

Tutorial zum Trainieren eines Klassifikators mit Mobile SSI

1 Ziel des Tutorial

Diese Tutorial soll schrittweise vermitteln, wie Sie schnell und einfach ein Pipeline für einen Klassifikator mit Mobile SSI erstellen.

2 Voraussetzungen

Es wird vorausgesetzt, dass bereits Testdaten aufgenommen, annotiert und mittels anno2sample.exe zu samples transformiert wurden. Ebenso muss im selben Ordner die Datei functionals.option vorhanden sein.

Das aktuelle Stand sollte in etwa wie folgt aussehen:

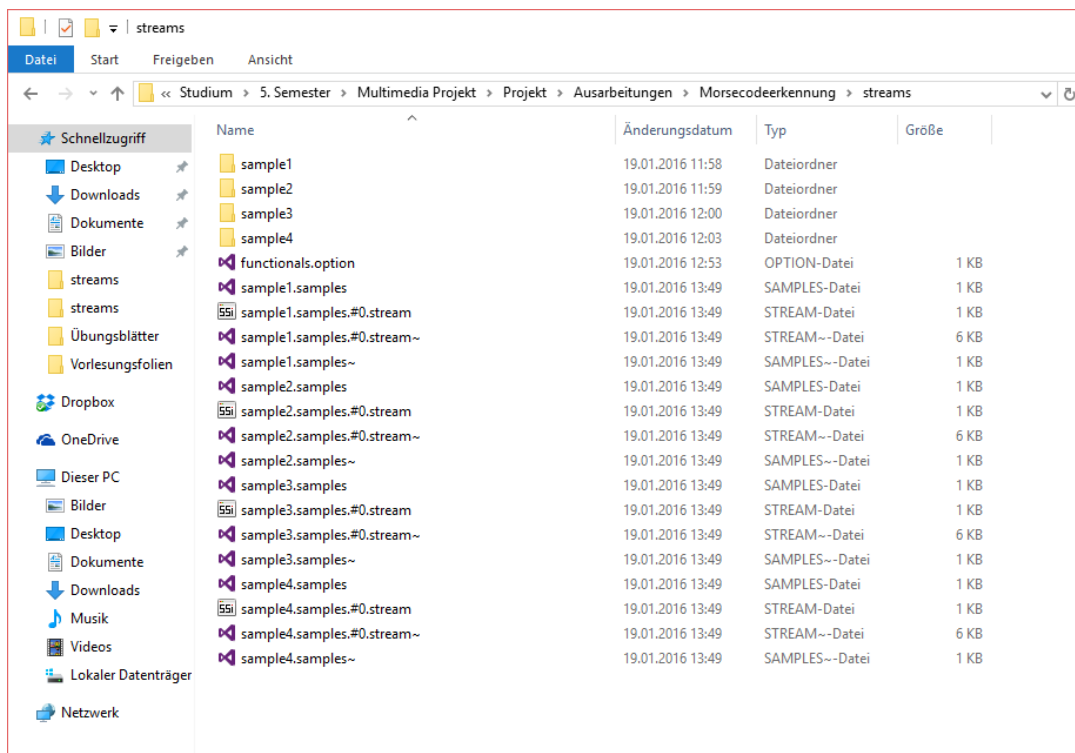


Abbildung 1

3 Anleitung

3.1 Das Grundgerüst

Zuerst muss eine Datei "trainModel-template.trainer" in dem Ordner, in dem sich auch die samples befinden, angelegt werden. Anschließend wird in dieser ein Grundgerüst für den Trainer erstellt:

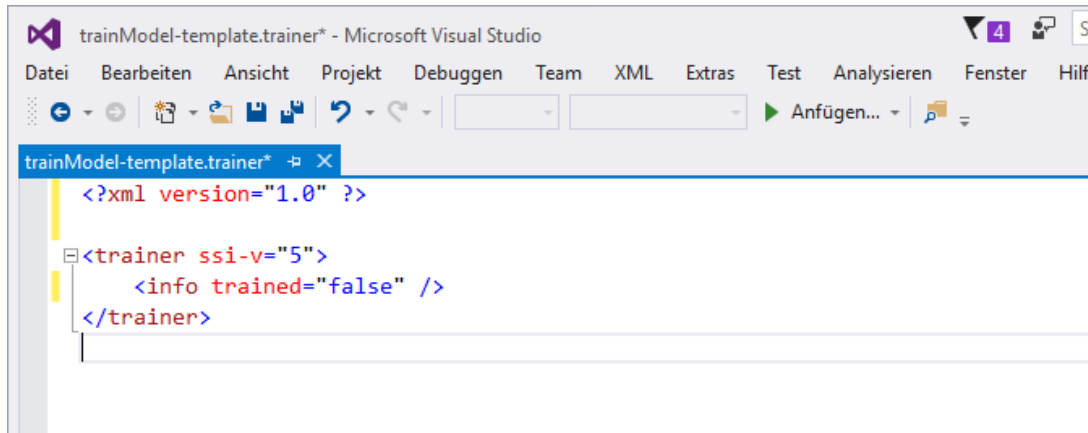
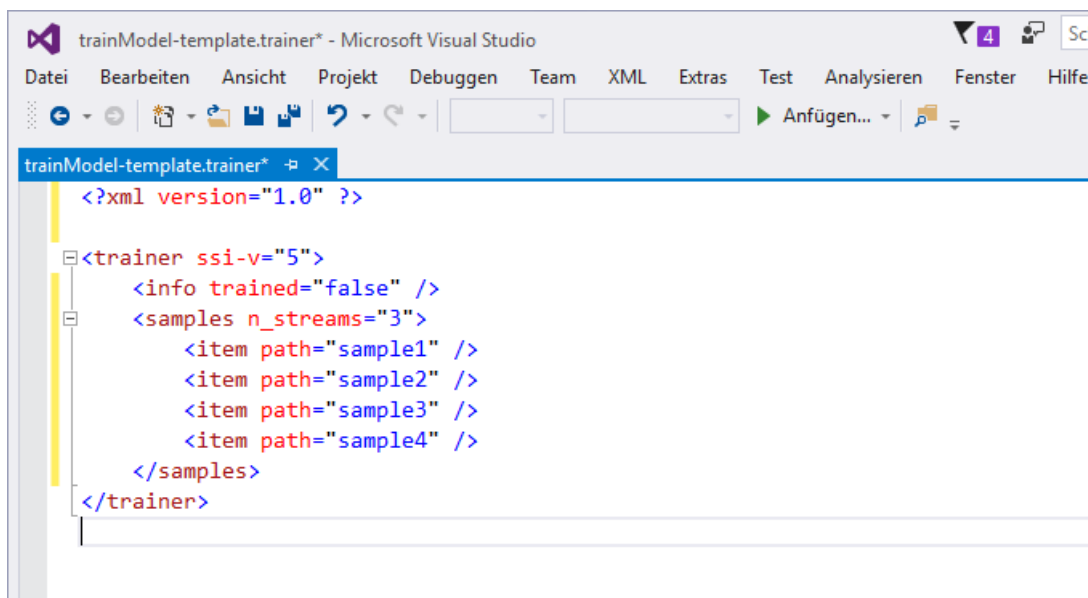


Abbildung 2

3.2 Einbinden der Samples

Für das Trainieren werden die oben genannten samples benötigt. Hierfür wird ein neuer Tag "samples" mit dem Attribut "n_streams="s"", wobei s die Anzahl aller Eingangskanäle ist, eingefügt. Die Nummerierung der streams beginnt bei 0 mit dem ersten in der Aufnahme-Pipeline angegebenen Eingabestroms (ein Eingabestrom, z.B. Accelerometer mit x-, y- und z-Achse, kann auch mehrere streams enthalten). Innerhalb der Tags werden die einzelnen streams als "item" mit dem Attribut "path="p"" hinzugefügt. p ist der Name des samples, der beim Aufruf von anno2sample.exe festgelegt und als Dateiname verwendet wird. Die Trainerdatei für die in Abbildung 1 gezeigten Daten folgendermaßen aussehen:



3.3 Features und Filter

Im nächsten Schritt wird eine Pipeline für die Bearbeitung und Auswertung der Daten erstellt. Diese Pipeline befindet sich in einem weiteren Tag "transform". Innerhalb dessen können Featureberechnungen und Filter in beliebiger Reihenfolge angewandt werden, um eine gute Klassifikation zu ermöglichen. Jedes Element ist ein "item" Objekt mit dem Attribut "create="i"", wobei i das zu erstellende Objekt ist. Die Menge aller möglichen Items ist in der Dokumentation nachlesbar. Als weiteres Attribut muss "stream="k"" hinzugefügt werden um festzulegen welcher Eingabestrom verarbeitet werden soll. Das Attribut "stream" ist nicht zu verwechseln mit den oben eingebundenen samples. k gibt hier wiederum an welcher stream verwendet wird. Gibt es mehrere streams, müssen die Features und Filter müssen für jeden angegeben werden.

Zusätzlich können die einzelnen Featureberechnungen und Filter durch die Angabe weiterer Attribute näher spezifiziert werden. Details sind ebenfalls in der Dokumentation zu finden. Die Fortsetzung der obigen Beispiele könnte also beispielsweise so aussehen:

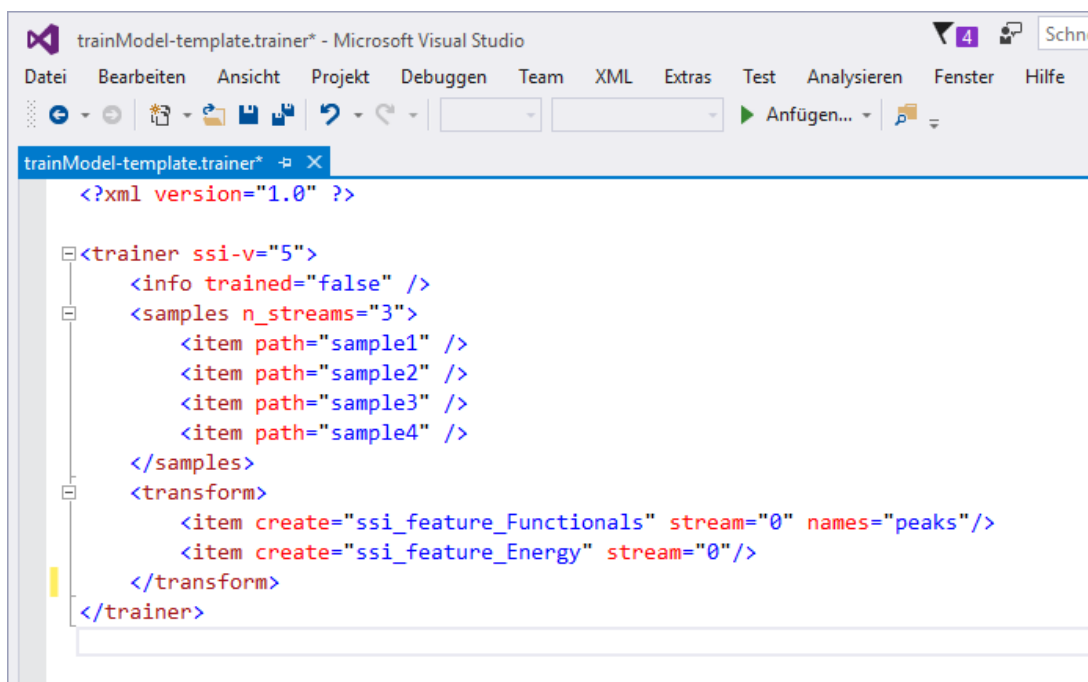


Abbildung 4

3.4 Der Klassifikator

Als letztes muss noch der anzuwendende Klassifikator gewählt werden. Hierfür wird nach dem transform-Block ein Tag "model" erstellt. Analog zu den Featurs muss mit "create="j"" ein spezifischer Klassifikator j und mit "stream="k"" der verwendete Eingabestrom angegeben werden. Es ist wiederum für jeden Eingabestrom ein Klassifikator anzugeben. Sollen mehrere Entscheidungen der Klassifikatoren fusioniert werden, müssen die Betroffenen in einen Tag "fusion" mit dem Attribut "create="n"", wobei n der Name der Fusionsfunktion ist, eingefasst werden. Details zu weiteren Attributen sind wiederum in der Dokumentation zu finden. Eine mögliche, vollständige Pipeline könnte so aussehen:

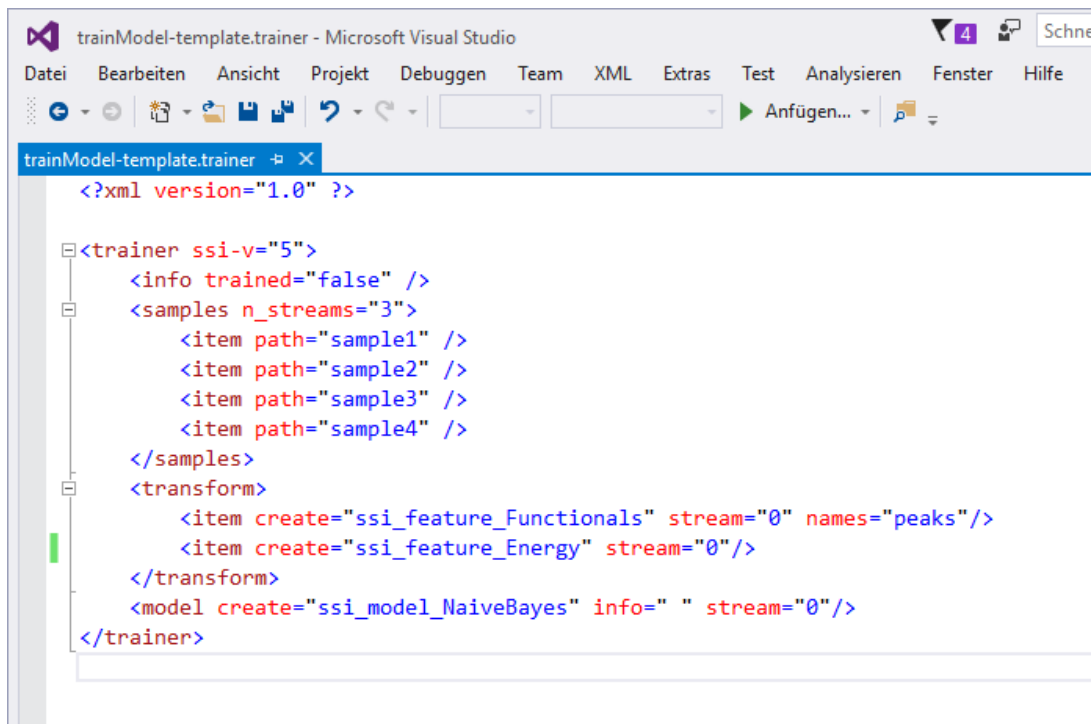


Abbildung 5

3.5 Anwenden des Klassifikators

Für die Anwendung des Klassifikators lohnt sich die Erstellung eines Kommandozeilen-skripts um wiederholte Eingaben in die Befehlszeile zu vermeiden, da häufig unterschiedliche Parameter in der Pipeline verändert und getestet werden um bessere Ergebnisse zu erzielen. Das Skript sieht wie folgt aus und ändert sich gewöhnlich nicht.

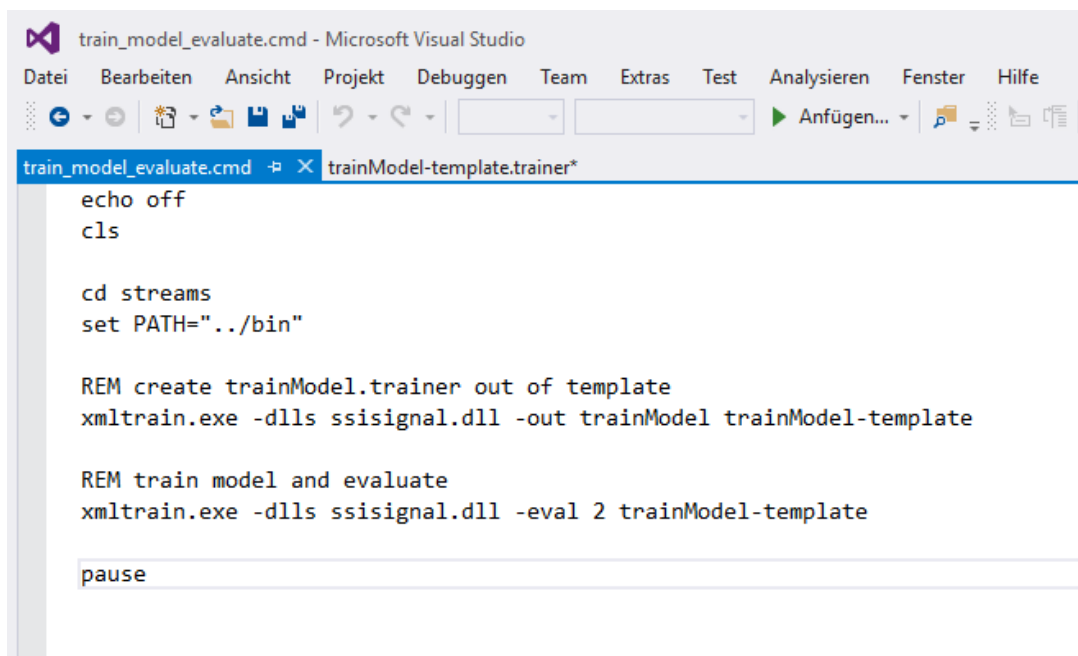
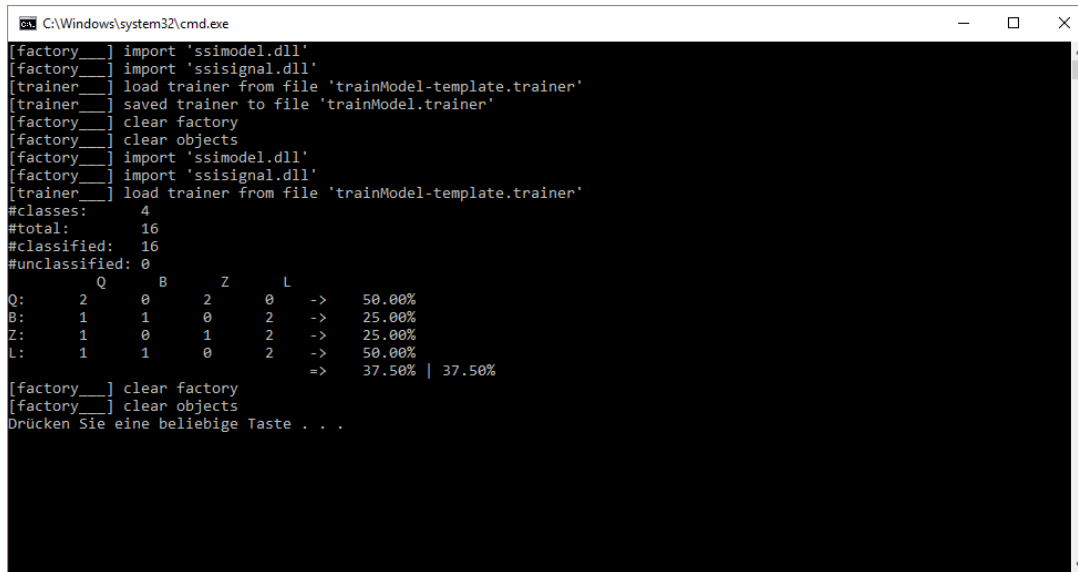


Abbildung 6

Wird diese ausgeführt erscheint die Auswertung in der Konsole:



```
C:\Windows\system32\cmd.exe
[factory_] import 'ssimodel.dll'
[factory_] import 'ssisignal.dll'
[trainer_] load trainer from file 'trainModel-template.trainer'
[trainer_] saved trainer to file 'trainModel.trainer'
[factory_] clear factory
[factory_] clear objects
[factory_] import 'ssimodel.dll'
[factory_] import 'ssisignal.dll'
[trainer_] load trainer from file 'trainModel-template.trainer'
#classes: 4
#total: 16
#classified: 16
#unclassified: 0
      Q      B      Z      L
Q:      2      0      2      0  ->  50.00%
B:      1      1      0      2  ->  25.00%
Z:      1      0      1      2  ->  25.00%
L:      1      1      0      2  ->  50.00%
=> 37.50% | 37.50%
[factory_] clear factory
[factory_] clear objects
Drücken Sie eine beliebige Taste . . .
```

Abbildung 7

Zu sehen ist welche samples einer bestimmten Klasse in ein Klasse eingeordnet wurden und die Statistik in Prozentangaben, wie genau der Klassifikator ist.

Das Ergebnis in Abbildung 7 ist 37,5% bei 4 Klassen mit einer Zufallswahrscheinlichkeit von 25%. Das bedeutet der Klassifikator ist 12,5% Prozentpunkte besser als der Zufall. Durch Änderungen an der Trainer-Pipeline würde sich das Ergebnis ändern. Die geeinte Wahl der Methoden und Klassifikatoren liegt in den Händen des Nutzers.

4 Verweise

Für das Schreiben eines Trainers sind die Dokumentationen von "ssisignal.dll" und "ssi-model.dll" sehr hilfreich. Hier finden Sie die möglichen Features, Filter und Klassifikatoren sowie deren Parameter.