Ein SAS Enterprise Miner Interface für systematisches Hyperparameter-Tuning mit (ml) R

Jannes Rausch-Stroomann
Hochschule Stralsund
Zur Schwedenschanze 15
18435 Stralsund
jannes.rausch@gmx.de

Gero Szepannek
Hochschule Stralsund
Zur Schwedenschanze 15
18435 Stralsund
gero.szepannek@hochschule-stralsund.de

Zusammenfassung

Es wird eine Möglichkeit vorgestellt, Hyperparameter von Machine Learning Verfahren innerhalb des SAS Enterprise Miners automatisiert zu tunen. Die Infrastruktur des SEMMA Prozesses, wie er im SAS Enterprise Miner implementiert ist gestattet pro Knoten je ein parametrisiertes Modell und damit nur ein manuelles Hyperparameter-Tuning. Die Vorteile einer automatischen Hyperparameter-Optimierung liegen dabei auf der Hand: ein in der Regel geringerer Zeitaufwand bei der Modellentwicklung sowie oft eine höherer Modellgüte. Die vorgeschlagene Umsetzung nutzt den Open Source Integration Knoten zur Einbindung des R Pakets mlr. Hierfür sind keine tiefergehenden R- bzw. SAS-Kenntnisse erforderlich. Der vorgestellte Code kann durch wenige Anpassungen auch auf zahlreiche andere Klassifikations- und Regressionsverfahren und -probleme angepasst werden. In einer abschließenden Benchmarkstudie konnte eine signifikante Verbesserung der Modellgüte auf einem echten Datensatz nachgewiesen werden.

Schlüsselwörter: Machine Learning, SAS Enterprise Miner, Hyperparameter Tuning, R, mlr, irace, Open Source Integration Node

1 Einleitung

Nahezu sämtliche Verfahren des maschinellen Lernens erfordern die Spezifikation sogenannter Hyperparameter, wobei keine allgemeinen Regeln für die Wahl der Parameter bestehen, sondern diese vom jeweiligen Anwendungskontext abhängig ist (Wolpert & Macready, 1995).

Innerhalb des SAS Enterprise Miners lassen sich die implementierten Machine Learning Algorithmen über deren Properties Panels parametrisieren. Zur Identifikation einer guten Parametrisierung lassen sich unterschiedlich Parametrisierte Modelle anhand des Model Comparison Knotens miteinander vergleichen. So kann sich durch Testen verschiedener Parameterkombinationen einem besseren Modell genähert werden.

Dieser Prozess eines manuellen Hyperparametertunings ist jedoch zeitaufwändig und die Vielzahl an Parametern, die ein Algorithmus besitzen kann- sowie die große Spannweite der Wertebereiche einzelner Parameter von teilweise mehreren Größenord-

nungen erfordern eine hohe Anzahl zu testender Modelle, ohne dabei die Garantie zu liefern in die Nähe der bestmöglichen Performance zu gelangen.

In der vorliegenden Arbeit wurde die Möglichkeit entwickelt, aus dem SAS Enterprise Miner heraus den Hyperparameter-Tuningprozess automatisiert durchzuführen und somit zeitsparend eine zielgerichtete Suche im Hyperparameterraum durchzuführen. Hierfür wird der Open Source Integration Knoten unter Einbeziehung des R Pakets „mlr“ verwendet.

Der Rest der Arbeit gliedert sich wie folgt: In Kapitel 2 werden zunächst kurz der Modellierungsprozess innerhalb des SAS Enterprise Miners, sowie insbesondere der für die Arbeit erforderliche Open Source Integration Knoten beschrieben. Kapitel 3 stellt den irace Algorithmus vor, der für ein systematisches Hyperparametertuning genutzt werden kann, sowie das R Paket „mlr“, über dessen Einbindung das Tuning aus dem SAS Enterprise Miner heraus möglich ist. In Kapitel 4 wird die konkrete Umsetzung des Hyperparametertunings innerhalb eines SAS EM Modellierungsprozesses inklusive des erforderlichen Codes entwickelt. Abschließend werden in Kapitel 5 die Ergebnisse einer Benchmark-Studie vorgestellt, die zeigt, dass es möglich sein kann, mit Hilfe automatisierten Hyperparameter-Tunings deutliche Verbesserungen in der Prognosefähigkeit zu erzielen und Kapitel 6 beinhaltet eine Zusammenfassung der Ergebnisse.

2 SAS Enterprise Miner Software und der SEMMA Prozess

Der SAS Enterprise Miner liefert eine benutzerfreundliche grafische Oberfläche, mit der sich unkompliziert statistische Modelle im Data Mining Kontext entwickeln lassen. Allgemein lässt sich der Modellierungsprozess dabei anhand der **SEMMA** Prozesschritt-Abfolge (Sample Explore Modify Model Assess, vgl. Santos & Azevedo, 2008) beschreiben. Das folgende Diagramm zeigt eine einfache Umsetzung innerhalb eines Enterprise Miner Diagramms, wobei auf den Sample-Schritt im Data Paritioning Knoten direkt die Modellierung – hier durch einen Random Forest (vgl. z.B. Szepannek, Lieber, & Bohnen, 2018) erfolgt. Durch die so erzeugte Schrittfolge wird ein Modell basierend auf einer prespezifizierten Kombination von Parametern erstellt und evaluiert.

Abbildung 1: Beispiel Umsetzung Modellierung entlang des SEMMA Prozesses

Durch Schnittstellen zur Integration SAS Code und R Code bietet der Enterprise Miner zusätzlich einen hohen Grad an Flexibilität und ermöglicht dessen Erweiterung um zusätzliche Funktionalitäten. Im Folgenden Abschnitt sei der Open Source Integration Knoten beschrieben, über den die Einbindung von R Code ermöglicht wird.

2.1 Open Source Integration Node

Als Schnittstelle der SAS Software zu R kann der Open Source Integration Node genutzt werden. Dieser ermöglicht das Ausführen von R Code, für den im Properties-Panel ein Editor bereitgestellt wird (vgl. Abb. 2). Durch Ausführen des Knotens wird ein R Prozess gestartet, der den im Editor geschriebenen Code ausführt. Um die Funktionalität nutzen zu können ist von daher zunächst eine R Installation erforderlich. Hierzu sind die folgenden Schritte erforderlich:

- Installation von R (R Core Team, 2017) sowie den Paketen „PMML“ (Guazzelli, Zeller, Lin, & Williams, 2009) und „mlr“ (Bischl, et al., 2016).
- Anlegen der Umgebungsvariable „R_HOME“ mit dem Pfad der entsprechenden R Version.
- Hinzufügen der Option „-RLANG“ in die SAS Konfigurationsdatei (sasv9.cfg).

Welche R Version benötigt wird kann online eingesehen werden (Usage Node 54806). Bei einer inkompatiblen R Version kann es Probleme mit der Verbindung geben.

Bei der Erstellung von R Code aus dem SAS Enterprise Miner heraus erfolgt das ex- bzw. importieren von Daten über Data Handles. Hierfür existieren folgende nützliche Macrovariablen (SAS Institute Inc., 2016, S. 1087):

- &EMR_IMPORT_DATA — Trainings Daten
- &EMR_IMPORT_VALIDATE — Validierungs Daten
- &EMR_IMPORT_TEST — Test Daten
- &EMR_IMPORT_SCORE — Score Daten
- &EMR_IMPORT_TRANSCATION — Transaction Daten

Im Reiter „Imported Data“ des Open Source Integration Knotens befindet sich eine tabellarische Übersicht verfügbarer Macrovariablen. Einzelne Variablen können für die Modellierung über ihren Variablenamen verwendet werden (s. Reiter „Variables“ im Open Source Integration Knoten). Zusätzlich stehen weitere Variable Handles zur Modellspezifikation zur Verfügung, die Variablen desselben Typs zusammenfassen (SAS Institute Inc., 2016, S. 1086):

- &EMR_NUM_INPUT
- &EMR_CLASS_INPUT
- &EMR_CHAR_INPUT
- &EMR_NUM_TARGET
- &EMR_CLASS_TARGET

Bei der Ausgabe des Knotens verschiedene Modi zur Verfügung, je nach verwendeten R Funktionen: „PMML“, „Merge“ und „None“. Eine Übersicht bietet die Tabelle 1.

Tabelle 1: Open Source Integration Node Output Mode (SAS Inst., 2016, 1081-1082)

Output Mode	Training Mode	Suggested Use: Example Packages	Assessment Requirements	Output Column Role
PMML	Supervised	Basic predictive modeling packages: lm, glm, rpart, nnet, and multinom	not available	Prediction
	Unsupervised	k-means clustering package: kmeans	Segment Profile node	Distance, Segment
Merge	Supervised	Advanced predictive modeling packages: gbm, party, and randomForest	Model Import node	Prediction
	Unsupervised	Unsupervised models and data transformation packages: svd and NMF	not available	Input
None	not available	Debugging, graphics, data exploration, and simulation	not available	not available

Der **„PMML" Outputmodus** (Guazzelli, Zeller, Lin, & Williams, 2009) bietet die Möglichkeit, aus einem in R erstellten Modell in einen DATA Step erzeugt, dass komfortabel dessen Weiterverwendung innerhalb von SAS Base / SAS Enterprise Miner ermöglicht, z.B. zur Vorhersage neuer Daten. Der PMML Outputmodus ist derzeit allerdings nur für eine eingeschränkte Auswahl von Verfahren implementiert (**lm** [lineare Modelle], **multinom** [multinomiale log-lineare Modelle], **glm** [generalisierte lineare Modelle], **rpart** [Recursive Partitioning-Entscheidungsbäume], **nnet** [einfache neuronale Netze] und **kmeans** [k-means Clustering], vgl. SAS Institute Inc., 2016, S. 1084).

So ist es ohne weiteres möglich, alternative Modelle im SAS Enterprise Miner und in R zu erzeugen und über den Model Comparison Node des SAS Enterprise Miners auszuwerten und miteinander zu vergleichen.

Anders im PMML Modus wird im **„Merge" Modus** (vgl. Abb. 2) kein SAS Code generiert, der Modus ist jedoch auch für beliebige andere R Pakete des überwachten- und unüberwachten Lernens anwendbar. Dies ermöglicht es, auch komplexere Aufgaben in R über den Open Source Integration Knoten auszuführen.

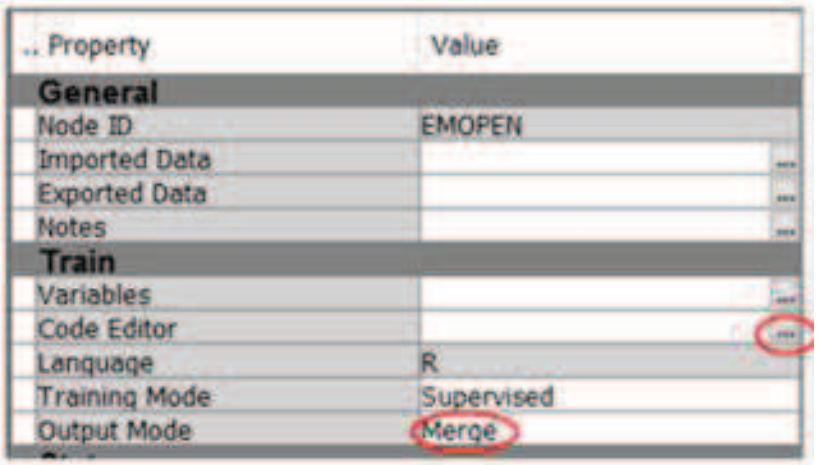

Abbildung 2: Auszug des Properties Panel für den Open Source Integration Knoten

Über den „Model Import Knoten" ist es auch hier möglich, die Ausgaben zurück zu importieren und mit anderen Modellen zu vergleichen. Hierfür müssen zunächst die Vorhersagen in Form von Data Frames in R erzeugt und anschließend Macrovariablen für den Export zugewiesen werden (SAS Institute Inc., 2016, S. 1085):

- &EMR_EXPORT_TRAIN — Trainings Daten
- &EMR_EXPORT_VALIDATE — Validierungs Daten
- &EMR_EXPORT_TEST — Test Daten
- &EMR_EXPORT_SCORE — Score Daten
- &EMR_EXPORT_TRANSACTION — Transactions Daten

Ein einfaches Beispiel zur Erstellung eines Random Forest Modells in R zeigt Abbildung 3:

Code Editor

```
library(randomForest)
&EMR_MODEL = randomForest(&EMR_CLASS_TARGET ~ &EMR_NUM_INPUT, data = &EMR_IMPORT_DATA)
&EMR_EXPORT_TRAIN    = predict(&EMR_MODEL, &EMR_IMPORT_DATA, type = "prob")
&EMR_EXPORT_VALIDATE = predict(&EMR_MODEL, &EMR_IMPORT_VALIDATE, type = "prob")
head(&EMR_EXPORT_TRAIN)
```

Abbildung 3: Beispiel zur Verwendung der Data Handles

Über library(randomForest) wird zunächst die erforderliche Bibliothek eingebunden. Über Macrovariablen wird die Modellgleichung (&EMR_CLASS_TARGET ~ &EMR_NUM_INPUT) definiert und auf Basis der Trainingsdaten (&EMR_IMPORT_DATA) ein Modell (&EMR_MODEL) erstellt, das wiederum mit Hilfe der predict() Funktion zur Vorhersage der Trainings- und der Validierungsdaten verwendet wird.

Typische Fehler an der Schnittstelle zwischen dem SAS EM und R sind:

- die Kodierung fehlender Werte (hier kann ein voranstehender „Impute“ Knoten eingesetzt werden),
- nicht vorkommende Kategorien nominaler Variablen, in Trainings-, bzw. Validierungs- und Testdaten,
- eine ggf. durch die R-Funktion erzeugte unterschiedliche Anzahl an Beobachtungen von im- und exportierten Daten.

Abbildung 4 zeigt den Modellierungsprozess über Open Source Integration in Merge Modus und den anschließenden Model Import.

Abbildung 4: Beispiel Modellierungsprozess über Open Source Integration im Merge Modus

Um die Vorhersagen im SAS Enterprise Miner nutzen zu können, müssen diese zunächst über einen „Model Import“ Knoten importiert werden. Hierbei ist es erforderlich, den Import Type = Data zu spezifizieren und im „Mapping Editor“ die „Modeling Variable“ der richtigen „Predicted Variable“ des exportierten R Data Frames zuzuordnen (vgl. Abb. 5).

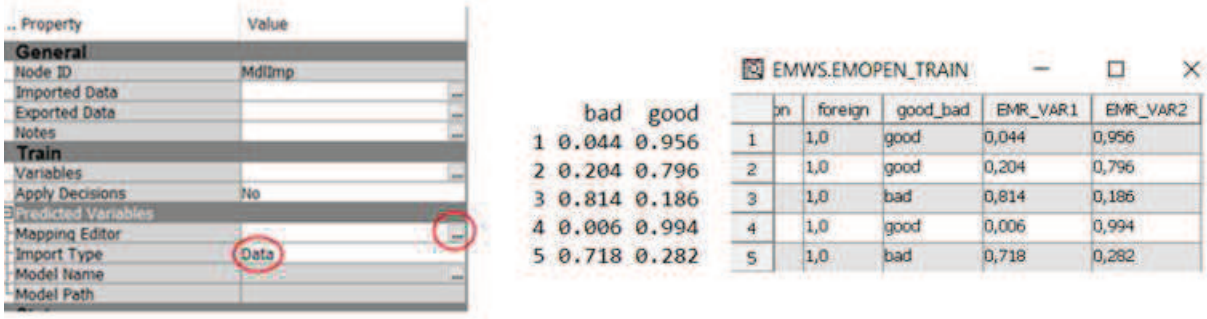

Abbildung 5: Auszug des Properties Panel für den Model Import Knoten (links). Vergleich der R Ausgabetabelle (Mitte) mit der in der von Open Source Integration Knoten exportierten Tabelle (rechts)

Abbildung 5 (Mitte) zeigt die mit dem Befehl head(&EMR_EXPORT_TRAIN) (vgl. Abb. 3) erzeugte Ausgabe der ersten Zeilen der Vorhersage (im Result-Fenster des Open Source Integration Knotens). Dieser kann (nach der Ausführung) mit den Spalten der vom Knoten exportierten SAS-Tabelle (EMOPEN<ID>_train über den SAS Enterprise Miner Explorer) (vgl. Abb. 5, rechts) verglichen und im Mapping Editor zugewiesen werden (Abb. 6).

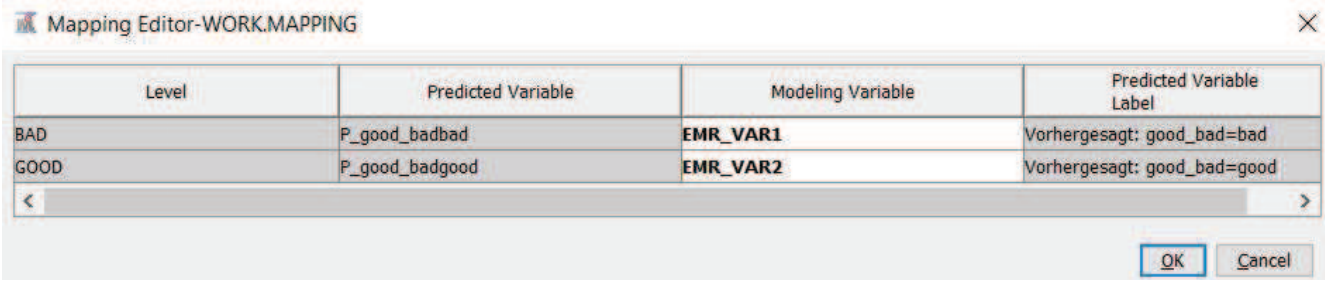

Abbildung 6: Mapping Editor

Auf den Ausgabemodus „None“, der beliebigen R Code unterstützt, jedoch keine Möglichkeit zum Import von Ergebnissen bietet, soll hier nicht weiter eingegangen werden.

3 Hyperparametertuning

3.1 Iterated F-Racing

Im Hyperparametertuning werden die Modellparameter hinsichtlich eines Gütekriteriums optimiert. Hierzu werden systematisch Modelle mit verschiedener Parametrisierung trainiert und auf Validierungsdaten evaluiert. Ein einfacher Ansatz zur Lösung ist das automatisierte Parametertuning mit einer „Brute Force“ Methode, einer Gittersuche, bei der sämtliche möglichen Kombinationen von Parametern getestet werden. Bei hoch parametrisierten Algorithmen kann die Anzahl der möglichen Kombinationen schnell die zur Verfügung stehende Rechenzeit übersteigen. Einen Lösungsansatz für dieses Problem bildet iterated F-Racing (vgl. z.B. Bischl, Kühn, & Szepannek, 2016).

Die allgemeine Problemformulierung des iterated F-Racing (irace) beinhaltet:

- einen Algorithmus (hier: ein Machine Learning-Verfahren),
- der über einen Parametervektor θ konfigurierbar ist
- und hinsichtlich einer Zielfunktion $F(\theta)$ (hier: Performancekriterium) optimiert werden soll.

Die Optimalität soll hierbei bezüglich einer Menge möglicher Instanzen (in diesem Fall repräsentiert durch zufällige Training-/Holdout-Unterteilungen der Trainingsdaten) erzielt werden, wobei das Zielkriterium der erwarteten Performance entspricht.

Zur automatischen Algorithmus-Konfiguration werden im irace iterativ „Races“ gemäß der folgenden drei Schritte durchlaufen, bis ein Zielkriterium erfüllt ist (vgl. López-Ibáñezet al., 2017):

(1) Sampling von Konfigurationskandidaten θ aus unabhängigen, parameterspezifischen Verteilungen.
(2) Auswahl der besten Konfigurationen aus den erzeugten Kandidaten über Racing.
(3) Update der parameterspezifischen Verteilungen mit dem Ziel einer Verzerrung in Richtung der besten Kandidaten.

Jedes Race startet dabei mit einem endlichen Satz von Konfigurations-Kandidaten. In jedem Race-Schritt werden die Konfigurations-Kandidaten auf einer einzigen Instanz ($I_j \in I$) (in diesem Fall einer zufälligen Training-/Holdout-Unterteilung der Trainingsdaten) ausgewertet. Nach einer Anzahl von Schritten werden die Kandidaten, die statistisch schlechter[1] als mindestens ein anderer sind, verworfen. Das Verfahren setzt sich mit den restlichen Kandidaten fort und wird so lange fortgesetzt, bis eine minimale Anzahl von Konfigurationen (entsprechend den Spalten in Abb. 7) erreicht ist, eine maximale Anzahl von Instanzen (entsprechend den Zeilen in Abb. 7) verwendet wurde oder ein vordefiniertes Berechnungsbudget erschöpft wurde. Dieses Berechnungsbudget kann eine Gesamtberechnungszeit oder eine Anzahl von Experimenten sein, wobei ein Experiment die Anwendung einer Konfiguration auf eine Instanz ist (entsprechend einem Punkt in Abb. 7).

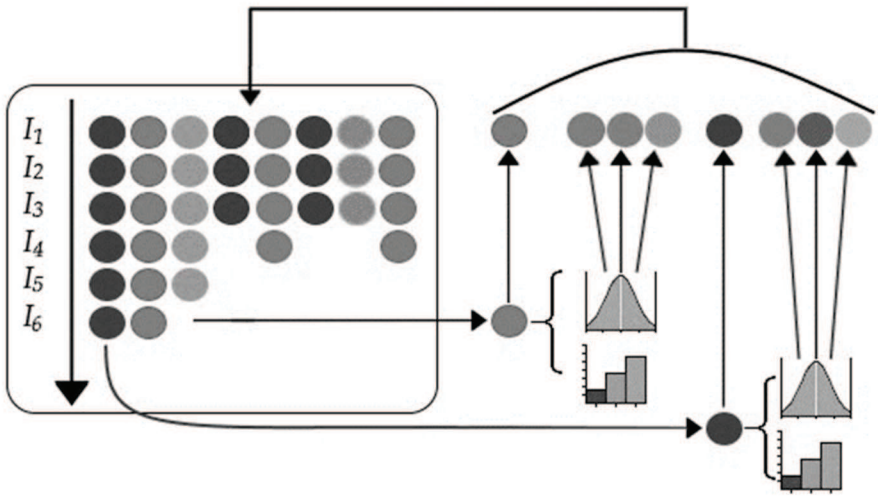

Abbildung 7: Schema des irace (López-Ibáñez, Dubois-Lacosteb, Cáceresb, Birattarib, & Stützleb, 2017)

3.2 Das R Paket „mlr“

Das R Paket „mlr“ (Bischl B. et al., 2016, Szepanneket al., 2010) bietet ein einheitliches Interface für eine Vielzahl von R Implementierungen von Verfahren der Klassifikation, Regression, Survival- sowie Clusteranalyse, einschließlich Parametrisierung, Resampling Performanceevaluierung und Vorverarbeitungen wie Imputation, Variablenselektion oder Korrektur unbalancierter Klassen. Insbesondere sind ebenfalls verschiedene Techniken zur automatisierten Hyperparameter-Optimierung implementiert.

Im Hyperparametertuning werden die Modellparameter hinsichtlich eines Gütekriteriums optimiert. Hierzu werden systematisch Modelle mit verschiedenen Parametrisierung trainiert und auf Validierungsdaten evaluiert. Hierfür existieren verschiedene wie z.B. eine einfache Gittersuche, Zufallssuche, sequentielle modellbasierte Optimierung

[1] Als Test wird Friedmans nichtparametrische ANOVA nach Rängen verwendet (vgl. Conover, 1999).

(SMAC) oder iterated F-Racing (irace, López-Ibáñez, Dubois-Lacosteb, Cáceresb, Birattarib, & Stützleb, 2017). Abbildung 8 zeigt ein Schema zur Umsetzung des Hyperparametertunings mit Hilfe von mlr, wie es für beliebige Lernsituationen und -verfahren angewendet werden kann.

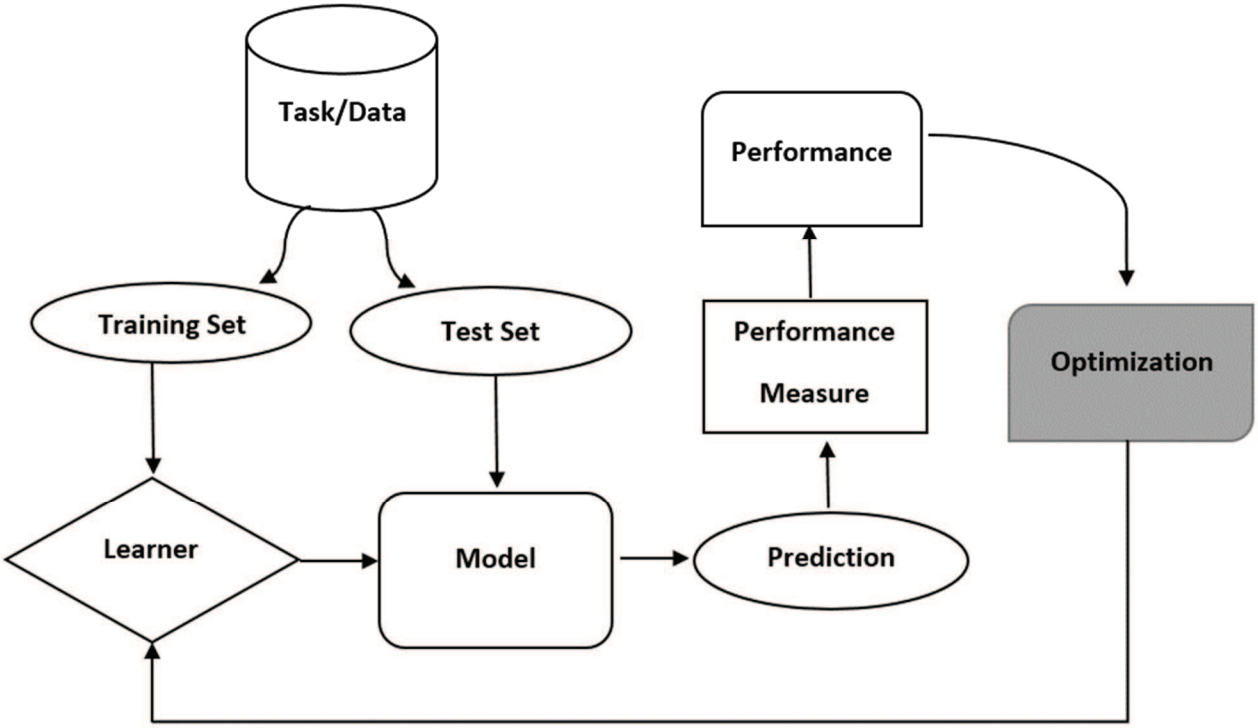

Abbildung 8: mlr Hyperparametertuning Schema (Bischl & Lang, Applied Machine Learning and Efficient Model Selection with mlr, 2015, S. 51)

4 Umsetzung

4.1 Prozess zum Hyperparameter-Tuning im SAS Enterprise Miner

Im Folgenden ist der Prozess beschrieben, wie eine Modelloptimierung via irace im SAS Enterprise Miner über die Einbindung des R Pakets mlr umgesetzt werden kann. Grundlage bildet der in Abb. 8 dargestellte Prozess. Das Hyperparameter-Tuning wird innerhalb des Open Source Integration Knotens ausgeführt und die Ergebnisse über den Model Import Node wieder importiert. Diese können dann mit Hilfe des Model Comparison Knotens evaluiert werden. Der Output Mode des Open Source Integration Node muss „Merge" sein. In den Code Editor kann der im folgenden Abschnitt dargestellte R-Code eingefügt werden. Für den Model Import muss der Import Type auf „Data" gesetzt sein und im Mapping Editor müssen wie beschrieben die die Namen der Variablen überprüft werden (vgl. Kap. 3).

4.2 Code für den Editor des Open Source Integration Knotens

Im folgenden Code müssen zunächst:

- eine Lernaufgabe (`task`)
- ein Lernverfahren (hier: Random Forest zur Klassifikation mit Vorhersagetyp Wahrscheinlichkeit, `predict.type = "prob"`)
- eine Menge von Parametern mit Wertebereich (hier: `ps` mit den Parametern `"ntree", "mtry"` und `"nodesize"`, vgl. R-Hilfe zu Random Forests)
- eine Optimierungsmethode (hier: `irace`)
- sowie eine resampling-Strategie (`rdesc`)

definiert werden.

Anschließend führt der Funktionsaufruf tuneParams() die Suche nach einer optimalen Parameterkombination im spezifizierten Suchraum (`ps`) hinsichtlich eines festzulegenden Performancemaßes (hier die Area under the ROC Curve: `measures = auc`) durch. Die resultierenden Parameter sind im Objekt `finetune` gespeichert.

Abschließend wird ein neuer Lerner (`optlrn`) mit den optimierten Parametern definiert und auf der Gesamtmenge der Parameter trainiert (`optmod`). Das so erzeugte Modell kann nun wie in Abschnitt 3 beschrieben zur Vorhersage der Trainings- und Validierungsdaten verwendet werden.

R Code:

```
library(randomForest)
library(mlr)

#Daten einlesen
bmdata = rbind(&EMR_IMPORT_VALIDATE,&EMR_IMPORT_DATA)

#task and learner
task = makeClassifTask(data = bmdata, target = "ry")
lrn  = makeLearner("classif.randomForest", predict.type = "prob",
fix.factors.prediction = TRUE)

#parameterset
ps = makeParamSet
(
  makeIntegerParam("ntree",lower = 50, upper = 2500),
  makeIntegerParam("mtry", lower = 3, upper = 10),
  makeIntegerParam("nodesize", lower = 5, upper = 175)
)

#resample und irace
rdesc    = makeResampleDesc(method="Holdout")
ctrl     = makeTuneControlIrace(maxExperiments = 800)
```

```
finetune = tuneParams(lrn, task = task, resampling = rdesc,
par.set  = ps, control = ctrl, measures = auc)
finetune

#We can generate a Learner with optimal hyperparameter settings as
follows:
optlrn = setHyperPars(makeLearner("classif.randomForest", pre-
dict.type = "prob"), par.vals = finetune$x)
optlrn
optmod = train(optlrn, task,subset = train.set)

#Import Model - Data Handle
&EMR_MODEL = optmod$learner.model
&EMR_EXPORT_TRAIN = predict(&EMR_MODEL,&EMR_IMPORT_DATA,
type="prob")
&EMR_EXPORT_VALIDATE = predict(&EMR_MODEL,&EMR_IMPORT_VALIDATE,
type="prob")
&EMR_EXPORT_TEST = predict(&EMR_MODEL,&EMR_IMPORT_TEST, type="prob")
&EMR_EXPORT_TRAIN[1:17,]
```

Der Code ist auf andere Lernverfahren übertragbar und ebenso für Regressionsprobleme nutzbar. Hierbei müssen folgende Definitionen bzw. können folgende Anpassungen vorgenommen werden:

- Learner: Spezifiziert das Lernverfahren. Aufruf der Funktion „listLearners()“ zeigt die verfügbaren (aktuell ≥ 160) Learner an.
- Parameterset: Definiert die gewünschten Parameter inklusive Wertebereich für das Hyperparameter-Tuning. Aufruf der Funktion „getParamSet(leaner)“ zeigt die verfügbaren Parameter und deren Wertebereich für einen Lerner an.
- Optimierungs Algorithmus, sowie ggf. Resamplingmethode.
- Optimierungskriterium, z.B.: „auc“ oder “mmce“ (minimum misclassification error).

5 Benchmark Experiment

In einem Benchmarkexperiment wurden jeweils Modelle und Optimierungsstrategien anhand von Realdaten auf einem binären Klassifikationsproblem zur Responsevorhersage im Bank-Marketing (Moro, Laureano, & Cortez, 2011) verglichen: Innerhalb des SAS Enterprise Miners wurden verschiedene

- Entscheidungsbaum-,
- Random Forest-
- sowie Boostingmodelle

erzeugt. Dabei wurden zunächst Modelle mit den Default-Einstellungen erstellt. Diese wurden anschließend durch manuelle Parameteroptimierung verbessert und abschließend mit Hilfe von irace einem automatischen Hyperparametertuning unterzogen (vgl. Rausch-Stroomann, 2017). Als Performancekriterium wurde die AUC betrachtet. Tabelle 2 zeigt das Verbesserungspotenzial durch sowohl manuelles als auch automatisiertes

Hyperparametertuning. Das Ausführen des automatisierten Tunings dauerte dabei für den Decision Tree 15 Minuten (Budget=500), für den Random Forest 17 Stunden und 20 Minuten (Budget=1200) und für Boosting (XGBoost) 2 Stunden und 40 Minuten. (Budget=800) (i7 4770, 16gb Ram)

Tabelle 2: AUC der Modelle

	Decision Tree	**Random Forest**	**Gradient Boosting/XGBoost**
Default	0,7185	0,8045	0,7665
Manuell	0,727	0,807	0,806
Automatisch	0,769	0,814	0,8227 (XGBoost)

Weiterhin wurde zum Vergleich von manuellem und automatisiertem Parametertuning ein Experiment mit zwei Data Mining Kursen an der Hochschule Stralsund durchgeführt. Im Rahmen einer Kaggle Inclass Competition[2] wurden die oben beschriebenen Daten (d.h. Training + Validierung) 18 Studierendenteams zur Verfügung gestellt, zusammen mit einem Testsample mit unbekannter Zielgröße und dem Ziel das bestmögliche Vorhersagemodell (im Sinne der AUC) zu generieren. Die Vorhersagen konnten von den Teilnehmern per Upload nach kaggle evaluiert werden. Abbildung 9 (links) zeigt die Entwicklung des Gütekriteriums auf den Kaggle-Testdaten im zeitlichen Verlauf durch die manuelle Hyperparameter-Optimierung der Studierendenteams. Schnell stellt sich eine Sättigung der Performancekurve im Bereich einer AUC von etwa 0.8, die dann durch das Gewinnerteam auf 0.8035 verbessert werden konnte. Die maximale Anzahl an Submissions pro Team betrug hierbei 48. Im Vergleich hierzu konnte mit einer einzigen Submission nach automatischen Hyperparameter-Tuning nochmals eine deutliche Verbesserung auf eine AUC von 0.8198 erzielt werden (bestes Modell mit XGBoost, vgl. Chen & Guestrin, 2016). Dies stellt eine signifikante Modellverbesserung der AUC dar (p-Wert 0.00697, vgl. Henking, Blum, & Fahrmeir, 2006).

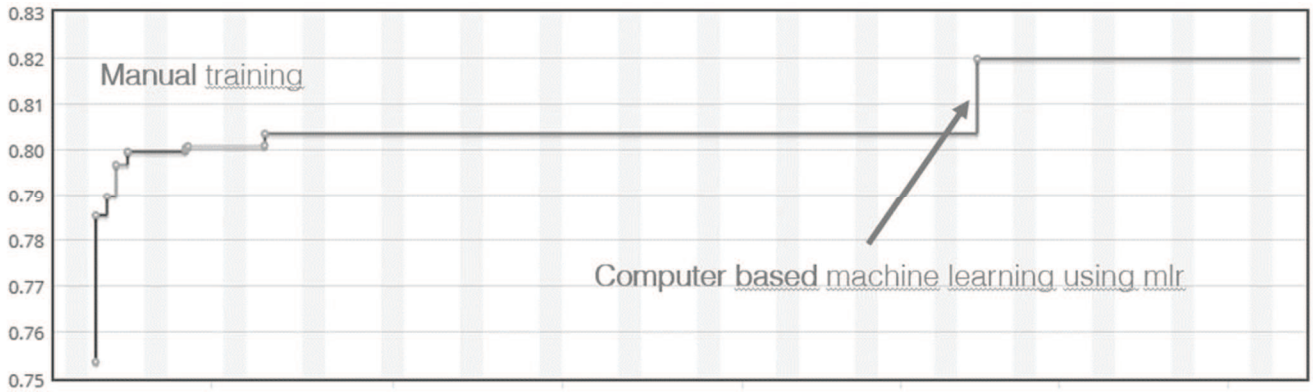

Abbildung 9: Kaggle Graph

[2] https://www.kaggle.com/about/inclass/overview

6 Zusammenfassung

Eine Methode zur Umsetzung eines automatisierten Hyperparametertunings aus dem SAS Enterprise Miner heraus wurde vorgestellt. Diese nutzt die Funktionalität zur Integration von R Code über den Open Source Integration Knoten und das R Paket mlr. Der vorgestellte Code ist übertragbar und bietet Zugang zu mehr als 100 Verfahren des Machine Learning für Klassifikations- und Regressionsprobleme.

In einem Benchmarkexperiment konnte durch die vorgestellte Methodik auf echten Daten aus dem Bankmarketing eine signifikante Verbesserung der Prognosegüte erzielt werden.

Literatur

[1] Bischl, B., Kühn, T., & Szepannek, G. (2016). On Class Imbalance Correction for Classification Algorithms in Credit Scoring. In M. Luebbecke, A. Koster, P. Letmathe, R. Madlenerm, B. Peis, & G. Walther, *Proc. OR 2014* (S. 37-43). Berlin: Springer.

[2] Bischl, B., Lang, M., Kotthoff, L., Schiffner, J., Richter, J., Studerus, E., . . . Jones, Z. (2016). mlr: Machine Learning in R. *JMLR 17 (170)*, 1-5.

[3] Chen, T., & Guestrin, C. (2016). *XGBoost: a Scalable Tree Boosting System.* https://arxiv.org/abs/1603.02754.

[4] Conover, W. (1999). *Practical nonparametric statistics, third edition.* New York: John Wiley & Sons.

[5] Guazzelli, A., Zeller, M., Lin, W., & Williams, G. (2009). PMML – an Open Standard for Sharing Models. *R Journal 1/1*, S. 60-65.

[6] Henking, A., Blum, C., & Fahrmeir, L. (2006). *Kreditrisikomessung – Statistische Grundlagen, Methoden und Modellierung.* Berlin: Springer.

[7] Bischl, B., Kühn, T., & Szepannek, G. (2016). On Class Imbalance Correction for Classification Algorithms in Credit Scoring. In M. Luebbecke, A. Koster, P. Letmathe, R. Madlenerm, B. Peis, & G. Walther, *Proc. OR 2014* (S. 37-43). Berlin: Springer.

[8] Bischl, B., Lang, M., Kotthoff, L., Schiffner, J., Richter, J., Studerus, E., . . . Jones, Z. (2016). mlr: Machine Learning in R. *JMLR 17 (170)*, 1-5.

[9] Chen, T., & Guestrin, C. (2016). *XGBoost: a Scalable Tree Boosting System.* https://arxiv.org/abs/1603.02754.

[10] Conover, W. (1999). *Practical nonparametric statistics, third edition.* New York: John Wiley & Sons.

[11] Guazzelli, A., Zeller, M., Lin, W., & Williams, G. (2009). PMML – an Open Standard for Sharing Models. *R Journal 1/1*, S. 60-65.

[12] Henking, A., Blum, C., & Fahrmeir, L. (2006). *Kreditrisikomessung – Statistische Grundlagen, Methoden und Modellierung.* Berlin: Springer.

[13] López-Ibáñez, M., Dubois-Lacosteb, J., Cáceresb, L. P., Birattarib, M., & Stützleb, T. (31. 01 2017). *The irace package: Iterated racing for automatic algorithm configuration*. Von sciencedirect: http://www.sciencedirect.com/science/article/pii/S2214716015300270#bib0010 abgerufen.

[14] Moro, S., Laureano, R., & Cortez, P. (2011). Using Data Mining for Bank Direct Marketing: An Application of the CRISP-DM Methodology. In P. e. Novais, *Proc. European Simulation and Modelling Conference* (S. 117-122).

[15] R Core Team. (2017). *R: A Language and Environment for Statistical Computing.* Wien: R Foundation for Statistical Computing. Von https://www.R-project.org/ abgerufen.

[16] Rausch-Stroomann, J. (2017). *Evaluierung der Möglichkeiten zur R Integration in die SAS Enterprise Miner Software mit dem Ziel eines systematischen hyperparameter-Tunings.* Bachelorarbeit, Hochschule Stralsund.

[17] Santos, A., & Azevedo, M. (2008). KDD, SEMMA and CRISP-DM: A Parallel Overview. *Europ. Conf. Data Mining (IADIS)*, S. 185-188.

[18] SAS Institute Inc. (1. November 2016). *SAS® Enterprise Miner™ 14.2: Reference Help.* Von https://support.sas.com/documentation/onlinedoc/miner/em142/emref.pdf abgerufen.

[19] Szepannek, G., Gruhne, M., Bischl, B., Krey, S., Harczos, T., Klefenz, F., & Weihs, C. (2010). Perceptually Based Phoneme Recognition in Popular Music. In H. Locareck-Junge, & C. Weihs, *Classification as a Tool for Research* (S. 751-758). Heidelberg: Springer.

[20] Szepannek, G., Lieber, D., & Bohnen, F. (2018). Methoden des Data Mining. In J. Deuse, B. Lotter, V. Hasselmann, & B. Konrad, *Industrial Engineering.* Heidelberg: Springer, erscheint.

[21] Wolpert, D., & Macready, W. (1995). *No free lunch theorems for search.* Technical Report SFI-TR-95-02-010; Santa Fe Institute.