FORSCHUNGSVEREINIGUNG AUTOMOBILTECHNIK E.V. | FAT-SCHRIFTENREIHE

FAT 264

Entwicklung von Methoden zur zuverlässigen Metamodellierung von CAE Simulations-Modellen

VDA | Verband der Automobilindustrie

Entwicklung von Methoden zur zuverlässigen Metamodellierung von CAE Simulations-Modellen

Forschungsstelle:
divis intelligent solutions GmbH

Bearbeiter:
Prof. Dr. Thomas Bäck
Dr.-Ing. Peter Krause
Christophe Foussette

Das Forschungsprojekt wurde mit Mitteln der Forschungsvereinigung Automobiltechnik e. V. (FAT) gefördert.

Inhaltsverzeichnis

1.	**Einleitung**	3
2.	**Metamodellierung**	4
2.1.	Einleitung	4
2.2.	Fehlermaße in der Regression	4
2.3.	Regressionstechniken	6
2.3.1.	Lineare Modelle	6
2.3.2.	Support Vector Machine	7
2.3.3.	Entscheidungsbäume (CART)	9
2.3.4.	Neuronale Netze	9
2.3.5.	Fuzzy-Modelle	10
2.4.	Modelloptimierung	11
2.5.	Modellauswahlkriterien	11
2.6.	Modellkomplexität	13
3.	**Empirische Untersuchung der Modellauswahlkriterien**	15
3.1.	Plausibilität von GDoF	15
3.2.	Versuchsaufbau	17
3.3.	Ergebnisse	19
4.	**Robustheitsanalyse**	20
4.1.	Robustheit in der Metamodellierung	20
4.2.	Metrik zur Robustheit	20
5.	**Anwendung auf Datensätze der Arbeitskreismitglieder**	22
5.1.	Datenbeschreibung	22
5.2.	Aggregierte Modellierungsergebnisse	24
6.	**Zusammenfassung/Ausblick**	29
7.	**Literaturhinweise**	31
A.	**Anhang**	34
A.1.	Matlab Code zur Modelloptimierung	34
A.2.	Detaillierte Modellierungsergebnisse der zur Verfügung gestellten Datensätze	43

1 Einleitung

Der FAT AK 27 UA Optimierung ist eingerichtet worden, um Werkzeuge und Methoden zur Optimierung von Fahrzeugstrukturen zu entwickeln. Ein Bestandteil hiervon ist die Generierung von zuverlässigen Metamodellen, die bei einer Optimierung die Rolle der sehr rechenintensiven FE-Simulationen übernehmen sollen. Das in diesem Bericht beschriebene Forschungsprojekt widmet sich der Metamodellierung. Die konkreten Ziele sind

- Erstellung einer Sammlung von repräsentativen industriellen Datensätzen zur Prüfung von Metamodellierungsalgorithmen.
- Systematische Auswertung der Metamodellierungsalgorithmen basierend auf diesen Datensätzen.
- Bestimmung der vielversprechendsten Metamodellierungstechniken für CAE Simulationsmodelle.
- Einführung einer geeigneten Methodik zur automatisierten Metamodellierung.
- Entwicklung von Richtlinien / Kriterien für eine sinnvolle Metamodellierung.

Folgender Aufbau liegt diesem Bericht zugrunde. Zunächst werden im ersten Kapitel Grundlagen der Metamodellierung, verwendete Regressionsmethoden, Fehlermaße, Modelloptimierung, Modellauswahlkriterien und Modellkomplexität behandelt. Die empirische Untersuchung der Modellauswahlkriterien ist Gegenstand des zweiten Kapitels. Das dritte Kapitel beschreibt in aggregierter Form die Ergebnisse der automatischen Modelloptimierung auf den Datensätzen, die von den Arbeitskreismitgliedern zur Verfügung gestellt wurden. Das letzte Kapitel widmet sich der Robustheitsanalyse und der entwickelten Metrik für die Robustheit in einem Arbeitspunkt. In der Zusammenfassung wird eine abschließende Bewertung bezüglich der angestrebten Ziele des Forschungsprojekts durchgeführt. Im Anhang befinden sich der Matlab-Code zur Modelloptimierung sowie die detaillierten Modellierungsergebnisse für die zur Verfügung gestellten Datensätze.

2 Metamodellierung

2.1. Einleitung

Bei der Metamodellierung soll anhand eines Datensatzes $D = (x_{1,\dots,} x_m)_{1,\dots,} (x_{1,\dots,} x_m)_n$ ein Modell M für den, in der Praxis unbekannten, Zusammenhang $y = f(x_{1,\dots,} x_m)$ gefunden werden. Der Datensatz D wird auch mit Lern- oder Trainingsdaten bezeichnet, die x sind die Eingangsvariablen und y die Ausgangsvariable. Man unterscheidet zwischen kategoriellen, geordneten und reellen/numerischen Eingangsgrößen. Ist die Ausgangsvariable reellwertig, spricht man von Regression, ansonsten von Klassifikation. Da in D die Ausgangsvariablen bekannt sind, nennt man diese Art der Metamodellierung überwachtes Lernen zur Abgrenzung vom unüberwachten Lernen, wie zum Beispiel dem Finden von Clustern. Ein trainiertes, gelerntes oder angepasstes Modell kann für jedes x ein $\hat{y}$ vorhersagen. Ziel der Metamodellierung ist es, ein Modell zu finden, bei dem ein Fehlermaß $F(y, \hat{y})$ für alle möglichen x minimal ist. Dies ist der Generalisierungsfehler, der nicht berechenbar ist, da sowohl der wahre Zusammenhang f als auch die gemeinsame Verteilung der Eingangsvariablen speziell in Anwendungsfällen nicht bekannt ist. Den Trainingsfehler, die Anwendung des Fehlermaßes auf Vorhersagen der Lerndaten, kann hingegen berechnet werden. Eine ausschließliche Minimierung des Trainingsfehler kann jedoch zu einer Überanpassung des Modells, dem sogenannten Overfitting, führen. Overfitting beschreibt die Situation, in der ein Modell zwar sehr gut die Lerndaten vorhersagen kann, für Daten mit unbekannter Ausgangsvariable, also für im Anwendungsfall interessante Daten, aber sehr schlecht prognostiziert. Zum Beispiel kann man sich ein Modell vorstellen, dass die Lerndaten „auswendig“ lernt, somit einen Trainingsfehler von 0 hat, aber für Vorhersagen auf neuen Daten beliebig schlecht werden kann. Die Vermeidung von Overfitting ist eine wichtige Motivation für die in Abschnitt 2.4 beschriebenen Modellauswahlkriterien.

2.2. Fehlermaße in der Regression

Im vorigen Abschnitt war allgemein von einem zu minimierenden Fehlermaß $F(y, \hat{y})$ die Rede. Für ein Modell M und einen Datensatz D mit wahren Ausgangswerten $y_{1,\dots,} y_n$, ihrem Mittelwert $\bar{y}$ und Varianz $var(y)$, und Vorhersagen $\hat{y}_{1,\dots,} \hat{y}_n$ durch M mit Mittelwert $\bar{\hat{y}}$ kommen bei der Regression üblicherweise die folgenden Fehlermaße zum Einsatz.

Mean Squared Error (MSE)

$$MSE := \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Root Mean Squared Error (RMSE)

$$RMSE := \sqrt{MSE}$$

Relative Root Mean Squared Error (RRMSE)

$$RRMSE := \frac{RMSE}{\sqrt{var(y)}}$$

Korrelation (Cor)

$$Cor := \frac{\frac{1}{n-1}\sum_{i=1}^{n}(y_i - \bar{y})(\hat{y}_i - \bar{\hat{y}})}{\sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(y_i - \bar{y})^2}\ \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(\hat{y}_i - \bar{\hat{y}})^2}}$$

R^2

$$R^2 := Cor^2$$

Der MSE ist Grundlage für weitere Maße, ist allerdings nicht so gut zu interpretieren, da durch das Quadrieren die Einheitentreue verloren geht. Der RMSE ist dagegen einheitentreu aber noch abhängig vom Wertebereich der Ausgangsgröße. Relative Maße sind mit dem RRMSE, der Korrelation und R^2 gegeben. Der RRMSE kann als Vergleich zwischen den Fehlern des Modells und den Fehlern eines Mittelwertschätzers[1] verstanden werden, für RRMSE=1 ist das Modell genauso gut wie ein Mittelwertschätzer, für RRMSE < 1 besser und für RRMSE > 1 schlechter. Die Korrelation, genauer der Korrelationskoeffizient, misst den linearen Zusammenhang zwischen zwei Größen, hier Vorhersagen und wahren Ausgangswerten. Cor=1 gilt bei einem vollständigen, positiven linearen Zusammenhang, Cor=-1 bei einem vollständigen, negativen linearen Zusammenhang und kein linearer Zusammenhang führt zu Cor=0. Das Maß R^2 wird auch Bestimmtheitsmaß genannt und ist für die Verwendung mit linearen Modellen entwickelt worden. Mit der obigen Definition ist R^2 auch für beliebige Modellierungsmethoden zu berechnen. Richtwerte für die Bewertung der Qualität von Modellen anhand der Korrelation oder des R^2 sind in Tabelle 1 angegeben. Sie beruhen auf Erfahrungswerten für allgemeine Regressionsprobleme. In speziellen Anwendungen, wie der in diesem Projekt betrachteten Regressionsproblemen auf Daten von Crash-Simulation, können Korrelationswerte unter 0.8 bzw. $R^2 < 0.6$ noch akzeptable Modelle darstellen.

[1] Ein Mittelwertschätzer gibt als Vorhersage den Mittelwert der Ausgangsgröße des Lerndatensatzes zurück. Sein quadratischer Trainingsfehler entspricht der Varianz der Ausgangsgröße.

Korrelation	R^2	Erwartete Modellqualität
[0.95, 1]	[0.9, 1]	Sehr gut
[0.9, 0.95)	[0.81, 0.9)	Gut
[0.8, 0.9)	[0.64, 0.81)	Akzeptabel
[0.6, 0.8)	[0.36, 0.64)	Schlecht
[0, 0.6)	[0, 0.36)	Sehr schlecht

Tabelle 1: Richtwerte zur Modellbewertung durch Cor und R^2

2.3. Regressionstechniken

Im Folgenden werden die Regressionstechniken beschrieben, die in ClearVu Analytics zum Einsatz kommen. Neben der Beschreibung der prinzipiellen Idee des Verfahrens werden die Parameter identifiziert, die bei der Modelloptimierung angepasst werden.

2.3.1. Lineare Modelle

Ein lineares Modell modelliert die Ausgangsgröße als Linearkombination von m Funktionen des Eingangsraums sowie eines Störvektors[2] ε:

$$y = \sum_{i=1}^{m} \beta_i \cdot f_i(x) + \varepsilon$$

Die Koeffizienten β_i müssen anhand der Daten angepasst werden. Ein einfaches lineares Modell für m Eingangsvariablen ist gegeben durch $y = \beta_0 + \sum_{i=1}^{m} \beta_i \cdot x_i$. Hierin wird β_0 Achsenabschnitt genannt. Es ist üblich, das einfache lineare Modell um weitere Funktionen zu erweitern. Zum einen sind dies quadratische, kubische oder höhergradige Transformationen einer Eingangsvariable, zum anderen sind dies die sogenannten Interaktionsterme, die ein Produkt aus einzelnen Eingangsvariablen bilden. Die Anpassung des Modells erfolgt durch die Minimierung des quadratischen Fehlers (least-squares-Problem), den das Modell auf den Lerndaten macht. Vorteile von linearen Modellen sind die schnelle Rechenzeit bei der Anpassung und ein einfach zu ermittelndes Komplexitätsmaß (siehe Abschnitt 2.6).

Im Gegensatz zu den anderen Regressionstechniken hat das lineare Modell keine Parameter, die zu optimieren wären. Stattdessen wird die Auswahl der Funktionsterme, insbesondere Interaktionen und quadratische Transformationen, optimiert.

[2] Für den Störvektor ε wird angenommen, dass seine Komponenten normalverteilt mit Mittelwert 0 sind.

2.3.2. Support Vector Machine

Die Support Vector Machine (SVM) wurde von Vapnik [Va1995] zur Klassifikation entwickelt. Der Ansatz zur Klassifikation versucht die Daten mit einer möglichst breiten Hyperebene zu trennen, so wie in Abbildung 1 dargestellt.

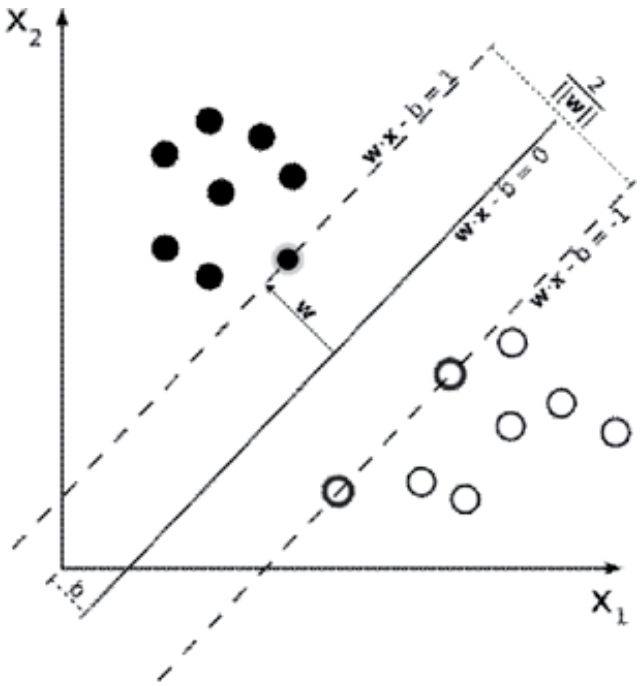

Abbildung 1: Trennende Hyperebene einer SVM

Dieser Ansatz wird als Optimierungsproblem mit Einschränkungen formuliert:

$$\min \frac{1}{2}\|w\|^2$$

$$u.d.B. \quad c_i\,(w \cdot x_i - b) \geq 1 \text{ für } 1 \leq i \leq n$$

Mit Hilfe des Lagrange-Formalismus kann dieses Optimierungsproblem in seine duale Form überführt werden, sie lautet:

$$\max \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i,} \alpha_i \alpha_j c_i c_j x_i x_j \text{ mit } \alpha_i \geq 0 \text{ und } \sum_{i=1}^{n} \alpha_i c_i = 0$$

Dieses quadratische Optimierungsproblem muss zum Trainieren einer SVM gelöst werden. Die Beobachtungen mit $\alpha > 0$ sind die Stützstellen (support vectors) des Modells und ein Weglassen der anderen Beispiele würde zum selben Modell führen.

Nicht immer ist es möglich, Daten mit einer Hyperebene perfekt zu trennen, so dass die bisherige Formulierung der SVM nicht immer eine Lösung haben muss. Um Trainingsfehler zuzulassen, wurden die sogenannten Schlupfvariablen (slack variables) ξ_i eingeführt.

$$\min \|w\|^2 + C \sum_{i=1}^{n} \xi_i$$

$$u.d.B. \quad c_i (w \cdot x_i - b) \geq 1 - \xi_i \text{ für } 1 \leq i \leq n$$

Mit C kann man steuern, wie stark Trainingsfehler bestraft werden sollen. In ihrer bisherigen Formulierung ist die SVM nur in der Lage, Daten in ihrem Eingangsraum linear durch eine Hyperebene zu trennen. Mit der Einführung von Kernfunktionen ist die SVM in der Lage auch nicht-linear zu trennen. Kernfunktionen berechnen einen Abstand in einem eventuell höherdimensionalen Raum, einem Hilbertraum. In diesem Hilbertraum ist dann wieder eine lineare Trennung mit einer Hyperebene möglich. Die Eleganz an diesem Ansatz liegt darin, dass die Daten nicht in den höherdimensionalen Raum transformiert werden müssen, sondern man nur an Abständen interessiert ist, die durch die Kernfunktion im Eingangsraum berechnet werden. Gewöhnlich stehen bei den SVM Implementierungen drei Kernfunktionen zur Verfügung. Dies sind die polynomielle Kernfunktion, $k(a,b) \coloneqq k(a \cdot b)^d$, die Radial-Basis-Funktion (RBF), $k(a,b) \coloneqq exp(-\gamma \|a-b\|^2)$ mit $\gamma > 0$, und die sigmoide Kernfunktion, $k(a,b) \coloneqq tanh(ka \cdot b+c)$.

Die Erweiterung der SVM zur Lösung von Regressionsproblemen [SS2004] wird durch eine Umformulierung des Optimierungsproblems erreicht. Statt die Daten durch eine Hyperebene zu trennen, wird versucht, die Daten in einem sogenannten ε-Schlauch unterzubringen. Analog zur Klassifikation werden auch hier Schlupfvariablen und Kernfunktionen angewendet, so dass nichtlineare Regression mit Trainingsfehlern möglich wird.

Die zu optimierenden Parameter bei der SVM sind der Kostenfaktor der Trainingsfehler C sowie die verschiedenen Parameter der Kernfunktionen.

2.3.3. Entscheidungsbäume (CART)

Entscheidungsbäume sind ursprünglich zur Klassifikation entwickelt worden und teilen Daten in Partitionen mit gleichem Ausgangswert auf. Die Partitionen sind durch Blätter in einem Baum repräsentiert. Die Knoten des Baums spiegeln einfache Entscheidungsregeln wieder, zum Beispiel den Größenvergleich zwischen einer Eingangsvariable und einer Konstante. Der Pfad von der Wurzel zu einem Blatt ist also eine Konjunktion von Regeln, die angewendet wird, um eine Vorhersage zu treffen.
Das Trainieren eines Baumes geschieht durch einen Aufbau von der Wurzel aus. Es wird eine Eingangsgröße gewählt, so dass durch ihre Aufteilung (Split) ein Informationsgewinn bzw. eine eindeutigere Aufteilung gemäß der Ausgangsgröße erzielt wird. Dies kann rekursiv weiterverfolgt werden, bis man nur noch Blätter mit gleichem Ausgangswert erhält. Dieses Vorgehen führt leicht zu Overfitting, so dass die Technik des Stutzens (Pruning) beim Trainieren eingeführt wurde. Hierbei werden Blätter vereinigt, um Trainingsfehler zuzulassen und ein Auswendiglernen der Daten zu verhindern. Breiman hat mit CART (Classification and Regression Trees) [Br1984] Entscheidungsbäume für die Regression erweitert.
Zwei Parameter, minSplit und maxDepth, werden zur Modelloptimierung verwendet. Der Parameter minSplit gibt vor, wie viele Beobachtungen mindestens in einem Knoten sein müssen, damit in ihm ein weiterer Split durchgeführt werden darf. Der Parameter maxDepth bestimmt die maximale Tiefe des Baums.

2.3.4. Neuronale Netze

Neuronale Netze sind durch die Funktionsweise des Gehirns, wo einzelne Neuronen über Synapsen miteinander verbunden sind, inspiriert worden. Der Vorläufer oder auch die erste Version eines neuronalen Netzes ist Rosenblatts Perzeptron [Ro1958]. Die Modellierung der Ausgangsgröße über eine Funktion $y = f(x)$, wobei f durch eine Komposition von einfachen linearen Funktionen g_i mit Gewichten w_i gebildet wird: $f(x) = \varphi(\Sigma_i\, w_i\, g_i\,(x))$. φ kann eine beliebige Funktion sein, im Fall der Regression ist φ die Identitätsfunktion, bei der Klassifikation kommen sigmoide Aktivierungsfunktionen zum Einsatz. Die Komposition f stellt eine Schicht von Neuronen dar. In einem neuronalen Netz können diese Schichten miteinander verbunden werden, in dem die Ausgänge einer unteren Schicht die Eingänge der oberen Schicht bilden. Das Trainieren eines neuronalen Netzes besteht im Anpassen der Gewichte w_i. Übliche Algorithmen zum Anpassen der Gewichte sind back propagation [We1975] oder die gradientenbasierte Breitensuche. In ClearVu Analytics wird die Implementierung von Ripley [Ri1996] verwendet. Ein Nachteil von neuronalen Netzen ist ihre Anfälligkeit von Overfitting, so dass beim Anpassen der Gewichte häufig ein Validierungsdatensatz verwendet wird.

Die zu optimierenden Parameter sind die Anzahl der Neuronen in einer Schicht sowie ein Wert, der zur Initialisierung der Gewichte verwendet wird.

2.3.5. Fuzzy-Modelle

Regelbasierte Modelle bestehen im Kern aus WENN-DANN-Regeln, die unter einer Bedingung (WENN) eine Handlungsanweisung (DANN) empfehlen. Der Vorteil dieser Modellart ist die einfache Nachvollziehbarkeit von Wirkungszusammenhängen eines Systems aufgrund der dem Menschen nahestehenden Wissensrepräsentation des Modells. Dadurch ist es zudem möglich, ein gegebenes Modell einfach zu erweitern und zu modifizieren, um es an veränderte oder neue Situationen anzupassen. Nachteilig ist jedoch, dass die Regeln nur aktiviert (Bedingung erfüllt) oder deaktiviert (Bedingung nicht erfüllt) sein können, woraus eine starke Diskretisierung der Ausgangsgröße folgt. Daher kann nicht erwartet werden, dass die Genauigkeit eines solchen Modells immer den Anforderungen für eine Regelungs- oder Optimierungsaufgabe genügt.

Durch die von Zadeh [Za65] eingeführte Fuzzy-Logik ist die Erweiterung der regelbasierten Modelle zu Fuzzy-Modellen möglich geworden. Im Gegensatz zur klassischen booleschen Logik, die nur die Wahrheitswerte wahr oder falsch kennt, lässt die Fuzzy-Logik auch Wahrheitswerte zwischen wahr und falsch zu. Die daraus resultierenden Fuzzy-Zugehörigkeitsfunktionen werden in den Fuzzy-Modellen für die Formulierung der Regeln genutzt. Durch die Verwendung von Zugehörigkeitsfunktionen kann die Modellierungsgüte gegenüber den regelbasierten Modellen deutlich verbessert werden. Daher verknüpfen Fuzzy-Modelle eindrucksvoll die Interpretierbarkeit eines Modells mit einer hohen Abbildungsgenauigkeit des Systems auf ein Modell.

Das Verfahren zur Generierung der Fuzzy-Regeln basiert auf der von Kiendl und Krabs vorgeschlagene Regel-Orientierte Statistische Analyse (ROSA) [KK89] zur automatischen Generierung regelbasierter Systeme. Die Grundidee besteht darin, für die Aufnahme einer potenziellen Regel in einen Regelsatz mithilfe eines statistischen Tests zu überprüfen, ob die Regel durch die Beobachtungsdaten belegt werden kann. Der dafür entwickelte Relevanzindex als Regeltest- und Bewertungsstrategie vereinigt ein aus der Literatur bekanntes Schema zur Bewertung der Unsicherheit von Regeln in Expertensystemen mit dem statistischen Konzept der Konfidenzintervalle [Kr94]. Im Gegensatz zu anderen Verfahren, die den gesamten Regelsatz evaluieren und optimieren (Pittsburgh-Ansatz), wird der Regelsatz im ROSA-Verfahren sukzessiv aus individuell getesteten und bewerteten Regeln aufgebaut (Michigan-Ansatz). Damit wird zum einen der Suchraum verkleinert, da nur Regeln und nicht Mengen von Regeln betrachtet werden müssen. Zum anderen wird durch den Test gewährleistet, dass die Regeln lokal vernünftig sind. Mit dem Ziel, diese Vorteile auch für die datenbasierte Fuzzy-Modellierung zu nutzen, ist 1994 die Vorgehensweise des ROSA--Verfahrens von Kiendl und Krone auf den Fall von Fuzzy-Regeln übertragen worden [KK95, Kr99].

Für die Modelloptimierung werden verschiedene Regelbewertungsverfahren mit unterschiedlichen Einstellungen verwendet, um einen optimalen Satz an Regeln zu erhalten.

2.4. Modelloptimierung

Alle Regressionstechniken, die im letzten Abschnitt vorgestellt wurden, sind parametrisierbar, entweder durch numerische Parameter, z.B. den Kostenfaktor C der SVM, oder durch strukturelle Parameter, z.B. die Art und Auswahl der Interaktionsterme bei linearen Modellen. Diese Parameter müssen für einen gegebenen Datensatz angepasst werden. Dies ist eine mühsame Aufgabe, wenn sie manuell durchgeführt werden muss. Modelloptimierung automatisiert diesen Prozess und findet die besten Einstellungen für die Parameter zu einem gegebenen Datensatz. Zur Modelloptimierung kann im Allgemeinen jedes Blackbox Optimierungsverfahren[3] eingesetzt werden, z.B. verwendet ClearVu Analytics Evolutionsstrategien [Bä1996] [BFK2013] zur Modelloptimierung. In Abschnitt 2.1 wurde beschrieben, dass das ausschließliche Minimieren des Trainingsfehlers zu Overfitting führen kann und somit alleine kein sinnvolles Auswahlkriterium für eine Modelloptimierung ist, um ein gut generalisierendes Modell zu erhalten. Im nächsten Abschnitt werden vier Modellauswahlkriterien vorgestellt, die als Auswahlkriterien für eine Modelloptimierung verwendet werden können.

2.5. Modellauswahlkriterien

Modellauswahlkriterien bewerten angepasste Modelle mit der Intention, dass ein besser bewertetes Modell einen besseren (d.h., kleineren) Generalisierungsfehler hat. Hierbei kann man zwei Ansätze unterscheiden. Zum einen gibt es Methoden, die aus der parametrischen Statistik und der Welt der linearen Modelle stammen, zum anderen die Kreuzvalidierung (cross-validation, CV), die einen nichtparametrischen Ansatz über zurückgehaltene Validierungsdatensätze verfolgt.

CV ist auf jede Modellierungsmethode anwendbar, da keine Annahmen gemacht werden bzw. Kenntnisse über die internen Abläufe der Modellierungsmethode vorhanden sein müssen. Bei der k-fachen Kreuzvalidierung wird der Lerndatensatz mit n Beobachtungen zufällig in k, mit $1 < k <= n$, möglichst gleichgroße Partitionen aufgeteilt. Mit denselben Parametern, die zur Erstellung des Modells auf dem gesamten Lerndatensatz verwendet werden, wird auf der Vereinigung von k-1 Partitionen ein Modell gelernt, das für die verbleibende Partition die Ausgangsgröße vorhersagt. Aus den Vorhersagen und den wahren Werten können so Fehlermaße berechnet werden, die auf ungesehenen Daten beruhen und dadurch besser geeignet sind, auf den Generalisierungsfehler des Modells zu schließen als der Trainingsfehler. Dieser Prozess wird k mal ausgeführt, so dass jede Partition und somit jede Beobachtung einmal in einer Validierungsmenge verwendet wird. Die k Fehlermaße werden gemittelt, um eine Größe zur Bewertung des Modells zu erhalten. Die Wahl von k führt zu gegenläufigen Konsequenzen, wie man bei der Betrachtung der Extremfälle k=2 und k=n gut beobachten kann. Für k=2 verzichtet man bei der Modellierung der beiden Partitionen auf die Hälfte der Daten, was die Modellgüte bei kleinen Datenmengen stark verschlechtern kann.

[3] Ein Blackbox Optimierungsverfahren benötigt keine Kenntnisse über die Struktur des zu optimierenden Prob-lems, sondern verwendet Funktionsauswertungen, die eine „Blackbox“ anhand von Lösungsvorschlägen des Optimierers ausgibt.

Der Vorteil dieses Extremfalls ist die geringe Rechenzeit, da nur zwei Modellierungen durchgeführt werden müssen. Für k=n, hier spricht man von leave-one-out, ist die Anzahl der Beobachtungen fast identisch zum ursprünglichen Lerndatensatz. Allerdings benötigen n Modellierungen sehr viel Rechenzeit. In der Praxis wird daher k=10 als guter Kompromiss zwischen Datenlage und Rechenzeit angesehen. Bei einer umfangreichen empirischen Untersuchung bezüglich der Schätzung des Generalisierungsfehlers [Ko1995] war die Wahl von k=10 sogar der leave-one-out Kreuzvalidierung überlegen.

Die nun folgenden Modellauswahlkriterien sind zur Bewertung von linearen Mo-dellen entwickelt worden. Im Gegensatz zur k-fachen Kreuzvalidierung werden hier Maße, die sich aus dem angepassten Modell ergeben, miteinander zu einer Größe verrechnet. Dabei handelt es sich um den Trainingsfehler, in der Form der Summe der quadratischen Fehler (RSS), und den Freiheitsgraden der Methode, im Fall der linearen Modelle die Zahl der anzupassenden Terme. Die Freiheitsgrade (degrees of freedom, DF) sind ein Komplexitätsmaß für das Modell und Abbildung 2 zeigt beispielhaft eine Pareto-Front zwischen Trainingsfehler und Modellkomplexität sowie die Auswahl einer Lösung anhand der im Folgenden definierten Modellauswahlkriterien.

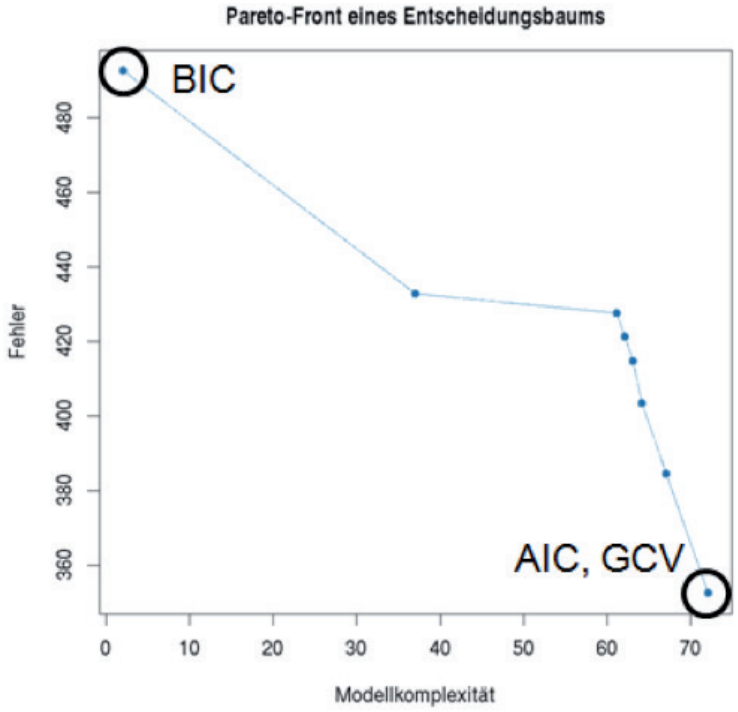

Abbildung 2: Tradeoff zwischen Fehler und Komplexität

Das Akaike Information Criterion (AIC) [Ak1974] ist definiert als

$$AIC := 2 \cdot DF + n \cdot \log(RSS/n) .$$

Das Bayesian Information Criterion (BIC) [Sc1978] ist definiert als

$$BIC := n \cdot \log(RSS/n) + DF \cdot \log(n) .$$

Die Generalized Cross Validation (GCV) [WC1978] ist definiert als

$$GCV := (RSS/n)/(n - DF)^2 .$$

Um diese drei Modellauswahlkriterien mit anderen Modellierungsmethoden nut-zen zu können, muss die Zahl der Freiheitsgrade bekannt sein bzw. ein anderes Komplexitätsmaß angewendet werden, das für beliebige Modellierungsmethoden berechenbar ist.

2.6. Modellkomplexität

Man spricht von einem komplexen Modell, wenn es in der Lage ist, einen kom-plexen Zusammenhang abbilden zu können. So ist zum Beispiel ein einfaches[4] lineares Modell nicht in der Lage multimodale Funktionen abzubilden. Eine Modellierung des wahren Zusammenhang y=sin(x) würde die Gerade y=0 ergeben. Ein neuronales Netz hingegen wäre in der Lage multimodale Funktionen zu modellieren. So kann die qualitative Aussage getroffen werden, dass neuronale Netze komplexer als (einfache) lineare Modelle sind. Aber auch innerhalb einer Modellierungsmethode können durch die Parametrisierung der Methode unterschiedlich komplexe Modelle erzeugt werden. Diese Eigenschaft wird bei der Modelloptimierung mit den parametrischen Modellauswahlkriterien ausgenutzt, um ein Modell zu finden, das einen guten Kompromiss aus den gegenläufigen Zielen Modellkomplexität und Vorhersagefehler darstellt. Man ist also an einem Maß interessiert, dass es einem ermöglicht, die Modellkomplexität zu quantifizieren.

Die Komplexität von linearen Modellen wird mit der Anzahl ihrer Freiheitsgrade gemessen. Dieses Konzept, das durch die Theorie der linearen Modelle motiviert ist, kann nicht ohne Weiteres auf andere Modellierungsmethoden angewendet werden. In [IM2007] wurde das Maß äquivalente Freiheitsgrade für neuronale Netze entwickelt und erfolgreich mit dem Modellauswahlkriterium AIC angewendet.

Ein weiteres Maß für die Modellkomplexität ist die von Vapnik und Chervonenkis entwickelte VC-Dimension [VC1971]. Sie gilt für Klassifikationsprobleme und misst, wie viele mögliche Aufteilungen der Ausgangsgröße durch eine Methode voneinander getrennt werden können. Anhand der VC-Dimension und des Trainingsfehler kann sogar eine obere Schranke für den Generalisierungsfehler bewiesen werden [Va1995]. Sie ist allerdings so hoch, dass sie für die praktische Anwendung keine Relevanz hat. Ein weiteres Problem mit der VC-Dimension ist, dass sie für Methoden, die Daten „auswendig“ lernen können, wie zum Beispiel Entscheidungsbäume, den Wert unendlich annimmt und so nicht für weitere Berechnungen geeignet ist.

[4] Mit einfach ist gemeint, dass keine Interaktionsterme oder quadratischen Terme verwendet werden, sondern das Modell durch eine Linearkombination der ursprünglichen Eingangsgrößen sowie eines Achsenabschnitts gegeben ist.

Ein guter Kandidat als Maß für die Modellkomplexität sind die Generalized Degrees of Freedom (GDoF) [Ye1998]. GDoF ist definiert als:

$$GDoF(M) = \sum_{i=1}^{n} \frac{\partial E[\hat{y}_i]}{\partial y_i}$$

Hierin ist M das Modell, das auf einem Datensatz mit n Beobachtungen trainiert wird, die y_i sind die wahren Ausgangwerte und die $\hat{y}_i$ die Vorhersagen des Modells. Die einzelnen Summanden geben an, wie stark sich die Vorhersage einer Beobachtung ändert, wenn sich der wahre Ausgangswert der Beobachtung ändert. Die partiellen Ableitungen analytisch zu berechnen können, setzt voraus, dass die Vorhersagen einer Modellierungsmethode abhängig von den wahren Ausgangswerten in differenzierbarer Form vorliegt. Da dies bei den meisten Modellierungsmethoden nicht der Fall ist, wird in [Ye1998] ein Monte-Carlo-Ansatz verfolgt, um die partiellen Ableitungen numerisch zu berechnen. Hierbei werden aus einer Umgebungsverteilung für jede der n Beobachtungen Änderungen der Ausgangsgröße gezogen, auf dem so geänderten Datensatz neu modelliert und mit dem neuen Modell vorhergesagt. Dies wird mit T Wiederholungen ausgeführt. Man erhält so für jede Beobachtung T geänderte Vorhersagen mit den zugrundeliegenden Ausgangsänderungen. Daraus wird mit einer linearen Regression eine Steigung bestimmt, die als Schätzer für die partielle Ableitung verwendet wird. Eine interessante Eigenschaft von GDoF ist, dass GDoF und Freiheitsgrade bei linearen Modellen identisch sind. Auf diese Weise wurden in [Ye1998] die Modellauswahlkriterien AIC und GCV erfolgreich auf Entscheidungsbäume angewendet. Der Monte-Carlo-Ansatz ist allerdings sehr rechenintensiv, da für die Bestimmung von GDoF eines Modells T·n neue Modelle trainiert werden müssen. Im Rahmen dieser Arbeit wird daher ein vereinfachter Ansatz verwendet, der auf T Wiederholungen verzichtet und die partiellen Ableitungen numerisch mit n Neumodellierungen schätzt. Anstatt die Pertubation h des Ausgangs y aus einer Umgebungsverteilung zu ziehen, wird sie deterministisch als $h = \max(\sqrt{\varepsilon} \cdot y, \varepsilon^2)$ berechnet[5] , für unsere Experimente haben wir ε=0,0001 verwendet. Diese Vereinfachung scheint gerechtfertigt, wie die empirische Untersuchung zur Plausibilität von GDoF in Abschnitt 3.1 zeigt. Eine weiterführende Vereinfachung, anstatt n Beobachtungen nur ein Sample zur verwenden, wurde verworfen, da die Schätzungen für GDoF zu sehr mit der Wahl des Samples schwankten. Mit GDoF verfügt man über ein Maß für die Modellkomplexität, dass für beliebige Modellierungsmethoden berechnet werden kann. Für die Untersuchungen im Rahmen dieser Arbeit spielt es keine große Rolle, dass n Modellierungen nötig sind, um GDoF zu berechnen. In der Praxis kann dies relevant sein, insbesondere, wenn man die parametrischen Auswahlkriterien hauptsächlich verwenden möchte, um Rechenzeit im Vergleich zu einer zehnfachen Kreuzvalidierung zu sparen. Hierbei würde die Rechenzeit sogar steigen, da die Berechnung genauso rechenintensiv ist wie eine leave-one-out Kreuzvalidierung.

[5] Die Maximumsbildung sorgt dafür, dass h bei kleinen Werten für y nicht 0 wird.

3 Empirische Untersuchung der Modellauswahl-kriterien

Wie im vorigen Kapitel beschrieben, ist bei der Metamodellierung in den meisten Anwendungsfällen die Minimierung des Generalisierungsfehlers das vorrangige Ziel. Inwiefern die verschiedenen Kriterien zur Modellauswahl zu einem gut generalisierenden Modell führen, ist Gegenstand einer empirischen Untersuchung. Bevor der Versuchsaufbau und die Ergebnisse dieser Untersuchung beschrieben werden, wird zunächst die Plausibilität von GDoF, das identifizierte Komplexi-tätsmaß für beliebige Modellierungsmethoden und somit Grundlage für die An-wendung der Modellauswahlkriterien, überprüft.

3.1. Plausibilität von GDoF

GDoF wurde in [Ye1998] erfolgreich auf Regressionsbäume (CART) angewendet. Bevor die eigene Implementierung von GDoF als Komplexitätsmaß verwendet wird, soll untersucht werden, ob GDoF sich den Erwartungen entsprechend verhält. Für zwei Modellierungsmethoden, Entscheidungsbäume und Support Vector Machine, wird abhängig von zu variierenden Modellparametern GDoF bestimmt. Für diese Untersuchung wird ein 500 Beobachtungen umfassendes Sample des Covertype Datensatzes aus dem UCI Machine Learning Repository (MLR) [BL2013] verwendet. Bei Entscheidungsbäumen werden die Parameter maxDepth und minSplit verändert. Je tiefer ein Entscheidungsbaum werden darf, gesteuert durch maxDepth, umso komplexer sollte er sein, da ein tieferer Baum den Datensatz in kleinere Partitionen einteilen kann. Bei minSplit wird ein gegenteiliges Verhalten erwartet. Je größer die minimale Anzahl von Beobachtungen in einem Knoten sein muss, um einen weiteren Split durchzuführen, desto weniger komplex sollte das Modell werden. Abbildung 3 zeigt den Zusammenhang zwischen maxDepth und GDoF.

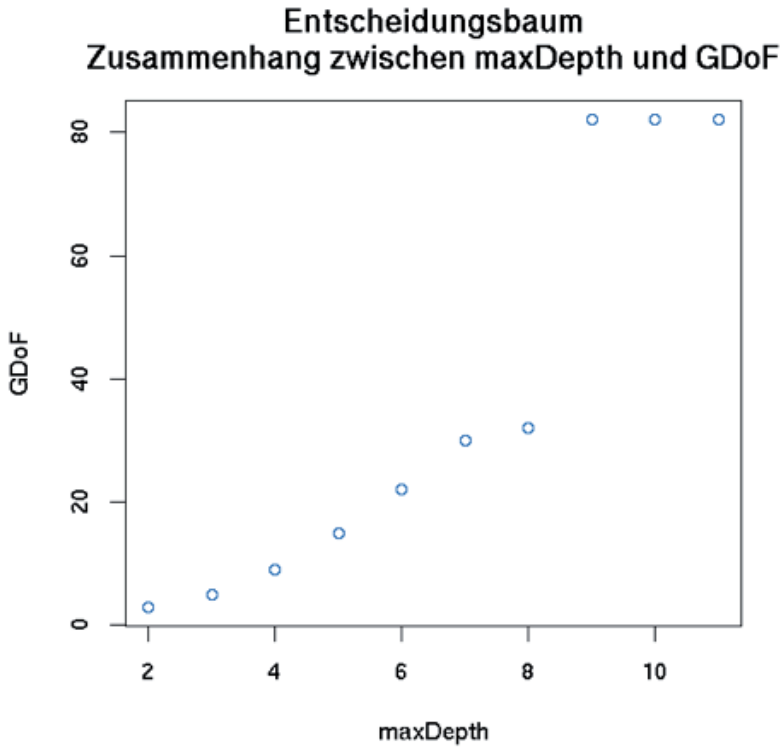

Abbildung 3: Zusammenhang zwischen maxDepth und GDoF

Der Verlauf entspricht den Erwartungen. Ab maxDepth > 8 stagniert die Komplexität in einem Plateau. Dies ist die direkte Konsequenz aus der Wahl eines 500 Beobachtungen umfassenden Samples. Ab einer Tiefe von 9 kann ein Entscheidungsbaum, binäre Splits vorausgesetzt, 2^9 = 512 > 500 Blätter haben. Er ist somit ab einer Tiefe 9 in der Lage, jede Beobachtung in einem Blatt unterzubringen , was einer Aufteilung mit maximaler Komplexität entspricht. Abbildung 4 zeigt den Zusammenhang zwischen minSplit und GDoF.

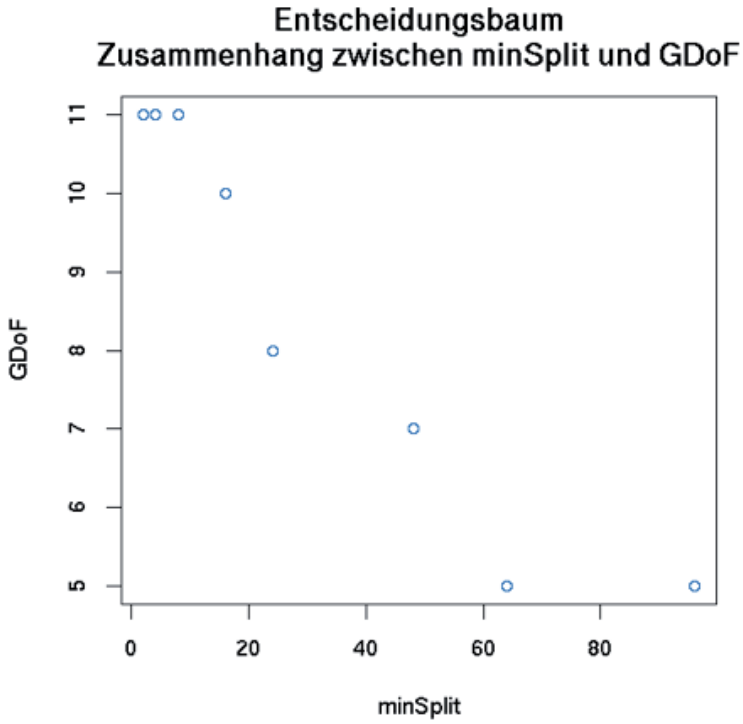

Abbildung 4: Zusammenhang zwischen minSplit und GDoF

Auch im Fall von minSplit ist dies der erwartete Verlauf. Das Plateau lässt sich analog zum Plateau bei maxDepth begründen. Bei der Variation von minSplit ist maxDepth = 4. D.h. unabhängig wie klein minSplit wird, eine Komplexität von 11 kann bei einer maximalen Baumtiefe von 4 nicht überschritten werden. Für die Support Vector Machine wurde der Kostenfaktor C variiert. Er ist ein Straffaktor für die zugelassenen Trainingsfehler und es wird erwartet, dass die Komplexität steigt, wenn C erhöht wird. In Abbildung 5 ist der erwartete Verlauf dargestellt. Da der Bereich, in dem C geändert wird, sehr groß ist, wird eine logarithmische Darstellung bezüglich C gewählt, was einen linearen Zusammenhang zwischen log(C) und GDoF erkennen lässt. Wie bei den Entscheidungsbäumen kommt es ab einem bestimmten Wert von C zu einer Plateaubildung bezüglich der Komplexität. Auch hier ist die Erklärung, dass ab einem bestimmten Wert von C, die Daten wahrscheinlich „auswendig“ gelernt werden, was dem höchsten Maß an Komplexität entspricht und eine weitere Erhöhung von C keine Auswirkungen auf die Komplexität haben kann.

In den drei hier dargestellten Fällen verhält sich GDoF wie erwartet und wird daher als Komplexitätsmaß für die zu untersuchenden Modellauswahlkriterien verwendet.

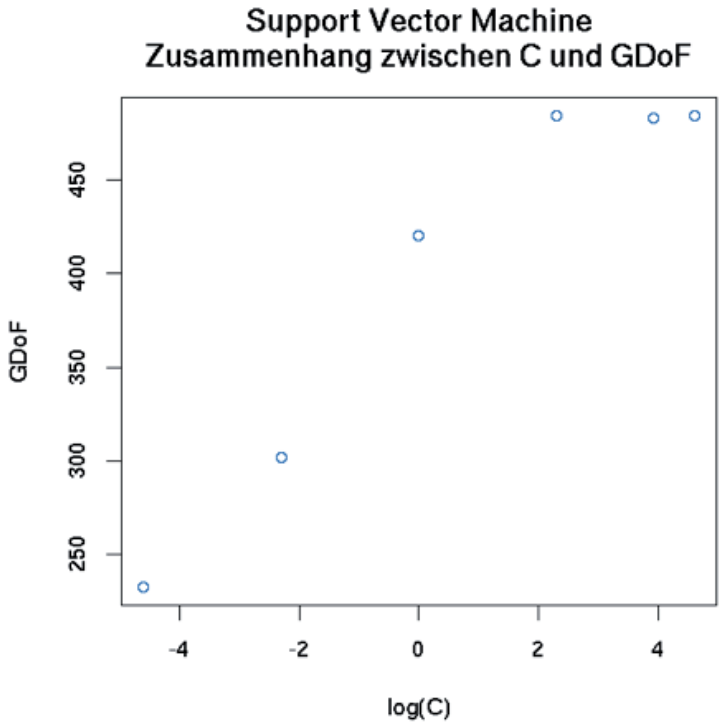

Abbildung 5: Zusammenhang zwischen log(C) und GDoF

3.2. Versuchsaufbau

Bei der Modelloptimierung können verschiedene Auswahlkriterien verwendet werden. Im vorigen Kapitel wurden die Modellauswahlkriterien CV, AIC, BIC und GCV vorgestellt. In einer empirischen Untersuchung soll festgestellt werden, welches dieser vier Kriterien zu gut generalisierenden Modellen führt. Wie schon im letzten Kapitel beschrieben, ist die Berechnung des Generalisierungsfehlers mit Ausnahme konstruierter Beispiele nicht möglich. Um den Generalisierungsfehler sinnvoll zu schätzen ist eine sehr große Datenmenge nötig. In dem benötigten Umfang sind weder die durch die Arbeitskreismitglieder zur Verfügung gestellten Datensätze noch solche aus dem MLR vorhanden. Daher werden für die Generierung von Datensätzen Testfunktionen aus dem Bereich der Blackbox Optimierung verwendet. Im Einzelnen sind dies die Funktionen Katsuura, Rastrigin und Rosenbrock, wie sie bei den Blackbox Optimization Benchmarks (BBOB) [HAFR2010] zum Einsatz kommen. Sie sind nichtlinear und besitzen mit der Ausnahme von Rosenbrock sehr viele lokale Optima und sind daher schwierig zu modellieren. Dies ist für die hier durchgeführte Untersuchung von Vorteil, da man erwarten kann, dass sich bei einem schwierig zu modellierenden Datensatz im Laufe einer Modelloptimierung eine umfangreiche Pareto-Front zwischen Komplexität und Trainingsfehler finden lässt, aus der die Auswahlkriterien Modelle auswählen können. Die Testfunktionen werden mit zwei Eingangsvariablen aus dem Designraum [-5, 5] x [-5, 5] verwendet.

Der Validierungsdatensatz zur Schätzung des Generalisierungsfehlers ist durch Auswertungen der Testfunktion auf einem Rastergitter mit Abstand 0.01 im Designraum gegeben, womit der Validierungsdatensatz aus knapp über 1 Million Beobachtungen besteht. Aus diesem Datensatz wird ein Lerndatensatz mit 100 Beobachtungen zufällig gezogen. Mit diesem Lerndatensatz werden für jedes Modellauswahlkriterium fünf Modelloptimierungsläufe durchgeführt. Die Modelloptimierungsläufe mehrfach durchzuführen ist dadurch motiviert, dass ClearVu Analytics eine Evolutionsstrategie zur Modelloptimierung verwendet und damit nicht deterministisch ist. Die erstellten Modelle werden auf den Validierungsdatensatz angewendet und der resultierende Vorhersagefehler wird als Schätzer für den Generalisierungsfehler des Modells benutzt, ein Beispiel der Ergebnisse von 5 Optimierungsläufen ist in Abbildung 6 dargestellt.

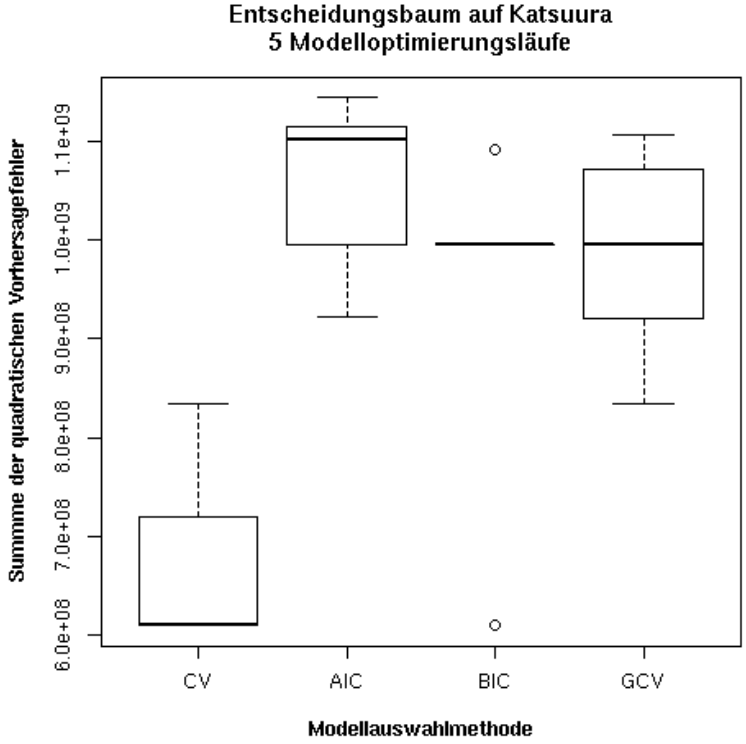

Abbildung 6: Bsp. für die Generalisierungsfehler aus 5 Modelloptimierungsläufen

Für die Auswertung werden die Generalisierungsfehler der einzelnen Modellauswahlkriterien paarweise mit einem nichtparametrischen Test, dem Student-t Test, auf signifikante Unterschiede überprüft. Im Falle eines signifikanten Unterschieds gewinnt das Modellauswahlkriterium mit dem kleineren Generalisierungsfehler, ansonsten wird der Vergleich als Unentschieden gewertet. Dieser Ansatz wird mit 30 verschiedenen Lerndatensätzen für die Modellierungsmethoden Lineare Modelle, Support Vector Machine, Entscheidungsbäume und Neuronale Netze durchgeführt. Die Auswahl dieser Methoden ist motiviert durch ihr gutes Abschneiden bei der Modellierung der gelieferten Datensätze (siehe Kapitel 5).

3.3. Ergebnisse

Zur Auswertung der in Abschnitt 3.2 beschriebenen Versuche werden die Siege, die ein Modellauswahlkriterium gegen ein anderes erzielte, für jede der drei Testfunktionen und jede der vier Modellierungsmethoden gezählt. Bei 30 Vergleichsläufen kann ein Modellauswahlkriterium so maximal 90 Siege erreichen. Die gezählten Siege sind in Tabelle 2 dargestellt.

Funktion	Modellierungsmethode	CV	AIC	BIC	GCV
Katsuura	Lineares Modell	90	0	0	0
	Support Vector Machine	1	2	3	3
	Neuronale Netze	2	0	0	0
	Entscheidungsbaum	63	4	20	5
Rastrigin	Lineares Modell	33	24	17	24
	Support Vector Machine	85	1	0	0
	Neuronale Netze	3	0	0	3
	Entscheidungsbaum	14	11	10	11
Rosenbrock	Lineares Modell	79	13	3	13
	Support Vector Machine	88	2	3	1
	Neuronale Netze	4	2	1	2
	Entscheidungsbaum	26	17	11	16

Tabelle 2: Ergebnisse des Vergleichs der Modellauswahlkriterien

Die Fälle, in denen nur wenige Siege gezählt wurden, also bei den neuronalen Netzen mit allen Testfunktionen und der SVM mit Testfunktion Katsuura, lassen darauf schließen, dass dort keine signifikanten Unterschiede zu beobachten waren. In den übrigen 8 Szenarien stellt sich die 10-fache Kreuzvalidierung als das beste Modellauswahlkriterium heraus. In 5 dieser 8 Szenarien schneidet CV sogar deutlich besser ab.

4 Robustheitsanalyse

4.1. Robustheit in der Metamodellierung

Die moderne Entscheidungstheorie brachte das Konzept der robusten Optimie-rung in der Formulierung von Walds maximin-Modell hervor [Wa1945], das bis heute in vielen Fachrichtungen angewendet wird. Bei der robusten Optimierung geht es darum ein möglich stabiles Optimum zu erreichen. Stabil bedeutet hier, das Optimum gegen Schwankungen der Eingangsparameter zu schützen. Man versucht eine Umgebung im Eingangsraum zu maximieren, die einen vom Eingangsraum abhängigen Wert optimiert.

In der Metamodellierung wird Robustheit vor allem bei der Entwicklung bzw. Analyse von Metamodellierungsverfahren verwendet. So wird in [GL1997] eine robuste Lösung für Least-Squares Probleme entwickelt, die bei vielen Metamodellierungsverfahren zum Einsatz kommen. Mit [LGBJ2002] wird eine Support Vector Machine zur Klassifikation eingeführt, die die Idee der robusten Optimierung umsetzt.

Diese Arbeiten zur Robustheit haben zum Ziel, ein Metamodellierungsverfahren robust zu machen bzw. seine Robustheit nachzuweisen. Die im nächsten Ab-schnitt beschriebene Metrik zur Robustheit wird hingegen auf einen Arbeitspunkt und ein angepasstes Metamodell angewendet, anstatt das Verfahren selbst zu bewerten oder zu verbessern.

4.2. Metrik zur Robustheit

Die im vorigen Abschnitt zitierten Arbeiten weisen für einige Metamodellierungsverfahren ihre Robustheit nach. Bei der Anwendung von Metamodellen, insbesondere bei Verwendung von Metamodellen in der Optimierung, ist die qualitative Aussage, ein robustes Verfahren verwendet zu haben, nicht genug. Hier ist es von Interesse, quantitative Aussagen zur Robustheit eines bestimmten Arbeitspunkts P machen zu können. Da die Metrik für generische Metamodellierungsmethoden berechenbar sein soll, kommen Ansätze, die die internen Abläufe einer Metamodellierungsmethode beim Lernen ausnutzen, nicht in Frage. Es wird ein Ansatz gesucht, der alleine mit den Vorhersagen des angepassten Metamodells eine Metrik berechnen kann.

Eine mögliche Metrik kann mit Hilfe einer ε-Umgebung im Eingangsraum definiert werden. In [SNW2012] werden verschiedene abgeschlossene Mengen für eine ε-Umgebung vorgestellt. Eine naheliegende Wahl für eine ε-Umgebung ist eine Hyperkugel mit einem von ε abhängigen Radius und Mittelpunkt P. Wegen der möglichen unterschiedlichen Wertebereiche im Eingangsraum, muss der Eingangsraum normiert werden, um eine Hyperkugel anwenden zu können. Im ursprünglichen Eingangsraum wird die ε-Umgebung so zu einer Hyperellipse. Die Metrik beruht auf der Idee, die Varianz der Vorhersagen des Metamodells für die ε-Umgebung zu untersuchen. Beobachtet man keine Varianz, ist der Arbeitspunkt bezüglich des angepassten Metamodells robust in der ε-Umgebung.

Je höher die Varianz ist, umso weniger robust ist P bezogen auf die ε-Umgebung. In der Praxis ist es allerdings nicht möglich, Vorhersagen für die gesamte ε-Umgebung zu berechnen, da nicht für alle Metamodellierungsverfahren analytische, integrierbare Ausdrücke für ihre Vorhersagen formuliert werden können. Dieses Problem kann mit einem Monte-Carlo Ansatz gelöst werden, in dem n Punkte innerhalb der ε-Umgebung vorhergesagt werden. Die Varianz der Vorhersagen ist abhängig von ihrem Wertebereich. Um vergleichbare Werte zu erhalten, werden die Vorhersagen in der ε-Umgebung auf das Intervall [0, 1] normiert. Sei v die so berechnete Varianz der Vorhersagen des Modells M innerhalb der ε-Umgebung um den Arbeitspunkt P, dann ist $R(M,P,\varepsilon,n) := 1 - 2 \cdot v$ ein Maß für die Robustheit des Arbeitspunkts P für den Radius ε bezüglich M. Für $R(M,P,\varepsilon,n) = 1$ ist die Varianz der Vorhersagen in der ε-Umgebung minimal bzw. die Robustheit maximal und $R(M,P,\varepsilon,n) = 0$ wird bei maximaler Varianz[6] der Vorhersagen bzw. minimaler Robustheit erreicht. Der Algorithmus zur Berechnung von $R(M,P,\varepsilon,n)$ ist durch folgenden Pseudocode beschreiben:

Eingabe:
Modell M, Arbeitspunkt P, Radius ε aus (0, 1], Monte-Carlo Samplegröße n
Ausgabe:
Robustheit R

1) Zufällige Ziehung von n Punkten aus der Hyperkugel um P mit Radius ε
2) Transformation der n Punkte anhand der Wertebereiche des Eingangsraums (Hyperellipse)
3) Vorhersage mit Modell M auf den transformierten Punkten
4) Normieren der Vorhersagen auf [0, 1]
5) Ausgabe von $R = 1 - 2 \cdot$ Varianz(normierte Vorhersagen)

Algorithmus 1: Berechnung der Robustheit

Der Radius ε ist durch die Anwendung bestimmt und sollte durch den Benutzer eingestellt werden. Er ist eine relative Größe, d.h. ein Radius ε=1 würde eine Hyperellipse um den gesamten Eingangsraum aufspannen. Ob $R(M,P,\varepsilon,n)$ als Schätzer für die wahre Robustheit des Arbeitspunkts P verwendet werden kann, hängt von der Güte des Modell M ab.

[6] Die maximal mögliche Varianz von Werten aus dem Intervall [0, 1] ist 0.5.

5 Anwendung auf Datensätze der Arbeitskreis-mitglieder

5.1. Datenbeschreibung

Sieben Arbeitskreismitglieder stellten Datensätze zur Verfügung, eine Übersicht ist durch Tabelle 3 gegeben.

Firma	Anzahl der Datensätze	Anzahl der Ausgangsvariablen
Behr	2	13
Bosch	1	67
Ford	21	645
Opel	9	55
Porsche	6	55
TRW	1	19
ZF	1	7

Tabelle 3: Übersicht der gelieferten Datensätze

Bei allen 861 Ausgangsgrößen sind reellwertige Größen und somit handelt es sich ausschließlich um Regressionsaufgaben. Der Zustand aller Daten ist direkt zur Modellierung geeignet, es gab keine fehlenden Werte. Auch sonstige Techniken wie Ausreißererkennung mussten nicht angewendet werden. Tabelle 4 gibt für jeden Datensatz die Anzahl der Beobachtungen, Ein- und Ausgangsgrößen wieder.

Datensatz	Beobachtungen	Eingangsvariablen	Ausgangsvariablen
Behr I	144	5	2
Behr II	36	7	11
Bosch I	459	45	67
Ford I	213	104	12
Ford II	245	52	10
Ford III	292	59	12
Ford IV	210	62	27
Ford V	223	62	60
Ford VI	151	34	2
Ford VII	301	63	1
Ford VIII	151	26	13
Ford IX	301	63	1
Ford X	301	63	195
Ford XI	300	63	183
Ford XII	470	77	20
Ford XIII	494	77	23
Ford XIV	472	109	10
Ford XV	477	109	10
Ford XVI	392	109	9
Ford XVII	498	104	10
Ford XVIII	383	109	12
Ford XIX	489	104	10
Ford XX	393	84	13
Ford XXI	374	84	12
Opel I	103	52	7
Opel II	90	45	8
Opel III	92	46	3
Opel IV	60	31	2
Opel V	126	64	3
Opel VI	100	50	5
Opel VII	708	142	23
Opel VIII	603	142	1
Opel IX	710	142	3
Porsche I	140	27	6
Porsche II	128	17	7
Porsche III	189	27	14
Porsche IV	169	17	14
Porsche V	169	27	6
Porsche VI	169	27	8
TRW I	65	8	19
ZF I	200	14	7

Tabelle 4: Eigenschaften der einzelnen Datensätze

5.2. Aggregierte Modellierungsergebnisse

Bei insgesamt 861 zu modellierenden Ausgangsgrößen können die Ergebnisse in diesem Dokument nur in aggregierter Form wiedergegeben werden. Ein zusätzliches Dokument mit den Fehlermaßen und Scatterplots für alle Ausgangsgrößen ist in Anhang A.2 referenziert. Zunächst ein Überblick über die Güte der Modelle bezüglich aller Ausgangsgrößen, bei 455 von 861 Ausgangsgrößen erzielten die Modelle in der Validierung eine Korrelation zwischen Vorhersage und wahrer Ausgangsgröße von mehr als 0.95, 543 von 861 mehr als 0.9 und 659 von 861 immerhin noch eine Korrelation von 0.8. Abbildung 7 zeigt die Verteilung in Form eines Histogramms.

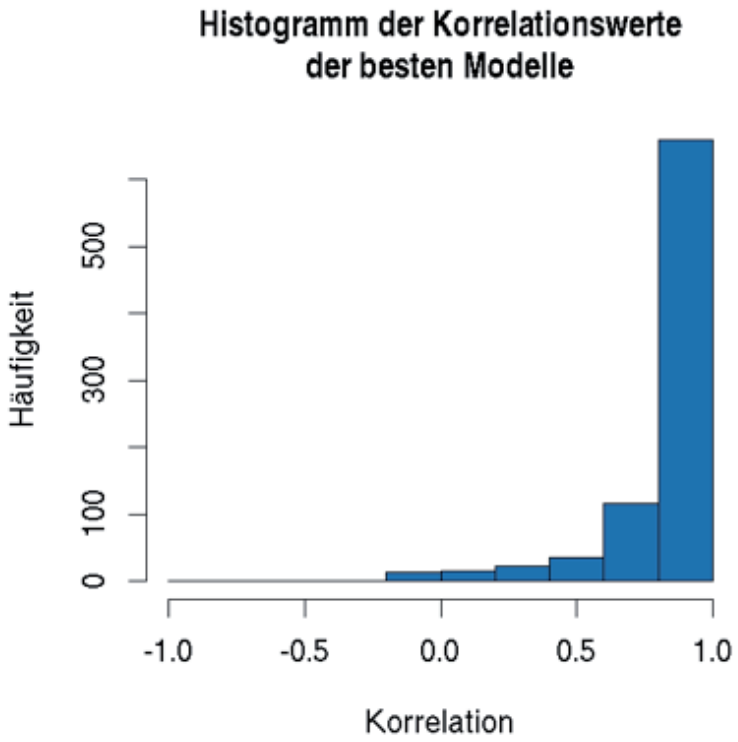

Abbildung 7: Korrelationswerte aus der Validierung für alle Ausgangsgrößen

Abbildung 8 stellt dar, wie häufig eine Modellierungsmethode bestes Modell für eine Ausgangsgröße ist. Für die meisten Ausgangsgrößen ist das lineare Modell die beste Modellierungsmethode, bei manchen Ausgangsgrößen sind die SVM, Entscheidungsbäume und selten die Kernel Quantile Regression, Fuzzy Modelle, Neuronale Netze und Gaussian Processes besser als die anderen.

Abbildung 9 zeigt zwei Histogramme, in denen die besten Modellierungsmethoden gezählt werden, die Ausgangsgrößen mit Korrelationswerten über 0.95 bzw. 0.8 vorhersagen konnten. Trotz dieser Einschränkungen bleibt das lineare Modell die beste Modellierungsmethode, wiederum gefolgt von SVM und Entscheidungsbäumen. Die übrigen Modellierungsmethoden erzielten auf den gelieferten Datensätzen keine kompetitiven Ergebnisse.

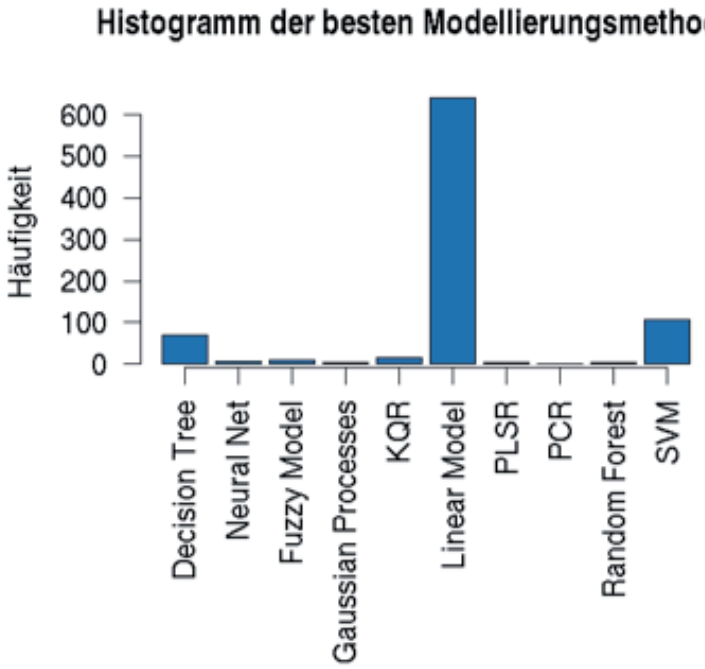

Abbildung 8: Beste Modellierungsmethoden für alle Ausgangsgrößen

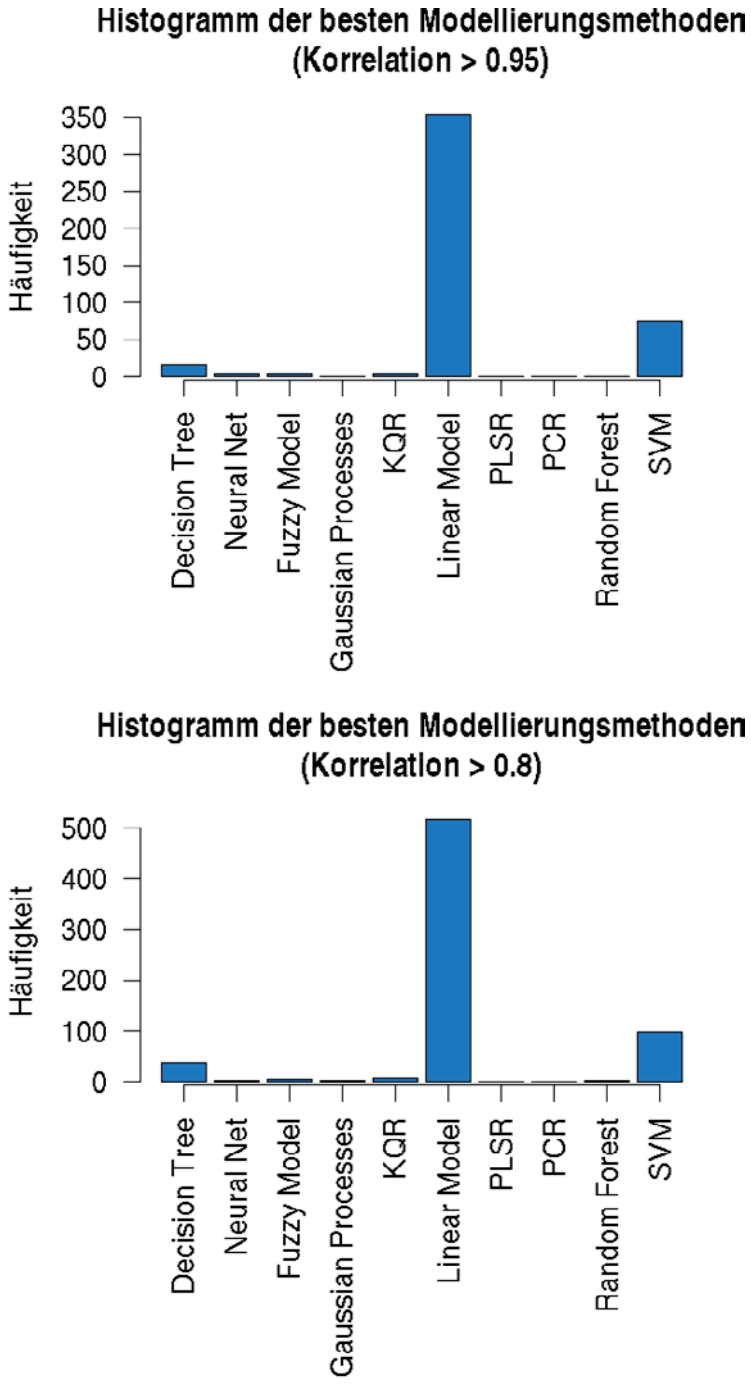

Abbildung 9: Beste Modellierungsmethoden für alle Ausgangsgrößen abhängig von den erreichten Korrelationswerten

Die nun folgenden Abbildungen zeigen die Modellierungsergebnisse für die Datensätze der einzelnen Firmen. Sie sind nach Modellierungsmethode und Korrelation aus Validierung aufgeschlüsselt und die Größe der Kreise ist proportional zur Häufigkeit. Zum Beispiel sind die Ergebnisse auf dem Datensatz der Firma ZF durchweg positiv, da nur Modelle mit einer Korrelation > 0.95 gefunden wurden. Hingegen konnte bei der Modellierung der Daten der Firma Bosch für mehr als die Hälfte der Ausgangsgrößen keine vernünftigen Modelle gefunden werden.

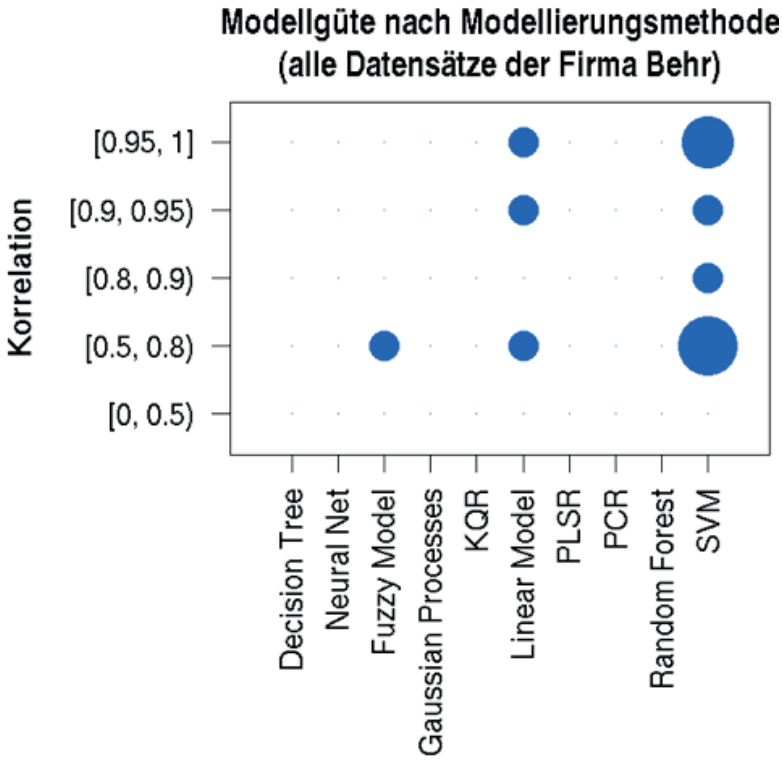

Abbildung 10: Aggregierte Modellierungsergebnisse für die Datensätze der Firma Behr

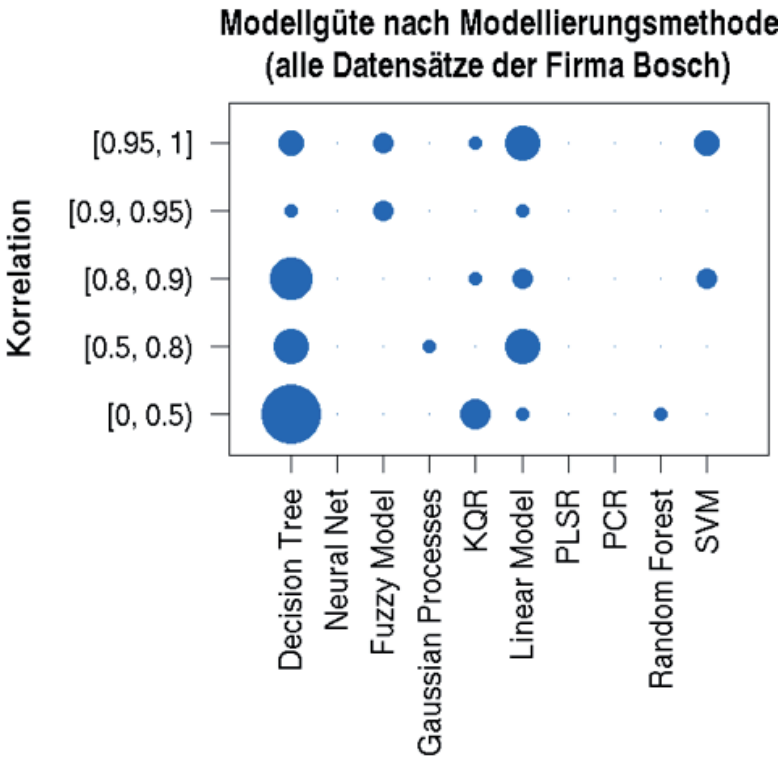

Abbildung 11: Aggregierte Modellierungsergebnisse für den Datensatz der Firma Bosch

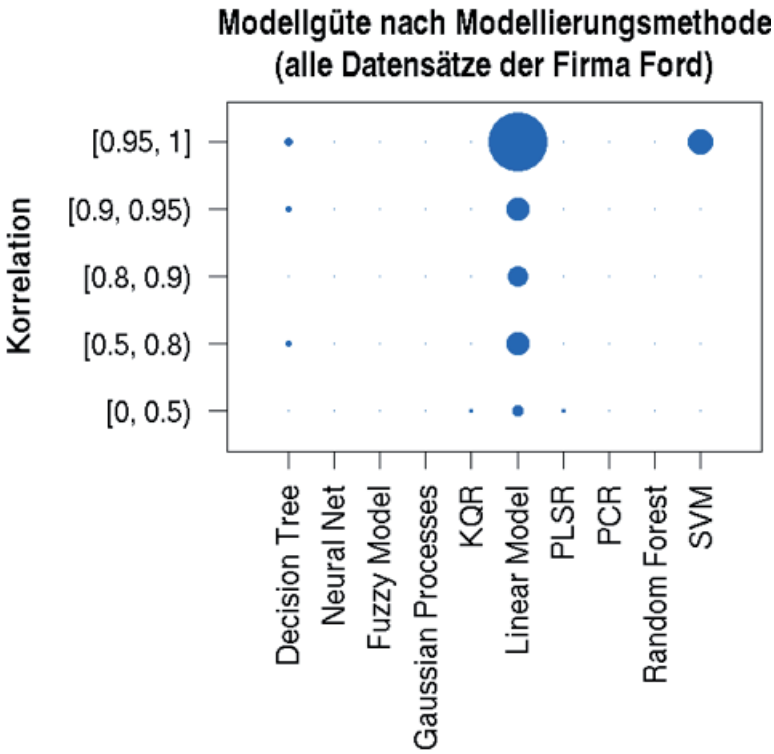

Abbildung 12: Aggregierte Modellierungsergebnisse für die Datensätze der Firma Ford

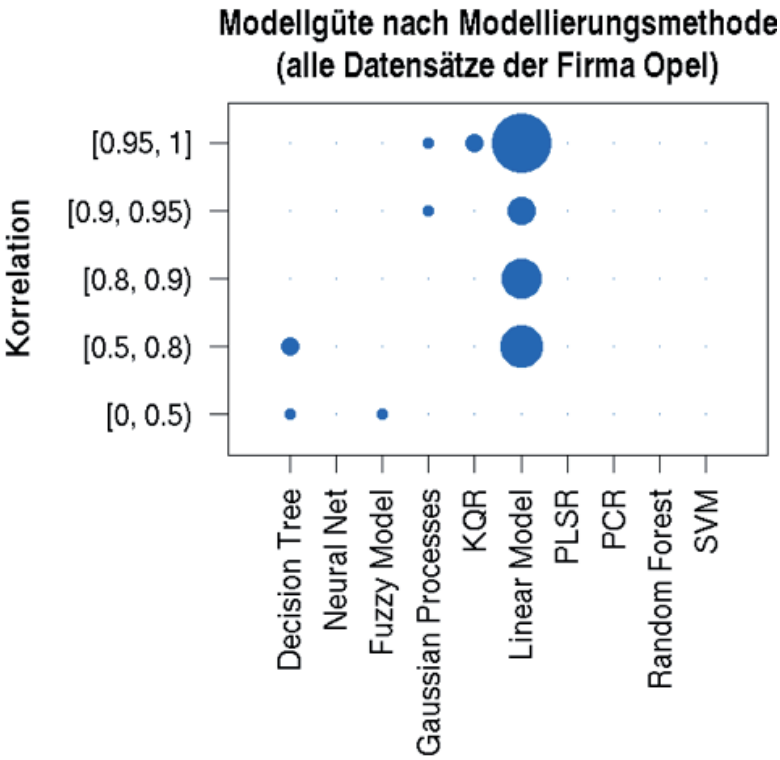

Abbildung 13: Aggregierte Modellierungsergebnisse für die Datensätze der Firma Opel

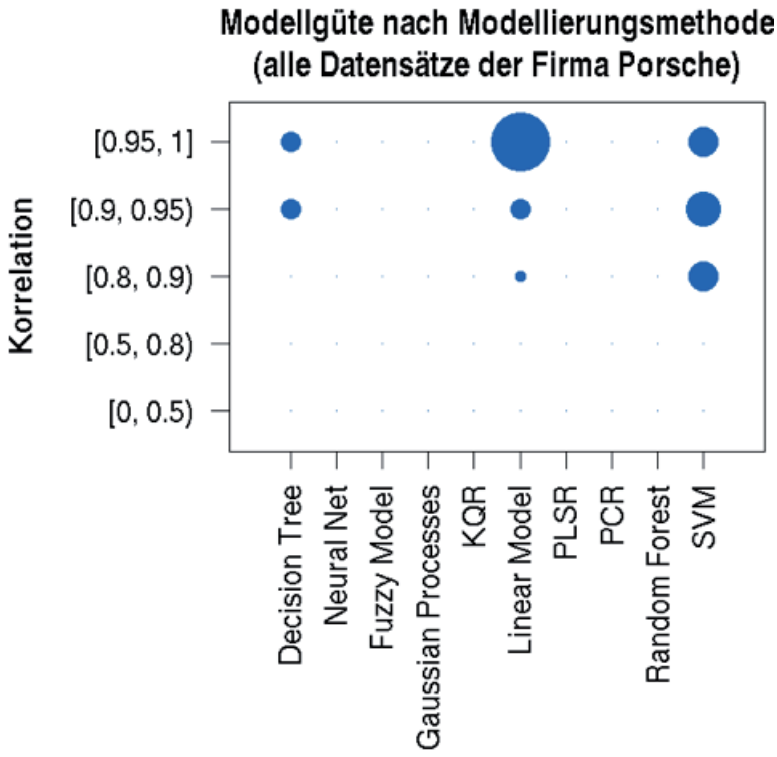

Abbildung 14: Aggregierte Modellierungsergebnisse für die Datensätze der Firma Porsche

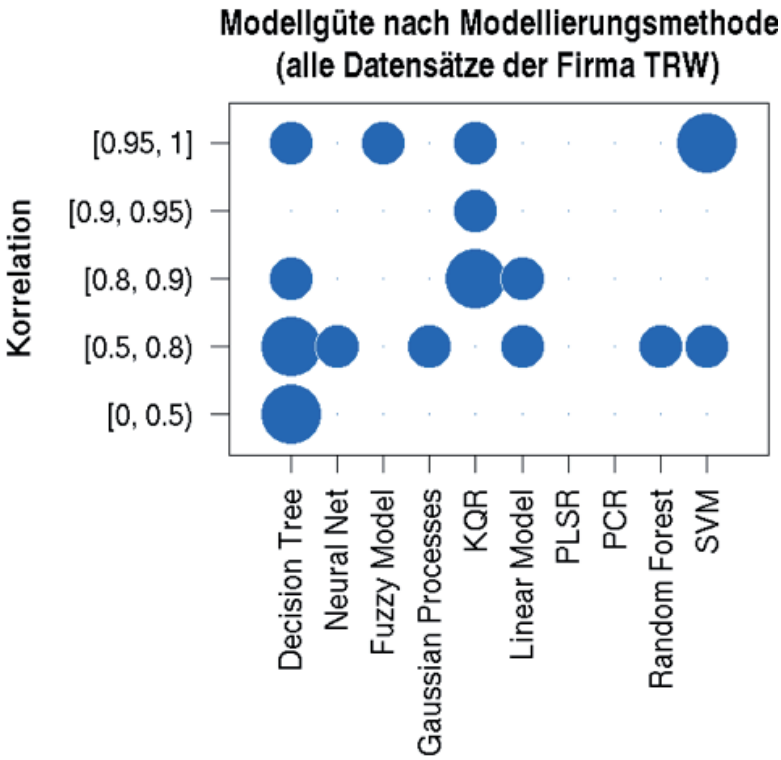

Abbildung 15: Aggregierte Modellierungsergebnisse für die Datensätze der Firma TRW

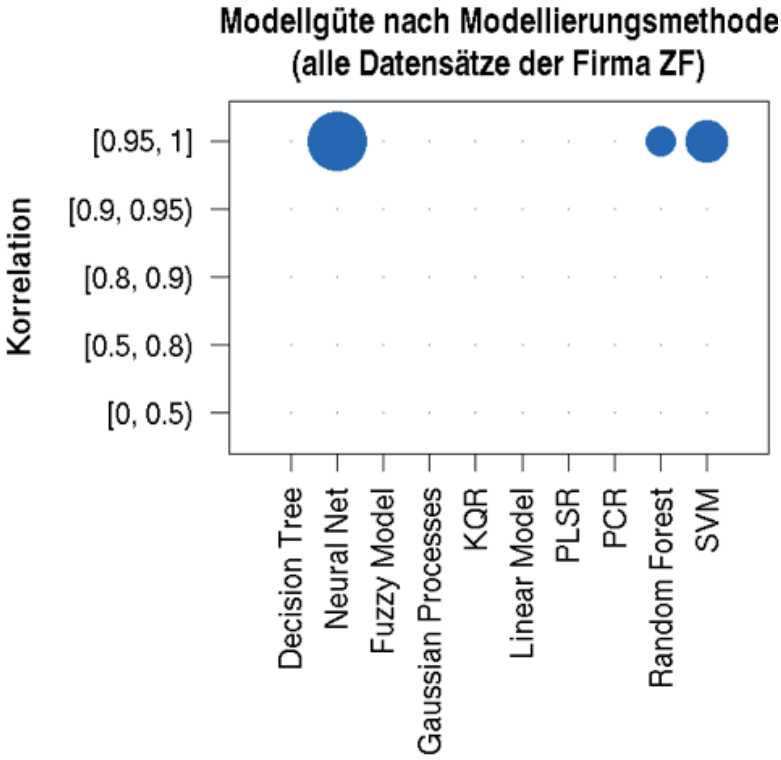

Abbildung 16: Aggregierte Modellierungsergebnisse für die Datensätze der Firma ZF

6 Zusammenfassung/Ausblick

Im Projektplan wurde die Bearbeitung des Forschungsprojektes in die Arbeitspakete

- AP 1 Datenverfügbarkeit
- AP 2.1 Benchmarking
- AP 2.2 Modellkomplexität
- AP 2.3 Datenaufbereitung
- AP 2.4 Modellauswahl
- AP 3 Robustheitsanalyse

aufgeteilt. Im Folgenden werden die Ergebnisse zu den einzelnen Arbeitspaketen zusammenfassend beschrieben.

AP 1 Datenverfügbarkeit
Die Mitglieder des AK27 UA Optimierung haben der divis 41 Datensätze mit insgesamt 861 zu modellierenden Ausgangsgrößen zur Verfügung gestellt. Die Eigenschaften der Datensätze sind in Abschnitt 5.1 aufgelistet.

AP 2.1 Benchmarking
Mit der automatisierten Modelloptimierung in ClearVu Analytics wurden alle 861 Ausgangsgrößen modelliert. Für über die Hälfte der Ausgangsgrößen konnten Modelle mit einer Korrelation größer als 0.95 gefunden werden. In den meisten Fällen waren lineare Modelle das beste Modellierungsverfahren. Aggregierte Ergebnisse der Modellierung sind in Abschnitt 5.2 zu finden, die detaillierten Ergebnisse sind ihres Umfangs wegen in einem separaten Dokument dargestellt.

AP 2.2 Modellkomplexität
Modellkomplexität wird in Abschnitt 2.6 behandelt. Aus verschiedenen Komplexitätmaßen wurden die Generalized Degrees of Freedom als generisch anwendbar und plausibel identifiziert. Ein Algorithmus zur numerischen Berechnung wurde angegeben und in ClearVu Analytics implementiert.

AP 2.3 Datenaufbereitung
Die gängigen Methoden zur Datenaufbereitung wurden auf die zur Verfügung gestellten Datensätze angewendet. Wie in Abschnitt 5.1 beschrieben, waren die Daten in einem Zustand, so dass keine Modifizierung der Daten notwendig war. Die dadurch frei gewordenen Aufwände wurden für AP 2.1 verwendet.

AP 2.4 Modellauswahl
Die Modellauswahlkriterien Kreuzvalidierung, Akaike Information Criterion, Bayesian Information Criterion und Generalized Cross-Validation sind in Abschnitt 2.5 beschrieben. Kapitel 3 widmet sich der empirischen Untersuchung zum Vergleich dieser Modellauswahlkriterien bezüglich der Generalisierungsfähigkeit während der Modelloptimierung. Bei der Untersuchung hat sich die 10-fache Kreuzvalidierung als überlegenes Modellauswahlkriterium herausgestellt. Eine generische Modelloptimierung anhand der vier Modellauswahlkriterien wurde in Matlab implementiert und der Source-Code ist in Anhang A.1 zu finden.

AP 3 Robustheitsanalyse
Zur Quantifizierung der Robustheit eines Arbeitspunktes bezüglich eines Meta-modells wurde in Kapitel 4 ein Maß hergeleitet. Für seine Berechnung wurde ein Monte-Carlo-Algorithmus entwickelt, der mit beliebigen Modellierungsmethoden anwendbar ist.

Als Fazit ist festzuhalten:

1. Automatische Verfahren zur Metamodellierung können für die Metamodellierung von CAE-Simulationsmodellen eingesetzt werden und liefern qualitativ hochwertige Modelle.
2. Ein generell bestes Verfahren für die Metamodellierung kann nicht identifiziert werden. Verallgemeinerte lineare Modelle sind allerdings für die im Rahmen des Projektes verfügbaren Daten in einer überwiegenden Vielzahl der Fälle am besten.
3. Eine Methodik zur automatisierten Metamodellierung existiert bei der divis GmbH und kann auf derartige Aufgabenstellungen angewandt werden.
4. Für die Modellauswahl ist die 10-fache Kreuzvalidierung das am besten geeignete Bewertungsmaß.
5. Die Robustheit von Arbeitspunkten kann anhand von Metamodellen abgeschätzt werden; ein geeignetes Maß dafür wurde hier vorgestellt.

In einem nächsten Schritt soll nun im Rahmen eines Folgeprojektes die Verwendung der Metamodelle in Kombination mit Optimierverfahren getestet werden, um zu bewerten, wie gut Metamodelle für die Lokalisierung von Optimallösungen im ein- sowie mehrkriteriellen Fall geeignet sind.

7 Literaturhinweise

[Ak1974] Akaike: A new look at the statistical model identification. IEEE Transactions on Automatic Control, Volume 19, Number 6, Seiten 716 - 723, Dec 1974. 1974.

[AYA2009] Aslan, Yildiz und Alpaydin: Calculating the VC-Dimension of Decision Trees. In Proceedings of The 24th International Symposium on Computer and Information Sciences (ISCIS 2009) 2009, North Cyprus. IEEE. 2009.

[Bä1996] Bäck: Evolutionary Algorithms in Theory and Practice. Oxford Uni-versity Press, New York. 1996.

[BFK2013] Bäck, Foussette und Krause: Contemporary Evolution Strategies. Springer. 2013.

[BL2013] Bache und Lichman: UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science. 2013.

[Br1984] Breiman, Friedman, Olshen and Stone: Classification and Regression Trees. Wadsworth. 1984.

[Br2001] Breiman: Random Forests. Machine Learning 45(1), Seiten 5-32. Kluwer Academic Publishers. 2001.

[HAFR2010] Hansen, Auger, Fink und Ros: Real-parameter black-box optimiza-tion benchmarking 2010: experimental setup. Research report RR-7215, INRIA. 2010.

[GL1997] El Ghaoui und Lebret: Robust Solutions To Least-Squares Problems With Uncertain Data. SIAM. 1997.

[KK1989] Kiendel und Krabs: Ein Verfahren zur Generierung regelbasierter Modelle für dynamische Systeme. at-Automatisierungstechnik, Vo-lume 37(11), Seiten 423 – 430. 1989.

[KK1995] Krone und Kiendel: Strategieelemente zur Beherrschung von Kom-plexität bei der rechnergestützten regelbasierten Modellierung. Ta-gungsband des 5. Workshops Fuzzy Control, Seiten 56 – 70. For-schungsbericht der Fakultät für Elektrotechnik, Nr. 0295, Universität Dortmund. 1995.

[Ko1995] Kohavi: A study of cross-validation and bootstrap for accuracy estimation and model selection. In Proceedings of the 14th International Joint Conference on Artificial Intelligence 2(12), Seiten 1137 – 1143. Morgan Kaufmann, San Mateo (CA). 1995.

[Kr1994] Krabs: Das ROSA-Verfahren zur Modellierung dynamischer Systeme durch Regeln mit statistischer Relevanzbewertung. Fortschritt-Berichte VDI, Reihe 8, Nr. 404. VDI Verlag, Düsseldorf. 1994.

[Kr1999] Krone: Datenbasierte Generierung von relevanten Fuzzy-Regeln zur Modellierung von Prozesszusammenhängen und Bedienstrategien. Fortschritt-Berichte VDI, Reihe 10, Nr. 615. VDI Verlag, Düsseldorf. 1999.

[LGBJ2002] Lanckriedet, El Ghaoui, Bhattacharyya und Jordan: A Robust Mini-max Approach to Classification. Journal of Machine Learning re-search, Volume 3, Seiten 555 – 582. 2002.

[IM2007] Ingrassia und Morlini: Equivalent number of degrees of freedom for neural networks. Advances in Data Analysis, Editors: Decker, Reinhold and Lenz. Springer. 2007.

[MN1989] Martens und Naes: Multivariate Calibration. Wiley. 1989.

[Ri1996] Ripley: Pattern Recognition and Neural Networks. Cambridge Uni-versity Press. 1996.

[Ro1958] Rosenblatt: The perceptron: A probabilistic model for information storage and organization in the brain. Psychological Review, Volume 65(6), Seiten 386 – 408. 1958.

[Sc1978] Schwarz: Estimating the Dimension of a Model. The Annals of Statistics, March 1978, Volume 6, Number 2, Seiten 461 – 464. The Institute of Mathematical Statistics. 1978.

[SNW2012] Sra, Nowozin und Wright: Optimization for Machine Learning. MIT Press. 2012.

[SS2004] Smola und Schölkopf: A Tutorial on Support Vector Regression. Statistics and Computing, Volume 14(3), Seiten 199 – 222. Kluwer Academic Publishers, Hingham (MA). 2004.

[Ta2006] Takeuchi, Le, Sears und Smola: Nonparametric Quantile Estimation. Journal of Machine Learning Research 7. 2006.

[Va1995] Vapnik: The nature of statistical learning theory. Springer. 1995.

[VC1971] Vapnik und Chervonenkis: On the uniform convergence of relative frequencies of events to their probabilities. Theory of Probability and its Applications, Volume 16, Seiten 264 – 280. SIAM. 1971.

[Wa1945] Wald: Statistical decision functions which minimize the maximum risk. The Annals of Mathematics, Volume 46(2), Seiten 265 – 280. 1945.

[WB1998] Williams and Barber: Bayesian classification with Gaussian processes. IEEE Transactions on Pattern Analysis and Machine Intelligence, Volume 20, Seiten 1342 - 1351. 1998.

[WC1978] Wahba und Craven: Smoothing noisy data with spline functions. Numerische Mathematik, Volume 31, Number 4, Seiten 377 – 403. Springer. 1978.

[We1975] Werbos: Beyond regression: New Tools for Prediciton and Analysis in the Behavioral Sciences. 1975.

[Ye1998] Ye: On Measuring and Correcting the Effects of Data Mining and Model Selection. Journal of American Statistical Association, March 1998, Volume 93, Number 441, Seiten 120 - 131. 1998.

[Za1965] Zadeh: Fuzzy Sets. Information Control, Volume 8(11), Seiten 338 – 353. 1965.

A. Anhang

A.1. Matlab Code zur Modelloptimierung

Die Modelloptimierung wurde in Matlab implementiert. Die Implementierung führt die Modelloptimierung mit einer Evolutionsstrategie anhand der vier in Abschnitt 2.4 beschrieben Modellauswahlkriterien durch. Sie ist für die Verwendung mit einer beliebigen Modellierungsmethode ausgelegt. Die Schnittstelle ist durch zwei Funktionen gegeben. Zum einen eine `fitFunction`, die anhand von Lerndaten ein Modell generiert, dieses auf übergebenen Testdaten anwendet, und die Vorhersagen der Testdaten sowie die Komplexität des Modells in Form von Freiheitsgraden zurückgibt. Zum anderen die Funktion `getParameterFunction`, die die zu optimierenden Parameter des Modells mit ihren Grenzen und Startwerten zurückgibt. Die Funktion `optimizeModel` optimiert diese Parameter und gibt den besten Parametersatz zurück. Detaillierte Beschreibungen können der Hilfe in Matlab entnommen werden (z.B. `help optimizeModel`). Die Implementierung ist durch die nun folgenden Funktionen gegeben.

Die Funktion zur Modelloptimierung `optimizeModel`:

```
function [ bestParameters ] = optimizeModel(dataIn, dataOut, fitFunction,
getParameterFunction, modelSelectionMethod, varargin)
%OPTIMIZEMODEL optimizes model parameters for a given data set
% Input arguments
% * dataIn - a data matrix containing the input variables of the data set
% * dataOut - a vector containing the output variable
% * fitFunction - a function with signature: [ predictions, de-
greesOfFree-dom ] = fitFunction( learnDataIn, learnDataOut, testDataIn,
testDataOut, parameters, varargin )
%                  It represents the modeling method and should train a
%                  model according to parameters on the data set
%                  [learnDataIn, learnDataOut], return the degrees of
%                  freedom and predictions made on the data set [testDataIn,
%                  testDataOut]. varargin will be passed through by
%                  optimizeModel().
% * getParameterFunction - a function with signature: [ parameters ] =
get-Parameter(varargin)
%                           This function provides the optimizable
%                           parameters of a modeling method. The argument
%                           varargin will be passed through by
%                           optimizeModel(). The return value parameters
%                           should be a cell array with four columns
%                           containing an optimizable parameter per row. The
%                           first column ist the name of the parameter, the
%                           second column is the lower bound, the third col-
umn
```

```
%                           is the start value and the fourth column is the
%                           upper bound.
% * modelSelectionMethod - the method that evaluates a model during
%                           optimization, it can be one of
%                           CV (10-fold cross validation)
%                           AIC (Akaike information criterion)
%                           BIC (Bayesian information criterion)
%                           GCV (generalized cross validation)
% * varargin - additional user defined arguments that will be passed to
%              fitFunction and getParameterFunction
% Return value
% * bestParameters - the set of best parameters found during the model
%                    optimization
%
parameters = feval(getParameterFunction, varargin);
parameterSize = size(parameters);
bestParameters = parameters;

% init optimization
mu = 2;
lambda = 10;
maxFunctionEvaluations = 1000;
es = ES_Optimization(cell2mat(parameters(:,2))',cell2mat(parameters(:,4))',
maxFunctionEvaluations, mu, lambda);
es = es.initOptimization;
bestFitness = Inf;
stagnationCounter = 0;
maxGenerations = floor(maxFunctionEvaluations/lambda);
% run ES loop
while(~es.isFinished && stagnationCounter < 0.05*maxGenerations)
   [ es, parameterSuggestions ] = es.getNextOffsprings;
   fitness = NaN*ones(lambda, 1);
   for(i=1:lambda)
       for(j=1:parameterSize(1))
           parameters(j,3) = {parameterSuggestions(i,j)};
       end
       fitness(i, 1) = evaluateModel( modelSelectionMethod, dataIn, dataOut,
fitFunction, parameters, varargin{:} );
   end
```

```
    es = es.setOffspringFitness(fitness);
    if(es.getBestFitness < bestFitness)
        bestFitness = es.getBestFitness;
        stagnationCounter = 0;
    else
        stagnationCounter = stagnationCounter + 1;
    end
end

bestIndividual = es.getBestIndividual';
for(i=1:parameterSize(1))
    bestParameters{i,3} = bestIndividual(i,1);
end

end
```

Die interne Funktion `evaluateModel`, die von `optimizeModel` aufgerufen wird:

```
function [ value ] = evaluateModel( method, dataIn, dataOut, fitFunction,
parameters, varargin )
% internal function to evaluate a model, invoked by optimizeModel()

value = 0;
dataInSize = size(dataIn);
if(strcmp(method, 'CrossValidation'))
    % split data in folds
    numFolds = 10;
    if(dataInSize(1) < numFolds)
        error('data has not enough rows for a %d-fold cross-validati-
on\n',numFolds);
    end
    randIndices = randperm(dataInSize(1));
    foldRanges = zeros(numFolds,2);
    if(mod(dataInSize(1), numFolds) > 0)
        for(i = 1:mod(dataInSize(1), numFolds))
            foldRanges(i,1) = ceil(dataInSize(1)/numFolds)*(i-1)+1;
            foldRanges(i,2) = foldRanges(i,1) + (ceil(dataInSize(1)/num-
Folds)-1);
        end
```

```
        for(i = mod(dataInSize(1), numFolds)+1:numFolds)
            foldRanges(i,1) = foldRanges(i-1,2)+1;
            foldRanges(i,2) = foldRanges(i,1)+(floor(dataInSize(1)/num-
Folds)-1);
        end
    else
        foldRanges(:,1) = ( (dataInSize(1)/numFolds) * (0:numFolds-1) + 1
)';
        foldRanges(:,2) = ( (dataInSize(1)/numFolds) * (1:numFolds) )';
    end
    % perform 10-fold cross-validation
    foldRMSE = NaN*ones(numFolds,1);
    for(i=1:numFolds)
        learnDataIn = [];
        learnDataOut = [];
        for(j=1:numFolds)
            if(i ~= j)
                learnDataIn = cat(1, learnDataIn, da-taIn(randIndices(fol-
dRanges(j,1):foldRanges(j,2)),:));
                learnDataOut = cat(1, learnDataOut, dataOut(randIndices(-
foldRanges(j,1):foldRanges(j,2))));
            end
        end
        testDataIn = da-taIn(randIndices(foldRanges(i,1):foldRan-
ges(i,2)),:);
        testDataOut = dataOut(randIndices(foldRanges(i,1):foldRan-
ges(i,2)));
        [ predictions, dof ] = feval(fitFunction, learnDataIn, learn-Data-
Out, testDataIn, testDataOut, parameters, varargin{:});
        foldRMSE(i,1) = sqrt(sum((predictions - testDataOut).^2));
    end
    % calculate RMSE as mean over all folds
    value = mean(foldRMSE);
elseif(strcmp(method, 'AIC'))
    [ predictions, dof ] = feval(fitFunction, dataIn, dataOut, dataIn, data-
Out, parameters, varargin{:} );
    % calculate AIC
    value = 2*dof + dataInSize(1)*log(sum((predictions - dataOut).^2)/da-
taInSize(1));
elseif(strcmp(method, 'BIC'))
```

```
    [ predictions, dof ] = feval(fitFunction, dataIn, dataOut, dataIn, data-
Out, parameters, varargin{:});
    % calculate BIC
    value = dataInSize(1)*log(sum((predictions - dataOut).^2)/dataInSi-
ze(1)) + dof*log(dataInSize(1));
elseif(strcmp(method, 'GCV'))
    [ predictions, dof ] = feval(fitFunction, dataIn, dataOut, dataIn, data-
Out, parameters, varargin{:} );
    % calculate GCV
    value = (sum((predictions - dataOut).^2)/dataInSize(1))/((dataInSize(1)
- dof)^2);
end

end
```

Die Klasse `ES_Optimization`, die eine populationsbasierte Evolutionsstrategie mit Selbstanpassung implementiert:

```
classdef ES_Optimization
    % Evolution Strategy Optimization
    % as class implementation with ask & tell interface
    % instructions to run an optimization loop:
    % * create an ES object with the constructor configured by the
    %   arguments:
    %   lowerBounds - a vector with the lower bounds of the design space
    %   upperBounds - a vector with the upper bounds of the design space
    %   maxFunctionEvaluations - after how much fitness function
    %                            evaluations the algorithm should stop
    %   mu - the count of parent individuals
    %   lambda - the count of offspring individuals
    % * initialize optimization with initOptimization
    % * getNextOffsprings returns initial population to evaluate
    % * set the evaluation for the initial population with
    %   setOffspringFitness
    % * loop until isFinished returns true
    %      * getNextOffsprings
    %      * evaluate the offsprings
    %      * setOffspringFitness
    % * get the best individual with getBestIndividual
```

```
    properties
        m_currentGeneration = 0;
        m_lowerBounds = [];
        m_upperBounds = [];
        m_mu = 2;
        m_lambda = 10;
        m_offsprings = [];
        m_fitness = [];
        m_localSigmas = [];
        m_evaluationCount = 0;
        m_bestIndividual = [];
        m_bestFitness = Inf;
        m_maxFunctionEvaluations = 0;
        m_isEvaluated = false;
        m_dim = 0;
        m_beta = 0.0873;
        m_tau = 0.5946;
        m_tauPrime = 0.5;
    end

    methods
        function esOpt = ES_Optimization(lowerBounds, upperBounds, max-
Func-tionEvaluations, mu, lambda)
            lowerBoundsSize = size(lowerBounds);
            upperBoundsSize = size(upperBounds);
            if(lowerBoundsSize ~= upperBoundsSize)
                error('dimensions of lower und upper bounds differ');
            else
                if(lowerBoundsSize(1) > lowerBoundsSize(2))
                    lowerBounds = lowerBounds';
                    upperBounds = upperBounds';
                end
            end
            esOpt.m_dim = lowerBoundsSize(2);
            esOpt.m_lowerBounds = lowerBounds;
            esOpt.m_upperBounds = upperBounds;
            esOpt.m_maxFunctionEvaluations = maxFunctionEvaluations;
            esOpt.m_mu = mu;
            esOpt.m_lambda = lambda;
        end
```

```
        function esOpt = initOptimization(esOpt)
            esOpt.m_evaluationCount = 0;
            esOpt.m_currentGeneration = 0;
            esOpt.m_isEvaluated = false;
            % generate initial population
            for(i=1:esOpt.m_lambda)
                esOpt.m_offsprings(i,:) = esOpt.m_lowerBounds + (esOpt.m_
upperBounds-esOpt.m_lowerBounds).*rand(1, esOpt.m_dim);
                esOpt.m_localSigmas(i,:) = 0.3*(esOpt.m_upper-
Bounds-esOpt.m_lowerBounds);
            end
        end
        function [esOpt, offsprings] = getNextOffsprings(esOpt)
            if(esOpt.m_currentGeneration > 0)
                if(~esOpt.m_isEvaluated)
                    offsprings = [];
                    warning('ES_Optimization:getNextOffsprings','offsprings
must be evaluated first');
                else
                    % selection
                    parent = zeros(1, esOpt.m_dim);
                    sigmas = zeros(1, esOpt.m_dim);
                    [ foo, sortedIndices ] = sort(esOpt.m_fitness);
                    for(i=1:esOpt.m_mu)
                        parent(1,:) = parent(1,:) + esOpt.m_offsprings(sor-
tedIndices(i),:);
                        sigmas(1,:) = sigmas(1,:) + esOpt.m_localSig-
mas(sortedIndices(i),:);
                    end
                    parent(1,:) = parent(1,:) / esOpt.m_mu;
                    sigmas(1,:) = sigmas(1,:) / esOpt.m_mu;
                    % recombination
                    for(i=1:esOpt.m_lambda)
                        esOpt.m_offsprings(i,:) = parent(1,:);
                        esOpt.m_localSigmas(i,:) = sigmas(1,:);
                    end
```

```
                    % mutation
                    for(i=1:esOpt.m_lambda)
                        tauPrimeContrib = esOpt.m_tauPrime*randn(1,1);
                        esOpt.m_localSigmas(i,:) = esOpt.m_localSigmas(i,:)
.* exp(tauPrimeContrib + esOpt.m_tau*randn(1, esOpt.m_dim));
                        offset = esOpt.m_localSigmas(i,:) .* randn(1,
esOpt.m_dim);
                        for(j=1:esOpt.m_dim)
                            esOpt.m_offsprings(i,j) = keepIn-Bounds(e-
sOpt.m_offsprings(i,j), offset(j), esOpt.m_lowerBounds(j), esOpt.m_upper-
Bounds(j));
                        end
                    end
                    offsprings = esOpt.m_offsprings;
                end
            else
                offsprings = esOpt.m_offsprings;
            end
            esOpt.m_isEvaluated = false;
        end
        function esOpt = setOffspringFitness(esOpt, fitnessValues)
            esOpt.m_fitness = fitnessValues;
            esOpt.m_evaluationCount = esOpt.m_evaluationCount + esOpt.m_
lambda;
            esOpt.m_isEvaluated = true;
            esOpt.m_currentGeneration = esOpt.m_currentGeneration + 1;
            for(i=1:esOpt.m_lambda)
                if(esOpt.m_fitness(i) < esOpt.m_bestFitness)
                    esOpt.m_bestFitness = esOpt.m_fitness(i);
                    esOpt.m_bestIndividual = esOpt.m_offsprings(i,:);
                end
            end
        end
        function individual = getBestIndividual(esOpt)
            individual = esOpt.m_bestIndividual;
        end
        function bestFitness = getBestFitness(esOpt)
            bestFitness = esOpt.m_bestFitness;
        end
        function isFinished = isFinished(esOpt)
```

```
            if(esOpt.m_evaluationCount >= esOpt.m_maxFunctionEvaluations)
                isFinished = true;
            else
                isFinished = false;
            end
        end
    end

end
```

Die Hilfsfunktion `keepInBounds` und `toBoxModulo`, die von `ES_Optimization` benötigt werden:

```
function r = keepInBounds(x,offset,lb,ub)
% function r = keepInBounds(x,offset,lb,ub)
% adds offset to x w.r.t. the lower bound lb and the upper bound ub
% by reflecting from the boundaries
    r = x + offset;
    r = toBoxModulo(r,lb,ub);
end

function t = toBoxModulo(x,lb,ub)
% function t = toBoxModulo(x,lb,ub)
% This function transforms x to t w.r.t. the low and high
% boundaries lb and ub. It implements the function T^{r}_{[a,b]} as
% described in Rui Li's PhD thesis "Mixed-Integer Evolution Strategies
% for Parameter Optimization and Their Applications to Medical Image
% Analysis" as alorithm 6.
if(isfinite(lb)&&isfinite(ub))
    y = (x-lb)/(ub-lb);
    if(mod(floor(y),2)==0)
        yprime = abs(y - floor(y));
    else
        yprime = 1 - abs(y - floor(y));
    end
    t = lb+(ub-lb)*yprime;
else
    %warning('both boundaries have to be finite, x will be unchanged');
    t = x;
end
end
```

A.2. Detaillierte Modellierungsergebnisse der zur Verfügung gestellten Datensätze

Die detaillierten Modellierungsergebnisse sind in dem zusätzlichen Dokument Anhang_ Modellierungsergebnisse.pdf zu finden.

Impressum

Herausgeber	FAT Forschungsvereinigung Automobiltechnik e.V. Behrenstraße 35 10117 Berlin Telefon +49 30 897842-0 Fax +49 30 897842-600 www.vda-fat.de
ISSN	2192-7863

VDA | Verband der Automobilindustrie

FAT | Forschungsvereinigung Automobiltechnik

Behrenstraße 35
10117 Berlin
www.vda.de
www.vda-fat.de