsSupported	Reference To BOOL	returns true if supported

Return value

BOOL

6.2.2.1.3 Configure

Syntax

Definition:

```
METHOD Configure : BOOL
```

Return value

BOOL

The method loads the specified ML model description file and configures the inference engine. Specify all settings using the `stPredictionParameter` before calling the configure method.

```
fbPredict.stPredictionParameter.MlModelFilepath := 'C:/myModel.xml';
fbPredict.stPredictionParameter.MaxConcurrency := 1;
bConfigured := fbPredict.Configure();
```

6.2.2.1.4 GetCustomAttribute_array

Syntax

Definition:

```
METHOD GetCustomAttribute_array : BOOL
VAR_INPUT
    sCustomAttributeName : T_MaxString;
    fmtAttributeDataType : Reference To ETcMllDataType;
    pDataBuffer          : PVOID;
    nDataBufferLen       : UDINT;
    nArrayLength         : Reference To UDINT;
    pnBytesWritten       : Pointer To UDINT;
END_VAR
```

Inputs

Name	Type	Description
sCustomAttributeName	T_MaxString	Name of the custom attribute
fmtAttributeDataType	Reference To ETcMllDataType [▶ 83]	Data format of the custom attribute
pDataBuffer	PVOID	Destination data buffer, where the custom attribute is copied into
nDataBufferLen	UDINT	Length of the destination data buffer in bytes
nArrayLength	Reference To UDINT	Number of data type elements (i.e. number of fp32 values) found in the custom attribute
pnBytesWritten	Pointer To UDINT	Returns the number of bytes that have been written to the destination buffer

Return value

BOOL

Methods reads a custom attribute specified by `sCustomAttributeName` of type Array. Refer to this sample code [▶ 96] showing how to read an array of LREAL.

6.2.2.1.5 GetCustomAttribute_fp64

Syntax

Definition:

```
METHOD GetCustomAttribute_fp64 : BOOL
VAR_INPUT
    sCustomAttributeName : T_MaxString;
    lrDataOut            : Reference To LREAL;
END_VAR
```

Inputs

Name	Type	Description
sCustomAttributeName	T_MaxString	Name of the custom fp64 attribute
lrDataOut	Reference To LREAL	Output value of the custom fp64 attribute

Return value

BOOL

Methods reads a custom attribute specified by `sCustomAttributeName` of type LREAL.

```
sCustomKey : T_MaxString := XmlTreeItem/XmlAttribute';
fValue : LREAL;
```

```
fbprediction.GetCustomAttribute_fp64(sCustomKey, fValue);
```

6.2.2.1.6 GetCustomAttribute_int64

Syntax

Definition:

```
METHOD GetCustomAttribute_int64 : BOOL
VAR_INPUT
    sCustomAttributeName : T_MaxString;
    nDataOut             : Reference To LINT;
END_VAR
```

Inputs

Name	Type	Description
sCustomAttributeName	T_MaxString	Name of the custom int64 attribute
nDataOut	Reference To LINT	Output value of the custom int64 attribute

Return value

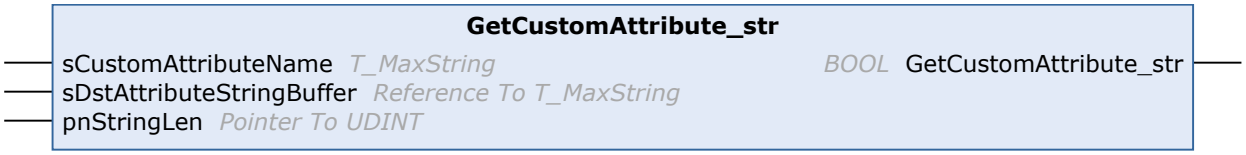
BOOL

Methods reads a custom attribute specified by `sCustomAttributeName` of type LINT.

```
sCustomKey : T_MaxString := XmlTreeItem/XmlAttribute';
iValue : LINT;
```

```
fbprediction.GetCustomAttribute_int64(sCustomKey, iValue);
```

6.2.2.1.7 GetCustomAttribute_str



Syntax

Definition:

```
METHOD GetCustomAttribute_str : BOOL
VAR_INPUT
    sCustomAttributeName      : T_MaxString;
    sDstAttributeStringBuffer : Reference To T_MaxString;
    pnStringLen               : Pointer To UDINT;
END_VAR
```

Inputs

Name	Type	Description
sCustomAttributeName	T_MaxString	Name of the custom string attribute
sDstAttributeStringBuffer	Reference To T_MaxString	Pointer to a string buffer to write the custom string attribute into
pnStringLen	Pointer To UDINT	(Optional) Actual length of the string attribute

Return value

BOOL

Methods reads a custom attribute specified by `sCustomAttributeName` of type T_MaxString.

```
sCustomKey : T_MaxString := XmlTreeItem/XmlAttribute';
sValue : T_MaxString ;
```

```
fbprediction.GetCustomAttribute_str(sCustomKey, sValue);
```

6.2.2.1.8 GetEngineIdFromRef

Syntax

Definition:

```
METHOD GetEngineIdFromRef : BOOL
VAR_INPUT
    sEngineRef : T_MaxString;
    nEngineId  : Reference To UDINT;
END_VAR
```

Inputs

Name	Type	Description
sEngineRef	T_MaxString	Reference string of model engine (or parameter set) used for prediction, use default value 0 if there are no multi-engines used
nEngineId	Reference To UDINT	Id of model engine (or parameter set)

Return value

BOOL

In case of multi engines in an ML model description file, engines can be selected via ID or reference name. While the Predict-method expects an ID as input, the PredictRef method expects a reference name as input.

`GetEngineIdFromRef` can convert a reference name into an engine ID. Reference names can be found in the Beckhoff ML XML file inside the <IODistributor> section, see XML attribute str_reference, and can be set via the TC3 Machine Learning Model Manager, see Merge Tool.

6.2.2.1.9 GetInputDim

Syntax

Definition:

```
METHOD GetInputDim : BOOL
VAR_INPUT
    nInputDim : Reference To UDINT;
END_VAR
```

Inputs

Name	Type	Description
nInputDim	Reference To UDINT	Size of the input data array

Return value

BOOL

6.2.2.1.10 GetMaxConcurrency

Syntax

Definition:

```
METHOD GetMaxConcurrency : BOOL
VAR_INPUT
    nConcurrency : Reference To UDINT;
END_VAR
```

Inputs

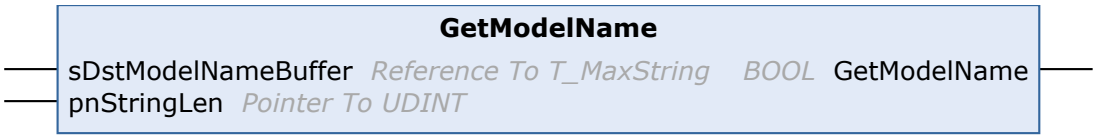
Name	Type	Description
nConcurrency	Reference To UDINT	Maximum supported number of concurrently processing threads

Return value

BOOL

Methods reads out current configuration of inference engine regarding possible number of concurrently processing threads. This value is set by the user using `stPredictionParameter` before calling Configure method.

6.2.2.1.11 GetModelName



Syntax

Definition:

```
METHOD GetModelName : BOOL
VAR_INPUT
    sDstModelNameBuffer : Reference To T_MaxString;
    pnStringLen         : Pointer To UDINT;
END_VAR
```

Inputs

Name	Type	Description
sDstModelNameBuffer	Reference To T_MaxString	Name of the loaded model
pnStringLen	Pointer To UDINT	(Optional) Actual length of the string attribute

Return value

BOOL

The method reads the model name of the loaded ML description file. The model name is an identifier of the machine learning model type. Returned strings can be for example ‘support_vector_machine’ or ‘mlp_neural_network’.

6.2.2.1.12 GetOutputDim

Syntax

Definition:

```
METHOD GetOutputDim : BOOL
VAR_INPUT
    nOutputDim : Reference To UDINT;
END_VAR
```

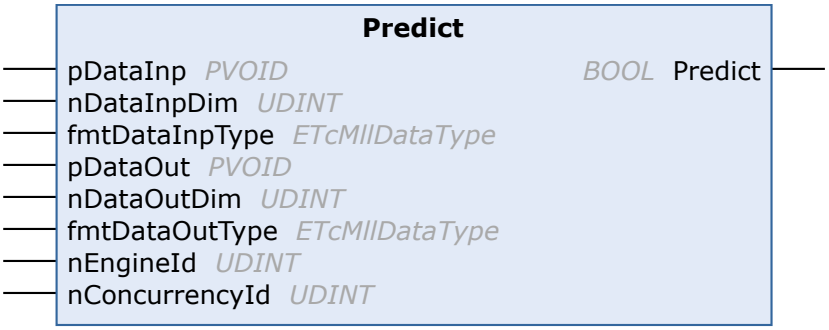
Inputs

Name	Type	Description
nOutputDim	Reference To UDINT	Size of the output data array

Return value

BOOL

6.2.2.1.13 Predict



Syntax

Definition:

```
METHOD Predict : BOOL
VAR_INPUT
    pDataInp       : PVOID;
    nDataInpDim    : UDINT;
    fmtDataInpType : ETcMllDataType;
    pDataOut       : PVOID;
    nDataOutDim    : UDINT;
    fmtDataOutType : ETcMllDataType;
    nEngineId      : UDINT;
    nConcurrencyId : UDINT;
END_VAR
```

Inputs

Name	Type	Description
pDataInp	PVOID	Pointer to the input data array (e.g. ARRAY[0..10] OF REAL in PLC)
nDataInpDim	UDINT	Number of inputs in the current vector
fmtDataInpType	ETcMlIDataType [▶ 83]	ETcMlIDataType data type of input array
pDataOut	PVOID	Pointer to the output data array (e.g. ARRAY[0..10] OF REAL in PLC)
nDataOutDim	UDINT	Number of outputs in the current vector
fmtDataOutType	ETcMlIDataType [▶ 83]	ETcMlIDataType data type of output array
nEngineId	UDINT	Id of model engine (or parameter set) used for prediction, use default value 0 if there are no multi-engines used
nConcurrencyId	UDINT	Id of the processing thread. Important: Never have two concurrently processing threads use the same id.

Return value

BOOL

The method performs the inference of the loaded model with the given input data and stores the result in the output data. Use configure method to load a ML model description file before calling Predict method.

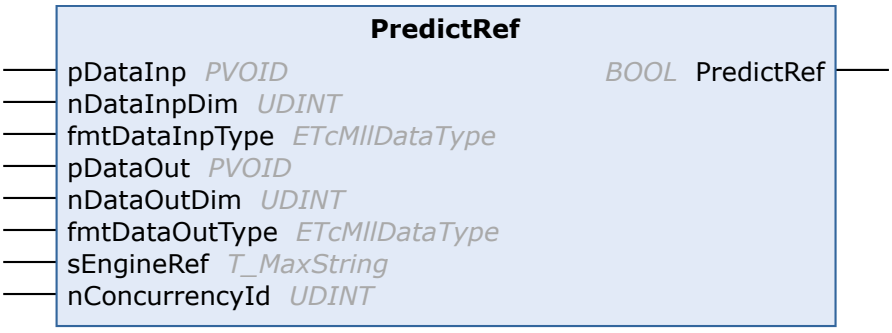
For the inputs and outputs a pointer, the number of inputs/outputs and the data type are needed.

Sample call:

```
dtype      : ETcMlIDataType.E_MLLDT_FP32_REAL;
nInputDim  : UDINT := 3;
nOutputDim : UDINT := 2;
nInput     : ARRAY[1..3] OF REAL;
nOutput    : ARRAY[1..2] OF REAL;
nCurrentEngineID : UDINT := 0;
nConcurrencyId : UDINT := 0;
```

```
fbprediction.Predict(
        pDataInp:=ADR(nInput) ,
        nDataInpDim:= nInputDim,
        fmtDataInpType:= dtype,
        pDataOut:=ADR(nOutput) ,
        nDataOutDim:= nOutputDim,
        fmtDataOutType:= dtype,
        nEngineId:= nCurrentEngineID,
        nConcurrencyId:= nConcurrencyId );
```

6.2.2.1.14 PredictRef



Syntax

Definition:

```
METHOD PredictRef : BOOL
VAR_INPUT
    pDataInp       : PVOID;
    nDataInpDim    : UDINT;
    fmtDataInpType : ETcMllDataType;
    pDataOut       : PVOID;
    nDataOutDim    : UDINT;
    fmtDataOutType : ETcMllDataType;
    sEngineRef     : T_MaxString;
    nConcurrencyId : UDINT;
END_VAR
```

Inputs

Name	Type	Description
pDataInp	PVOID	Pointer to the input data array (e.g. ARRAY[0..10] OF REAL in PLC)
nDataInpDim	UDINT	Number of inputs in the current vector
fmtDataInpType	ETcMllDataType [▶ 83]	ETcMllDataType data type of input array
pDataOut	PVOID	Pointer to the output data array (e.g. ARRAY[0..10] OF REAL in PLC)
nDataOutDim	UDINT	Number of outputs in the current vector
fmtDataOutType	ETcMllDataType [▶ 83]	ETcMllDataType data type of output array
sEngineRef	T_MaxString	Reference string of model engine (or parameter set) used for prediction, use default value 0 if there are no multi-engines used
nConcurrencyId	UDINT	Id of the processing thread. Important: Never have two concurrently processing threads use the same id.

Return value

BOOL

The method performs the inference of the loaded model with the given input data and stores the result in the output data. Use configure method to load a ML model description file before calling PredictRef method.

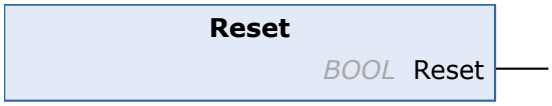
For the inputs and outputs a pointer, the number of inputs/outputs and the data type are needed.

Sample call:

```
dtype      : ETcMllDataType.E_MLLDT_FP32_REAL;
nInputDim  : UDINT := 3;
nOutputDim : UDINT := 2;
nInput     : ARRAY[1..3] OF REAL;
nOutput    : ARRAY[1..2] OF REAL;
sCurrentEngineRef: T_MaxString := 'EngineRef';
nConcurrencyId : UDINT := 0;
```

```
fbprediction.Predict(
        pDataInp:=ADR(nInput) ,
        nDataInpDim:= nInputDim,
        fmtDataInpType:= dtype,
        pDataOut:=ADR(nOutput) ,
        nDataOutDim:= nOutputDim,
        fmtDataOutType:= dtype,
        sEngineRef:= sCurrentEngineRef,
        nConcurrencyId:= nConcurrencyId );
```

6.2.2.1.15 Reset



Syntax

Definition:

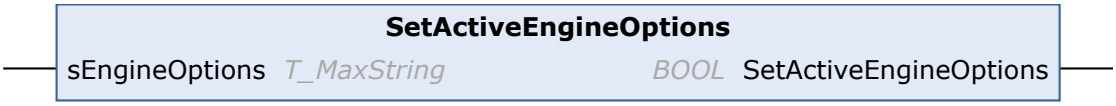
```
METHOD Reset : BOOL
```

Return value

BOOL

This Methods resets the FB's error state.

6.2.2.1.16 SetActiveEngineOptions



Syntax

Definition:

```
METHOD SetActiveEngineOptions : BOOL
VAR_INPUT
    sEngineOptions : T_MaxString;
END_VAR
```

Inputs

Name	Type	Description
sEngineOptions	T_MaxString	

Return value

BOOL

Input is a JSON-String with the following Key-Value-Pairs:

Key	Value	Defaut-Value*
allow_FPU	TRUE / FALSE	TRUE
Allow_SSE3	TRUE / FALSE	TRUE
Allow_AVX	TRUE / FALSE	TRUE
Allow_FMA	TRUE / FALSE	TRUE
Allow_AVX_512F	TRUE / FALSE	TRUE

*Default-Values: The library uses as default the maximum performance. Hence, all available SIMD-extensions provided by the target PC's CPU are set to TRUE by default.

Sample Code to disable FMA and allow AVX (all others will be left unaltered):

```
fbPredict : FB_MllPrediction;
EngineOpts : T_MaxString := '{ "allow_AVX":"true", "allow_FMA":"false" }';
```

```
fbPredict.SetActiveEngineOptions(EngineOpts);
```

7 Beispiele

7.1 PLC API

7.1.1 Schnellstart

Das Beispiel aus dem Abschnitt Schnellstart [▶ 17] kann hier heruntergeladen werden: https://infosys.beckhoff.com/content/1031/tf38x0_tc3_ML_NN_Inference_Engine/Resources/8746884875.zip.

Das ZIP enthält zum einen ein tszip-Archiv (siehe PLC-Dokumentation, tszip) und zum anderen eine Beckhoff ML XML Datei (KerasMLPExample_cos.XML). Kopieren Sie die XML Datei an die in der PLC als Zielort definierten Stelle, oder verändern Sie den String auf einen anderen Pfad.

7.1.2 Ausführliches Beispiel

Das Beispiel kann hier heruntergeladen werden: https://infosys.beckhoff.com/content/1031/tf38x0_tc3_ML_NN_Inference_Engine/Resources/8763219467.zip.

Das ZIP enthält zum einen ein tszip-Archiv (siehe PLC-Dokumentation, tszip) und zum anderen eine Beckhoff ML XML Datei (TrigonometryMLP.XML). Kopieren Sie die XML Datei an die in der PLC als Zielort definierten Stelle, oder verändern Sie den String auf einen anderen Pfad.

Die ML-Modell Beschreibungsdatei beinhaltet ein MLP mit einem Input und einem Output, vgl. XML-Tag <Configuration> mit `int64_numInputNeurons = 1` sowie der zweiten (letzten) Schicht mit `int64_numNeurons = 1`. Es existieren zwei Parameter-Tags, d.h. es sind zwei unterschiedlich trainierte aber von der Struktur (<Configuration>) identische MLPs in der Datei enthalten. Es handelt sich dabei einmal um ein MLP, welches trainiert wurde eine Sinusfunktion zu approximieren und einmal um ein MLP, welches eine Kosinusfunktion approximieren soll. Im Bereich <IODistributor> ist zu sehen, dass eine Engine mit der Referenz „sin_engine“ und die andere mit „cos_engine“ erreichbar ist. Im Bereich <CustomAttributes> sind einige Metadaten hinterlegt, z. B. der Name des Modells, die Version und der Gültigkeitsbereich der Input-Variablen.

Im SPS Quellcode wird, wie beim Schnellstartbeispiel eine einfacher Zustandsautomat durchlaufen. Der Unterschied zum Schnellstartbeispiel liegt in der Ausführlichkeit des Zustands „Configure“ sowie der Nutzung von mehreren Engines.

Im Zustand „Configure“ wird exemplarisch gezeigt, wie flexibel mit der Anzahl von Ein- und Ausgängen umgegangen werden kann und wie Sie möglichst viele Informationen aus der Beschreibungsdatei auslesen und direkt in der SPS nutzbar machen können.

Das Schalten zwischen den beiden Engines können Sie im Online View händisch herbeiführen, indem Sie die EngineId auf 0 oder 1 stellen.

7.1.3 Paralleler, nicht-blockierender, Zugriff auf ein Inferenzmodul

In diesem Beispiel wird gezeigt, wie auf eine Instanz des `FB_MllPrediction` aus zwei parallellaufenden Tasks zugegriffen werden kann.

Das Beispiel kann hier heruntergeladen werden: https://infosys.beckhoff.com/content/1031/tf38x0_tc3_ML_NN_Inference_Engine/Resources/8775872011.zip.

Die Instanz `fbpredict` ist in der GVL_ML deklariert. Somit haben alle Programme in der PLC Zugriff auf die Instanz. Als Programme sind angelegt.

- P_InitML: Hier ist die Schrittkette zur Initialisierung/zum Laden eines ML-Modells beschrieben.
- P_Predict_Task1: Hier wird die Predict-Methode des `fbpredict` aufgerufen, wobei das PRG auf Core 1 ausgeführt wird.
- P_Predict_Task2: Hier wird die Predict-Methode des `fbpredict` aufgerufen, wobei das PRG auf Core 2 ausgeführt wird.

Die wesentlichen Bestandteile zur parallelen Ausführung von 2 Predict-Aufrufen sind:

- Die maximale Anzahl paralleler Zugriffe muss bei der Configure Methode angegeben werden: `GVL_ML.fbpredict.stPredictionParameter.MaxConcurrency := nMaxConcurrency;` mit nMaxConcurrency = 2.
  Die Instanz hält dann diese Anzahl von **unabhängigen** Inferenz-Maschinen vor.
- Beim Aufruf der Predict-Methode ist eine eindeutige ID des aufrufenden Kontextes anzugeben. Diese sind in P_Predict_Task1 und P_Predict_Task2 als Konstanten deklariert, siehe `nConcurrencyId`.
  Der Anwender muss sicherstellen, dass jeder aufrufende Kontext eine eindeutige ID mit dem Predict-Aufruf übergibt.
- Der restliche Quellcode ist größtenteils identisch mit dem Schnellstart-Beispiel [▶ 96].

8 Support und Service

Beckhoff und seine weltweiten Partnerfirmen bieten einen umfassenden Support und Service, der eine schnelle und kompetente Unterstützung bei allen Fragen zu Beckhoff Produkten und Systemlösungen zur Verfügung stellt.

Downloadfinder

Unser Downloadfinder beinhaltet alle Dateien, die wir Ihnen zum Herunterladen anbieten. Sie finden dort Applikationsberichte, technische Dokumentationen, technische Zeichnungen, Konfigurationsdateien und vieles mehr.

Die Downloads sind in verschiedenen Formaten erhältlich.

Beckhoff Niederlassungen und Vertretungen

Wenden Sie sich bitte an Ihre Beckhoff Niederlassung oder Ihre Vertretung für den lokalen Support und Service zu Beckhoff Produkten!

Die Adressen der weltweiten Beckhoff Niederlassungen und Vertretungen entnehmen Sie bitte unserer Internetseite: www.beckhoff.com

Dort finden Sie auch weitere Dokumentationen zu Beckhoff Komponenten.

Beckhoff Support

Der Support bietet Ihnen einen umfangreichen technischen Support, der Sie nicht nur bei dem Einsatz einzelner Beckhoff Produkte, sondern auch bei weiteren umfassenden Dienstleistungen unterstützt:

- Support
- Planung, Programmierung und Inbetriebnahme komplexer Automatisierungssysteme
- umfangreiches Schulungsprogramm für Beckhoff Systemkomponenten

Hotline:	+49 5246 963-157
E-Mail:	support@beckhoff.com

Beckhoff Service

Das Beckhoff Service-Center unterstützt Sie rund um den After-Sales-Service:

- Vor-Ort-Service
- Reparaturservice
- Ersatzteilservice
- Hotline-Service

Hotline:	+49 5246 963-460
E-Mail:	service@beckhoff.com

Beckhoff Unternehmenszentrale

Beckhoff Automation GmbH & Co. KG

Hülshorstweg 20
33415 Verl
Deutschland

Telefon:	+49 5246 963-0
E-Mail:	info@beckhoff.com
Internet:	www.beckhoff.com

9 Anhang

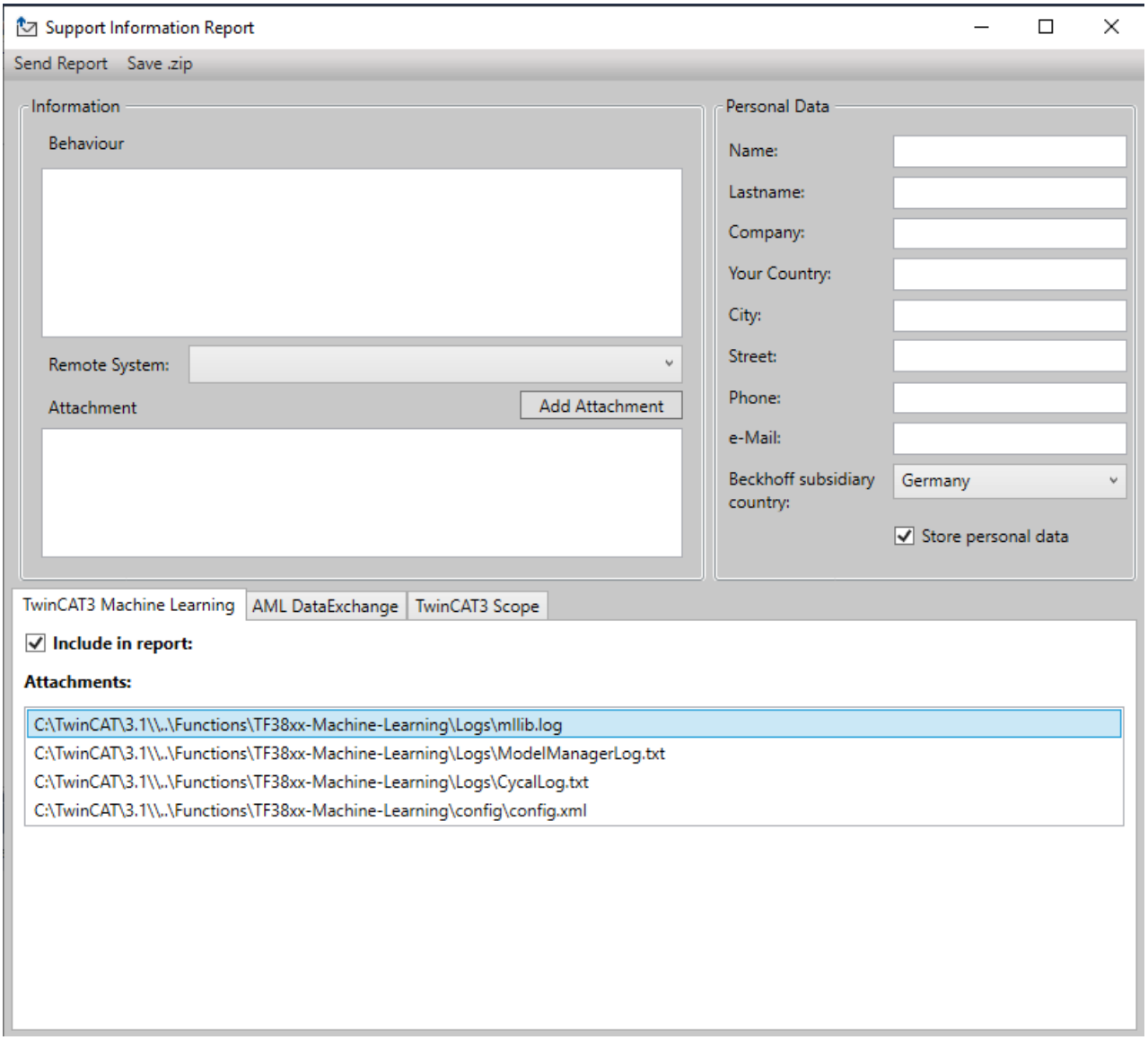
9.1 Log Dateien

Für den Support sind die Log Dateien wichtig. Diese finden Sie unter *<TwinCATInstallPath>\Functions\TF38xx-Machine-Learning\Logs*.

Auf dem XAE System werden die Dateien **CycalLog.txt** und **ModelManagerLog.txt** erstellt. Diese loggen das Verhalten beim Engineering von Komponenten. Das CycalLog.txt beinhaltet logs bzgl. der Konfiguration des TcMachineLearningModelCycal TcCOM. Der TC3 Machine Learning Model Manager schreibt in die Datei ModelManagerLog.txt.

Auf dem Runtime System wird die Datei **mllib.log** zum Loggen von Verhalten während der Laufzeit der Maschine erstellt.

Über die Menüleiste des Visual Studio gelangen Sie unter TwinCAT > Machine Learning zu Report an Issue. Dieser Dialog öffnet den **Support Information Report**, welcher Sie unterstützt einen Report an den Beckhoff-Support zu senden.



9.2 Third-party components

This software contains third-party components.

Please refer to the license file provided in the following folder for further information:
<TwinCatInstallPath>\Functions\TF38xx-Machine-Learning\Legal

9.3 XML-Exporter

Einordnung der XML-Exporter

Die von Beckhoff bereitgestellten XML-Exporter können frei genutzt und verändert werden. Sie sind quelloffen und unter MIT-Lizenz. Dies bietet Kunden die Möglichkeit, die XML-Exporter entsprechend ihren Bedürfnissen anzupassen, bspw. durch Einfügen unternehmens- oder projektspezifisicher CustomAttributes, vgl. Abschnitt XML Tag CustomAttributes [▶ 76].

Die XML-Exporter werden nach Installation des Produkts im Ordner *<TwinCATPath>\Functions\TF38xx-Machine-Learning\Utilities\exporter* bereitgestellt.

Die XML-Exporter basieren auf speziellen Versionen der Bibliotheken

Empfohlen wird das Exportieren von erstellen ML-Modellen über das ONNX-Format sowie entsprechender Konvertierung in XML oder BML. Die XML-Exporter waren in der frühen Phase von TwinCAT Machine Learning als Übergangslösung gedacht, bis alle einschlägigen Bibliotheken einen umfänglichen ONNX Support anbieten. Dies ist heute der Fall, sodass die XML-Exporter nicht mehr mit neueren Bibliotheken getestet und geupdatet werden.

Direkter XML-Export eines MLP

Es werden nur Netzarchitekturen unterstützt, die eine sequenzielle Struktur im folgenden Sinne aufweisen: Jedes Neuron einer Schicht ist ausschließlich mit jedem Neuron der darauffolgenden Schicht verknüpft. Es ist möglich, Layer ohne Bias zu exportieren.

Export aus Keras/Tensor Flow

- Datei: KerasMlp2Xml.py
- Beispielaufruf: https://infosys.beckhoff.com/content/1031/tf38x0_tc3_ML_NN_Inference_Engine/Resources/8746685963.zip
- Voraussetzungen an die Python-Umgebung:
  - Keras mit TensorFlow-Backend (Tensor Flow Version 1.15.0)
  - Numpy (Version 1.17.4)
  - Matplotlib
- Unterstützte Aktivierungsfunktionen: tanh, sigmoid, softmax, relu, linear/identity, exp, softplus, softsign
- Es können nur **sequential models** und keine **functional models** mit dem XML-Exporter exportiert werden. Das Modell ist entsprechend zu erzeugen mit:

```
from tensorflow.keras.models import Sequential
model = Sequential()
```

- Dropout Layer werden unterstützt (nur für Training relevant, werden beim Export ignoriert)
- Dense Layer werden unterstützt. Die Aktivierungsfunktionen müssen als Argument der Layer übergeben werden und nicht als eigenständige Aktivierungs-Layer.

```
from keras.layers import Activation, Dense
# this will not work !!!
model.add(Dense(64))
model.add(Activation('tanh'))
# this will work
model.add(Dense(64,activation='tanh'))
```

- Die API ist ausführlich im Kopf der Datei KerasMlp2Xml.py beschrieben.
  `net2xml(net,output_scaling_bias=None,output_scaling_scal=None)`
  - `net`, obligatorisch, Klasse des trainierten Modells
  - `output_scaling_bias`, optional, Liste (bei mehreren Features) sonst float oder int
  - `output_scaling_scal`, optional, Liste (bei mehreren Features) sonst float oder int
  - Rückgabe ist ein String-Dokument, welches als XML gespeichert werden kann.

Export aus MATLAB®

- Datei: MatlabMlp2Xml.m
- Beispielaufruf: https://infosys.beckhoff.com/content/1031/tf38x0_tc3_ML_NN_Inference_Engine/Resources/8746880267.zip
- Voraussetzungen an die MATLAB®-Umgebung:
  - MATLAB®
  - Deep Learning Toolbox
- Unterstützte Aktivierungsfunktionen: tanh, sigmoid, softmax, relu, linear/identity
- Unterstützte Modelle der Deep Learning Toolbox: fitnet, patternnet
- ProcessFcns werden unterstützt vom Typ mapminmax und mapstd
- Die Nutzung von ProcessFcns ist optional
- Wird eine ProcessFcn in der Ouput-Layer verwendet, muss dessen Aktivierungsfunktion purelin sein
- Die API ist ausführlich im Kopf der Datei MatlabMlp2Xml.m beschrieben
  `MatlabMlp2Xml(net, fnstr, varargin)`
- net, obligatorisch, Klasse des trainierten Modells
- fnstr, obligatorisch, String mit Pfad und Dateiname
- output_scaling_bias, optional, Vektor
- output_scaling_scal, optional, Vektor

Direkter XML-Export einer SVM

Export aus Scikit-learn

- Datei: SciKitLearnSvm2Xml.py
- Beispielaufruf: https://infosys.beckhoff.com/content/1031/tf38x0_tc3_ML_NN_Inference_Engine/Resources/8746882571.zip
- Voraussetzungen an die Python-Umgebung:
  - Python Interpreter: 3.6 oder höher
  - Scikit-learn: Version 0.22.0 oder höher
  - Matplotlib
  - Numpy
- Es können nur numerische Class-Labels exportiert werden.
- Unterstützte Klassen bzw. Modelle: SVC, NuSVC, OneClassSVM, SVR, NuSVR
  - LinearSVR und LinearSVC werden vom Exporter nicht unterstützt, können aber alternativ über die Klassen SVR und SVC mit jeweils linearem Kernel umgesetzt werden.
- Unterstützte Kernel-Funktionen: Linear, rbf, sigmoid, polynomial
  - Individuelle Kernel-Funktionen sowie precomputed Funktionen werden nicht unterstützt
- Anmerkungen zu Modellparametern:
  - *Gamma* = scale wird nicht unterstützt
  - *Gamma* = auto_deprecated: Es wird exportiert gamma = 0.0
  - *Gamma* = auto: Es wird gamma = 1/n_features exportiert.
  - *C* = inf wird nicht unterstützt
  - *decision_function_shape* = ovr wird nicht unterstützt. Es muss *decision_function_shape* = ovo verwendet werden. Default in Scikit-learn ist ovr!
  - *break_ties* wird ignoriert, da *decision_function_shape* = ovr nicht unterstützt wird.
- Die API ist ausführlich im Kopf der Datei SciKitLearnSvm2Xml.py beschrieben.
  `svm2xml(svm, input_scaling_bias=None, input_scaling_scal=None)`
  - `net`, obligatorisch, Klasse des trainierten Modells
  - `input_scaling_bias`, optional, Liste (bei mehreren Features) sonst float oder int

- `input_scaling_scal`, optional, Liste (bei mehreren Features) sonst float oder int
- Rückgabe ist ein String-Dokument, welches als XML gespeichert werden kann.

9.3.1 XML Exporter - Beispiele

Hier können kleine Beispiele für die Nutzung der von Beckhoff bereitgestellten XML Exporter [▶ 100] heruntergeladen werden.

- Export eines MLP aus Keras/TensorFlow: https://infosys.beckhoff.com/content/1031/tf38x0_tc3_ML_NN_Inference_Engine/Resources/8746685963.zip
- Export eines MLP aus MATLAB®: https://infosys.beckhoff.com/content/1031/tf38x0_tc3_ML_NN_Inference_Engine/Resources/8746880267.zip
- Export einer SVM aus Scikit-learn: https://infosys.beckhoff.com/content/1031/tf38x0_tc3_ML_NN_Inference_Engine/Resources/8746882571.zip

Mehr Informationen:
www.beckhoff.com/tf3800

Beckhoff Automation GmbH & Co. KG
Hülshorstweg 20
33415 Verl
Deutschland
Telefon: +49 5246 9630
info@beckhoff.com
www.beckhoff.com