Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

Fachbereich
Informatik

Free and Open Source Software Conference

Science Track FrOSCon 2016

Sayeed Klewitz-Hommelsen, Martin Lang, Bernd Schönbach (Hrsg./Eds.)

Digital Object Identifier doi:10.18418/978-3-96043-032-2 **DOI-Resolver** http://dx.doi.org/

ISBN 978-3-96043-032-2 Veröffentlicht : 27. April 2018

Inhaltsverzeichnis

1 Vorwort

Autor:
Sayeed
Klewitz-
Hommelsen

FrOSCon Science 2016

Im Jahre 2015 feierte die Free and Open Source Software Conference ihr 10 Jähriges Bestehen. Entstanden aus einer Idee von Studierenden, wissenschaftlichen Mitarbeitern und Professoren des Fachbereichs Informatik entwickelte sich eine der wichtigsten Konferenzen im Bereich der freien und quelloffenen Software in Deutschland.

Von Anfang an fand diese Konferenz in den Räumlichkeiten der Hochschule Bonn-Rhein-Sieg statt. Eigentlich lag es deshalb nahe, neben den praktischen und anwendungsbezogenen Themen auch stärker wissenschaftliche Bezüge herzustellen. Letztlich führten diese Überlegungen dazu, zur 11. FrOSCon 2016 zum ersten Science Track einzuladen. In diesen Proceedings finden sich die ersten Beiträge aus dieser Reihe. Gewünscht waren Beiträge die entweder die Entwicklung oder den Einsatz von Open Source Software in der Forschung im Fokus haben.

Ich danke allen Mitwirkenden an der Entstehung dieses ersten FrOSCon Science Bandes, der in den nächsten Jahren fortgesetzt werden soll. Mein besonderer Dank gilt Robert Hartmann, der sich insbesondere um die druckfertige Aufbereitung gekümmert hat und dem Biblitotheksteam der Hochschule Bonn-Rhein-Sieg, die das Hosting dieser elektronischen Publikation übernommen haben.

Sankt Augustin, Juli 2017
Sayeed Klewitz-Hommelsen

Sayeed Klewitz-Hommelsen, Martin Lang, Bernd Schönbach (Hrsg./Eds.)
FrOSCon Science 2016
Tagungsband der 11. Free and Open Source Conference
Proceedings of the 11th Free and Open Source Conference
Veröffentlicht : 27. April 2018

Digital Object Identifier doi:10.18418/978-3-96043-032-2
DOI-Resolver http://dx.doi.org/
ISBN 978-3-96043-032-2

2 Data Mining in astronomischen Surveydaten variabler Sterne mit Python

Autor: Christian Dersch, Andreas Schrimpf, Milan Spasovic

Variable Sterne

Variable Sterne sind Sterne, welche in bestimmten Messparametern variabel sind. In unserem Fall ist dies die Helligkeit der Sterne. Grundsätzlich gibt es hier zwei Arten der Variabilität, intrinsiche und extrinsische Prozesse. Unter intrinischen Prozessen versteht man Variabilität, deren Ursache im Stern selbst liegt. Ein klassisches Beispiel hierfür sind die verschiedenen Arten von Cepheiden, Sternen welche sich in Schwingung befinden. Ihre Variabilität zeichnet sich durch eine Beziehung zwischen Helligkeit und Periodizität aus (sog. Periode-Leuchtkraft-Beziehung) und wurde von Edwin Hubble 1923 zur Entfernungsbestimmung der Andromedagalaxie verwendet. Die zweite Art vom Prozessen sind die extrinsischen Variablen, hier liegt die Ursache der Variabilität außerhalb des Sterns. Ein typisches Beispiel sind Doppelsternsysteme, wie in Abbildung 2.1 dargestellt.

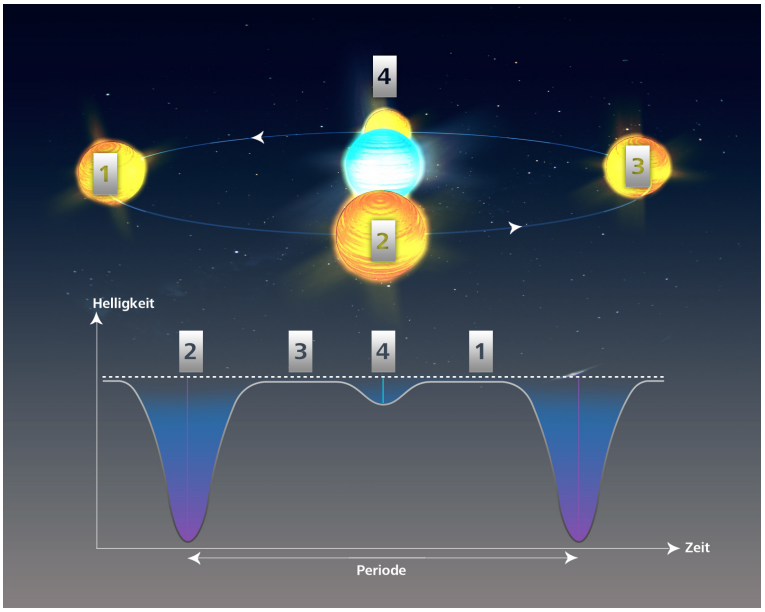


Abbildung 2.1: Doppelsternsystem als extrinischer Variabler [9].

Astronomische Surveys

Unter Surveys, auch Himmelsüberwachung genannt, versteht man systematische Durchmusterungen größrer Himmelsbereiche. Diese Durchmusterungen haben im Regelfall entweder die Erfassung statistischer Kenngrößen, beispielsweise die Verteilung bestimmter Sternpopulationen, oder das Auffinden bestimmter Objekte oder Events zum Ziel. Ein prominentes Beispiel hierfür ist die Suche nach Exoplaneten, für welche das Weltraumteleskop Kepler seit 2009 einen kleinen Bereich unserer Milchstraße regelmäßig aufnimmt.

OGLE-III

Das **OGLE!** (**OGLE!**) [31] ist ein seit 1992 laufender Survey, welcher sich mittlerweile in der vierten Projektphase OGLE-IV befindet. Wissenschaftliches Ziel des Projektes ist die Untersuchung des sog. Mikro-Gravitationslinseneffektes, bei welchem sich die Helligkeit eines Objektes erhöht, wenn ein massives Objekt die Sichtlinie zwischen Erde und Objekt passiert. Hierfür wird ein Bereich des Südhimmels regelmäßig aufgenommen, als Nebenprodukt werden von OGLE auch Exoplaneten und variable Sterne gefunden. Im Rahmen der dritten Projektphase **OGLE!**-III wurde daher auch der **OIII-CVS!** (**OIII-CVS!**) [30] mit fast 400.000 variablen Sternen veröffentlicht.

Data Mining und Maschinelles Lernen

Die Analyse großer Datenmengen kann nur noch automatisiert geschehen. In der Astrophysik geschieht dies häufig durch theoretisch motivierte Modelle, welche als Grundlage zur Entwicklung der Analysesoftware dienen. In den letzten Jahren wurden vermehrt Methoden des maschinellen Lernens bzw. des Data Minings verwendet. Diese basieren auf Verfahren der linearen Algebra und der Statistik. Hier gibt es zwei große Klassen von Verfahren, welche in den folgenden beiden Abschnitten kurz diskutiert werden, das Ziel ist die Einteilung von Objekten in, im Idealfall physikalisch direkt interpretierbare, Klassen.

Unterstütztes Lernen

Ein häufiger Anwendungsfall ist die Klassifizierung von Objekten in bekannte Klassen. Hier werden Algorithmen des unterstützten Lernens (engl. supervised learning), beispielsweise auf Entscheidungsbäumen basierend, verwendet. Die hierzu nötigen Entscheidungsparameter werden aus einem vorklassifizierten Musterdatensatz berechnet, wofür z.B. neuronale Netze verwendet werden.

Selbstständiges Lernen

Liegt hingegen keine Klassifizierung vor, beispielsweise weil vollkommen neuartige Daten vorliegen, man nach noch nicht bekannten Klassen oder einer besseren Klassifikation sucht, so werden Algorithmen des selbstständigen Lernens eingesetzt (engl. unsupervised learning). Ein

Beispiel für solche Algorithmen sind Clusteralgorithmen wie k-Means. Hier wird, eingegrenzt durch vorgegebene Kriterien wie z.B. die Anzahl der Cluster, eine Klassifikation der Daten vorgeschlagen.

Python in Astronomie und Astrophysik

Wissenschaftliches Rechnen mit Python - der SciPy-Stack

Eine häufig gestellte Frage ist die der Eignung von Python für das wissenschaftliche Rechnen. Python ist als Skriptsprache angelegt und daher durch zur Laufzeit notwendige Aufgaben wie Typprüfungen verglichen mit kompilierten Sprachen deutlich langsamer. Dem gegenüber steht die Möglichkeit des interaktiven Arbeitens, was gerade bei der Datenanalyse eine große Erleichterung ist. Abhilfe schafft hier der SciPy-Stack [39], bestehend aus den Kernmodulen NumPy und SciPy sowie weiteren Modulen, z.B. Matplotlib zur graphischen Darstellung und IPython zum interaktiven Arbeiten. Um eine hohe Performanz zu erreichen implementiert der SciPy-Stack die aufwändigen Operationen in C oder Fortran, insbesondere werden Array-Datentypen eingeführt, welche die Verwendung von Schleifen stark reduzieren und zur Laufzeit aufwändige Prüfungen vermeiden.

Astropy - Astronomie mit Python

Speziell für astronomische und astrophysikalische Anwendungen wurde das Astropy Projekt [4] initiiert. Neben dem Kernmodul astropy, welches die grundlegenden Operationen wie z.B. Koordinatentransformationen und Unterstützung für Kataloge bereitstellt, werden einige weitere Module für speziellere Zwecke entwickelt. So ermöglichen beispielsweise die Pakete ccdproc und photutils die photometrische Analyse von Messaufnahmen. Astropy setzt auf dem SciPy-Stack auf und integriert sich daher sehr gut mit weiteren darauf basierenden Paketen.

scikit-learn - Data Mining mit Python

Ebenfalls auf dem SciPy-Stack aufbauend implementiert das scikit-learn Projekt [23] Algorithmen des Data Minings und des maschinellen Lernens zur Verwendung in Python. Scikit-learn stellt hier die ganze Bandbreite an Algorithmen zur Verfügung:

- cluster: Clusteralgorithmen
- decomposition: Principal Component Analysis, Independent Component Analysis
- ensemble: Ensemble-Klassifikatoren, z.B. Random Forest
- tree: Entscheidungsbäume

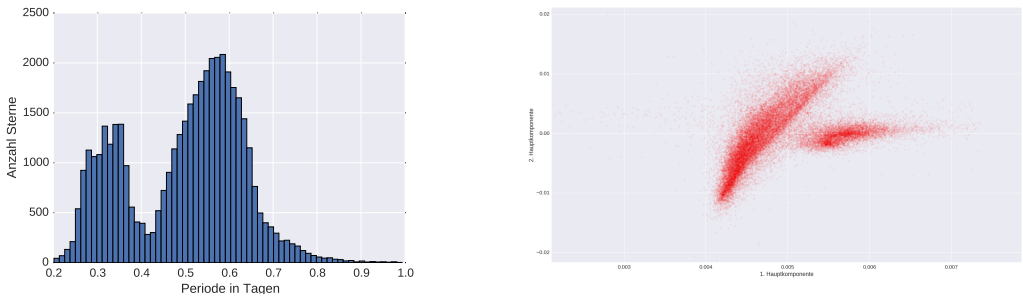
Weitere Pakete

Mit dem SciPy-Stack, Astropy und scikit-learn sind die zentralen Module für das Data Mining vorhanden. Allerdings gibt es darauf aufbauend eine Vielzahl weiterer Module, welche die Datenanalyse erleichtern:

- astroML: Erweiterungen und Beispiele zum Data Mining mit Python, basierend auf Astropy und scikit-learn [2]
- gatspy: Astronomische Zeitreihenanalyse [1]
- pandas: Datenanalyse und -strukturen [32]
- statsmodels: Statistische Methoden [42]

Beispiel: RR-Lyrae in der Großen Magellanschen Wolke

Im **OIII-CVS!** werden knapp 400.000 variable Sterne katalogisiert. Neben allgemeinen Angaben wie Position und mittlerer Helligkeit in zwei Frequenzbereichen werden auch die zugehörigen Lichtkurven/Zeitreihen und, sofern möglich, die Fourierkomponenten (Zerlegung der Zeitreihe) angegeben. Letztere lassen sich auch mit den Optimierungsfunktionen von scipy (curve_fit) berechnen. Am Beispiel der RR-Lyrae-Sterne in der Großen Magellanschen Wolke wird die Anwendung von Methoden des Data Minings gezeigt. RR-Lyrae sind kurzperiodische Variable (Periodendauer unter 1 Tag), welche im Wesentlichen zwei Schwingungszustände zeigen. Bereits das Histogramm der Periodizität deutet dies an (Abb. 2.2a): In diesem Fall

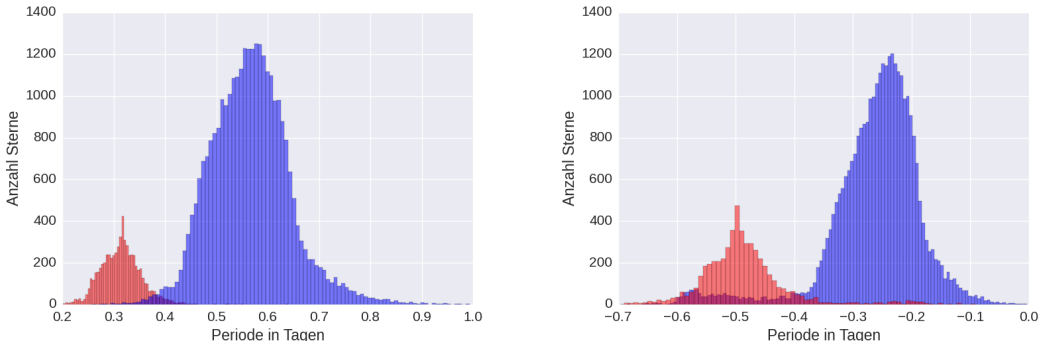


(a) Verteilung der RR-Lyrae-Perioden.

(b) Erste 2 Hauptkomponenten.

Abbildung 2.2: Klassifikation mit k-Means Clusterung.

bietet sich die Analyse der Fourierzerlegung an, zur optimalen Ausrichtung der Daten wird eine **PCA!** (**PCA!**) angewendet. Abb. 2.2b zeigt die ersten beiden Hauptkomponenten, hier sind klar zwei Cluster zu erkennen. Hierauf wird zur Klassifikation nun entsprechend ein Cluster-Algorithmus, in diesem Falle k-Means, angewendet. Schließlich wird mittels der erhaltenen Klassifikation die Periodizität erneut dargestellt. Abb. 2.3a zeigt, dass die gefundenen Klassen klar den beiden Schwingungszuständen entsprechen. Sind die Daten bereits für sich sehr klar auf das Problem zugeschnitten, bei Fourierkomponenten im Fall von Schwingungszuständen



(a) Klassifizierung mit PCA.

(b) Klassifizierung ohne PCA.

Abbildung 2.3: Klassifikation mit k-Means Clusterung.

ist dies die physikalische Grundlage, so treten die Klassen auch ohne vorige Transformation klar heraus (Abb. 2.3b). Durch die nicht ganz optimale Ausrichtung sind die Klassen ohne **PCA!** allerdings etwas schlechter getrennt.

Zusammenfassung

Mit Python, dem SciPy-Stack, Astropy, scikit-learn und darauf aufbauenden Modulen ist es möglich, Daten aus astronomischen Surveys professionell zu analysieren. Hierbei ist es möglich, startend mit den Messaufnahmen, alle notwendigen Schritte mit Python durchzuführen, weitere Werkzeuge sind nicht notwendig. Das Astropy-Projekt unternimmt seit 2011 große Anstrengungen, Python zur standardmäßigen Programmiersprache in astronomischen Datenanalysen zu machen und so proprietäre Lösungen wie z.B. IDL durch freie Software abzulösen. Die sehr aktive Community im Bereich des wissenschaftlichen Einsatzes von Python erweitert die bestehenden Module kontinuierlich und erweiert sie um neue Module. Es ist daher zu erwarten, dass sich Python in Astronomie und Astrophysik weiter etabliert und sich in den nächsten Jahren viele weitere interessante Möglichkeiten ergeben.

3 Automatisierte Strukturauslegung im Flugzeugvorentwurf mit Python

Autor:
Michael
Petsch

Flugzeugvorentwurf Allgemein

Beim Entwurf eines effizienten und sicheren Luftfahrzeugs müssen viele fachliche Aspekte berücksichtigt werden. Die Bereiche Aerodynamik, Strukturmechanik als auch Flugmechanik spielen eine wichtige Rolle und hängen voneinander ab. Daher ist ein iterativer Entwurfsprozess erforderlich, um einen an die Anforderungen bestmöglich angepassten Kompromiss zu finden. In der Forschung am Deutschen Zentrum für Luft- und Raumfahrt e.V. (DLR) werden dafür automatisierte Prozessketten entwickelt, die zur Bewertung und Entwicklung von neuen Flugzeugkonzepten dienen. Das Institut für Bauweisen und Strukturtechnologie (BT) des DLR in Stuttgart hat dafür das Rumpfstrukturmodellierungs und -dimensionierungs-Tool (TRAFUMO [37][38]) entwickelt. Um mehr Schnittstellen mit anderen Tools zu schaffen, wird aktuell eine Python basierte Open Source Toolumgebung als Nachfolger von TRAFUMO entwickelt, welche im Kapitel 3 dargestellt wird.

Prozesskette Flugzeugvorentwurf am DLR

Für den automatisierten, multidisziplinären Entwurfsprozess eines Luftfahrzeugs am DLR werden spezifische Vorentwurfs-Tools über die Remote Component Environment (RCE) [11] Software des DLR zu einer Prozesskette verbunden. Um den Datenaustausch zwischen verschiedenen Vorentwurfs-Tools zu realisieren wurde ein XML basiertes Datenaustauschformat entwickelt. Dieses Austauschformat genannt Common Aircraft Configuration Schema (CPACS [29][10]) wird fortlaufend erweitert und enthält Parameter zur Beschreibung eines Luftfahrzeugentwurfs. Diese reichen von der Beschreibung der Oberflächengeometrie über die Definition von Strukturkomponenten bis hin zu Missionsbeschreibungen.
Das Tool TRAFUMO vom DLR-BT basiert weitestgehend auf der Skriptsprache APDL der kommerziellen Finite Elemente (FE) Software ANSYS [18]. Neben den Vorteilen mit APDL auf viele Funktionalitäten in ANSYS skriptbasiert zugreifen zu können, ergeben sich gleichzeitig viele Nachteile z.B. durch die eingeschränkte Nutzung von Vektoroperationen gegenüber einer universellen Programmiersprache wie Python mit umfangreichen Modulen wie NumPy. Die fehlende Objektorientierung erschwert den Aufbau einer übersichtlichen und modularen Toolumgebung. Auch lässt sich der Funktionsumfang in APDL nur bedingt erweitern, wodurch komplexere Algorithmen nicht realisierbar sind oder aufwändig implementiert werden müssen und dadurch mehr Rechenzeit benötigen als vergleichbare Umsetzungen in Python.
Um die Strukturmodellierungs- und Optimierungsmöglichkeiten zukünftig offener und flexibler zu gestalten, wird aktuell eine komplett Python basierte Toolumgebung entwickelt. Erweiter-

te Möglichkeiten ergeben sich durch die Einbindung von Modulen wie NumPy, Pandas oder Mayavi sowie in der Nutzung von weiterer Open Source Software. Ein Einblick in den Aufbau sowie die Möglichkeiten der Open Source basierten Toolumgebung wird im Kapitel 3 gegeben.

Open Source Toolumgebung zur Strukturauslegung

Ziel der Open Source basierten Toolumgebung für die Strukturauslegung ist sowohl die Generierung einer Luftfahrtstruktur (primär Rumpfstruktur) über einen CPACS-Datensatz sowie die iterative Anpassung der Struktur an auftretende Lasten. Eine Verwendung des Strukturmodells für zusätzliche Crash-Berechnungen stellt zudem ein Sekundärziel dar. Die klar definierten Aufgabenbereiche dieser Toolumgebung repräsentieren auch die einzelnen Tools, diese sind:

- Verwaltung von CPACS XML-Daten (Kapitel 3)
- Aufbereitung von CPACS-Geometrie (Kapitel 3)
- Verwaltung von FE-Daten (Kapitel 3)
- Modellierung eines Basis-FE-Modells (Kapitel 3)
- Modellierung eines detaillierten FE-Modells (Kapitel 3)
- Konvertierung von FE-Daten (Kapitel 3)
- Dimensionierung von FE-Daten (Kapitel 3)

Durch diese Trennung vereinfacht sich die Ausbaufähigkeit der einzelnen Tools und gleichzeitig können weitere Anwendungsgebiete erschlossen werden. Die ausschließliche Verwendung von Open Source Software ermöglicht auch den Austausch mit einer größeren Anzahl an Entwickler und Nutzer. Die Erstellung eines FE-Strukturmodells aus einem CPACS-Datensatz basiert auf dem Grundgedanken, zuerst ein einfaches Basismodell zu generieren, welches bei allen Modellierungszielen (Dimensionierung, Crash,...) gleich ist. Ausgehend von diesem FE-Solver unabhängigen Basismodell kann die Komplexität bei Bedarf modular gesteigert werden (z.B. durch spezifische Modellierungen und Netzanpassungen), um ein detallierteres FE-Modell (z.B. für Crashberechnungen) zu erhalten. Eine kurze Beschreibung der in Entwicklung befindlichen Tools folgt in den Kapiteln 3 bis 3.

Verwaltung von CPACS-XML-Daten

Zum Verwalten des CPACS-Datensatzes, und damit aller Parameter zur Beschreibung des Luftfahrzeugentwurfs, wurde ein objektorientiertes Modul entwickelt (basierend auf dem Python Modul lxml). Neben Möglichkeiten zum Modifizieren, Validieren und Iterieren von CPACS-Daten sind zusätzliche Funktionalitäten integriert, um die Nutzung zu vereinfachen:

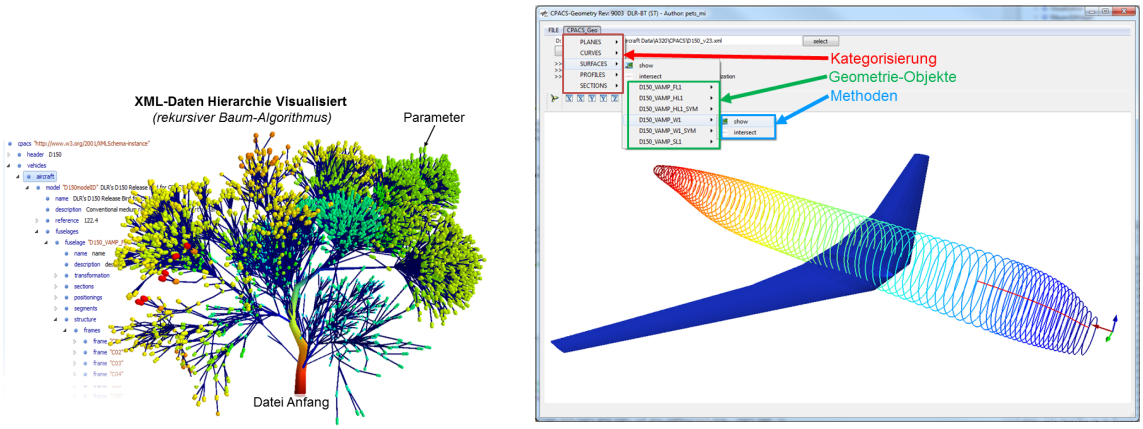
- Automatische Konvertierung von XML-Text zu Skalar- oder Vektordaten.
- Autovervollständigung eines Pfades in der Konsole.
- Setzen von Links zwischen verknüpften Objekten.
- Konvertierung von einheitlichen Unterästen in Tabellenform.

- Visualisieren der Datenstruktur.

Die Visualisierung von CPACS-XML-Daten (siehe Abbildung 3.1a) erfolgt über einen einfachen Baumalgorithmus. Jeder Verzweigung (Ast) des Datensatzes wird rekursiv eine Punkt und Vektor zugewiesen, basierend auf dem übergeordneten Ast, der Anzahl Unteräste sowie weiteren Parametern. Diese Art der Visualisierung kann z.B. zum Hervorheben von Änderungen in CPACS genutzt werden.

Aufbereitung von CPACS-Geometrie

Die Oberflächengeometrie des Luftfahrzeugs ist im CPACS-Datensatz nicht direkt in Form von Flächendefinitionen gegeben sondern wird z.B. über die Interpolation zwischen Stützprofilen beschrieben. Durch weitere Definitionen wie z.B. Schnittebenen mit der Oberfläche werden Referenzen für Strukturkomponenten definiert. Durch diese indirekte Definition der Geometrie lässt sich die Beschreibung des Luftfahrzeugs leichter modifizieren, ist aber auch aufwändiger zu extrahieren und aufzubereiten. Ein bereits am DLR entwickeltes und frei verfügbares Geometrie-Tool TIGL [12], basierend auf dem Geometrie-Kern OpenCascade (OCC) [36], bietet die Möglichkeit die Oberflächengeometrie darzustellen sowie einfache Verschneidungen durchzuführen.
Um die Laufzeit der Toolumgebung bei vielen Geometrieoperationen zu reduzieren, werden aktuell zusätzliche Möglichkeiten zur Geometrienutzung am DLR Institut BT untersucht. So kann durch die Erstellung der Geometrie direkt in Python über pythonOCC [35] der Overhead reduziert werden, während voller Zugriff auf alle OCC-Geometrieoperationen besteht. Eine weitere Möglichkeit besteht in der Nutzung von MESH-Geometrie (diskretes Netz aus Punkten), wofür in Python eigene Algorithmen (basierend auf Modulen wie NumPy, Pandas oder SciPy) entwickelt wurden. Bei variabler Netzgröße der Geometrie kann so die Rechenzeit und Genauigkeit entsprechend den Anforderungen variiert werden. So sind z.B bei aufwändigen Verschnitten Reduktionen der Rechenzeit um Größenordnungen möglich, was bei der Berechnung vieler Geometriekomponente von Vorteil sein kann. Steht dagegen die Geometriequalität (Genauigkeit, Gradienten) im Vordergrund ist die OCC-Geometrie zu bevorzugen. Ein objek-



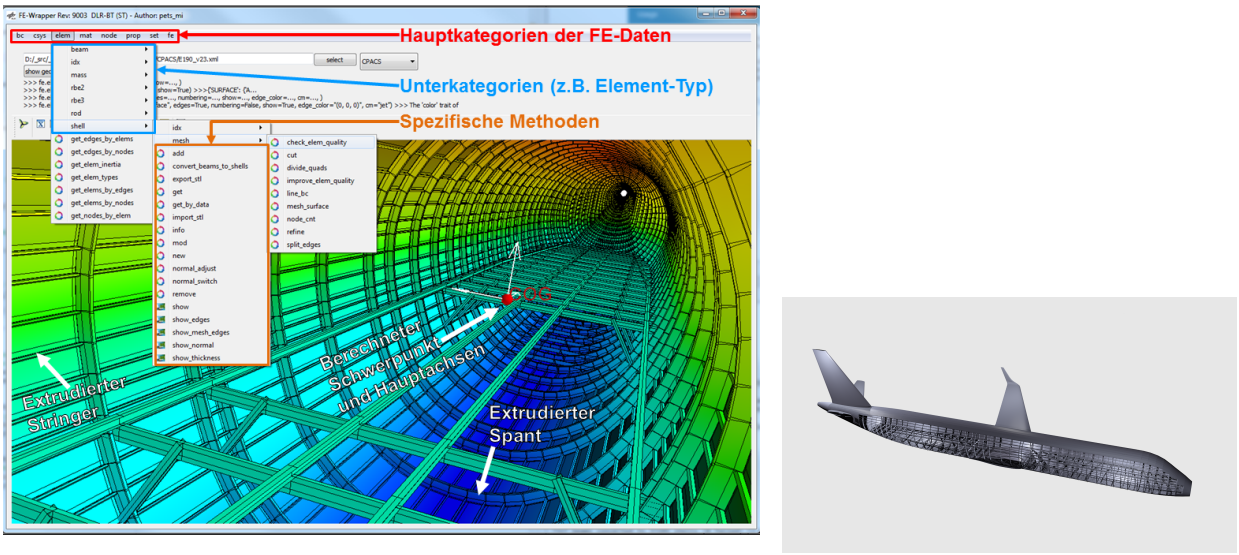
(a) CPACS Hierarchie Visualisierung. **(b)** CPACS Geometrie-Modul GUI.

Abbildung 3.1: Visualisierung von CPACS Daten und Geometrie.

torientiertes Geometrie-Modul zur Verwaltung der in CPACS beschriebenen Geometrie bietet hier die Möglichkeit entweder MESH- oder auch OCC-Geometrie zu nutzen, indem jede Geometrie unabhängig vom Geometrie-Kern und -Typ als einheitliches Geometrie-Objekt mit unterschiedlichen Eigenschaften definiert ist. Geometrieoperationen wie z.B. Verschnitte können so zwischen beliebigen Geometrie-Objekten über eine einzige Methode ausgeführt werden, wobei die benötigten Geometrie-Operationen abhängig vom Geometrie-Kern und -Typ automatisch aufgerufen werden und ein neues Geometrie-Objekt zurückgeliefert wird. In Abbildung 3.1b ist beispielhaft eine GUI des Geometrie-Moduls gezeigt, mit den Möglichkeiten der Kategorisierung der aus CPACS aufbereiteten und zur Verfügung gestellten Geometrie-Objekten.

Verwaltung von FE-Daten

Um einen flexiblen und FE-Solver unabhängigen FE-Modellaufbau zu ermöglichen, wurde ein Tool zur Verwaltung von FE-Daten (FE-PreProzessor) entwickelt. Neben dem Modifizieren von Knoten, Elementen oder Randbedingungen sind auch spezifische Funktionalitäten wie die Konvertierung von vereinfachten Balken- in Schalenelemente implementiert. Weiterhin bestehen Möglichkeiten für verschiedene Visualisierungen (z.B. Elementnormalen), Netzmodifikationen oder Modell-Exporte (z.B. für FE-Solver über das Konverter-Tool oder als STL-Format für CAD- und Rendering-Software). Als Kern des FE-PreProzessors wird das interne FE-Daten-Format des Konverter-Tools (Kapitel 3) genutzt. Neben der skriptbasierten Nutzung des FE-PreProzessors über Python ist auch eine mit PyQt erstellte GUI (siehe Abbildung 3.2a) implementiert, um den Aufbau des Tools zu Visualisieren und einem Anwender die Möglichkeit zu geben ohne Programmierung ein FE-Modell zu erstellen.



(a) FE-PreProzessor GUI. **(b)** Gerendertes Basis-FE-Modell.

Abbildung 3.2: FE-PreProzessor und Basis-FE-Modell.

Modellierung eines Basis-FE-Modells

Das Basis-FE-Modell kann sowohl basierend auf der OCC- als auch der MESH-Geometrie (siehe Kapitel 3) aufgebaut werden. Über die Oberflächengeometrie können je nach benötigter Netzfeinheit sowohl einfachere Verschnittpunkte als auch aufwändigere Verschnittkurven berechnet werden, um die Rumpfstruktur des FE-Basismodells aufzubauen. Dazu zählen luftfahrttypische Strukturkomponenten wie Spante (umlaufende Versteifungen) oder Stringer (Längsversteifungen), welche in Abbildung 3.2a eingezeichnet sind. Des Weiteren werden über CPACS die FE-Eigenschaften wie Materialien oder Hautdicken definiert sowie realitätsnahe Modellrandbedingungen festgelegt. Spezifische Anpassungen des Basismodells an eine reale Luftfahrtstruktur werden über erweiterte Modellierungs-Tools (Kapitel 3) realisiert. In Abbildung 3.2b ist beispielhaft ein gerendertes Basis-FE-Modell mit der freien GPL-lizensierten 3D-Grafiksuite Blender [14] dargestellt.

Modellierung eines detaillierten FE-Modells

Zur erweiterten Ausmodellierung des Basis-FE-Modells zählen z.B. Fahrwerksschächte, Flügelanbindungen oder Druckschotte. Diese Komponenten werden basierend auf dem Basis-FE-Modell über spezifische Definitionen in CPACS sowie angepasste Algorithmen generiert. Für Ausschnitte, Netzverfeinerungen oder Extrusionen können Methoden des FE-PreProzessors verwendet werden. Das detaillierte FE-Modell kann dann für die Strukturdimensionierung oder durch weitere Anpassungen (z.B. der Netzfeinheit) auch für Crashsimulationen verwendet werden. Eine Umsetzung dieser bisher weitesgehend APDL basierten Algorithmen in Python ist momentan in Entwicklung.

Konvertierung von FE-Daten

Der FE-Modell Konverter (Conspyre) ist zentraler Bestandteil der Open Source Toolumgebung und kann sowohl als eigenständiges Tool für die FE-Konvertierung zwischen zwei FE-Solver-Formaten verwendet werden, als auch um ein über die Toolumgebung erstelltes FE-Modell zu exportieren. Damit stellt Conspyre die Schnittstelle zwischen FE-Modellen und kommerziellen sowie Open Source FE-Solvern her und kann durch modulare sowie objektorientierte Programmierung für beliebige FE-Formate erweitert werden. Als Kern dienen Pandas-DataFrames welche gut zur Verwaltung großer Datenmengen geeignet sind. Genutzt werden kann Conspyre sowohl über die Kommandozeile sowie über eine GUI basierend auf PyQt. Der Konvertierungsablauf gliedert sich in drei Hauptebenen:

- Die Parser-Klasse (spezifisches Einlesen und Aufbereiten der FE-Daten).
- Die Model-Klasse (Verwalten und Visualisieren der FE-Daten im einheitlichen Format).
- Die Writer-Klasse (spezifisches Schreiben der eingelesenen FE-Daten).

Die spezifischen Parser- und Writer-Klassen bestehen dabei aus einem gemeinsamen Kern-Modul (enthält die Basisfunktionalitäten) dass über Vererbung um ein Plugin-Modul (enthält dem jeweiligen FE-Format angepasste Methoden) erweitert wird.

Dimensionierung von FE-Modellen

Für die Dimensionierung eines FE-Modelles (z.B. Anpassung der Wandstärke an vorgegebene Lasten) wird ein iterativer Prozess durchlaufen. Eine Iteration besteht dabei aus der Berechnung der Strukturbelastung des FE-Models (über einen FE-Solver), dem Dimensionierungsalgorithmus zur Berechnung der neuen Ausgangswerte (z.B. Hautdicken) sowie der Anpassung des FE-Modells (z.B. über den FE-PreProzessor, Kapitel 3).
Für die Berechnung können dabei mittelfristig beliebige FE-Solver verwendet werden. Die Konvertierung in das passende Format erfolgt über das Konverter-Tool (Kapitel 3). Für die Dimensionierung wurde bisher ein APDL-Skript basiertes Tool (S-BOT+ [37]) verwendet. Zukünftig wird dieses durch Python basierte Dimensionierungsalgorithmen abgelöst. Durch klare Trennung des Algorithmus von der Modellerstellung und -anpassung können auch weitere Optimierungsaufgaben ermöglicht werden. Erste Tests mit Python-Algorithmen zeigen bereits deutliche Laufzeitreduktionen gegenüber bisher verwendeter APDL-Skripte.

Fazit und Ausblick

Der komplette Umbau des Strukturmodellierungs- und Dimensionierungs-Tools der bisher weitestgehend APDL basierten Prozesskette hin zu einer Python basierten Toolumgebung zeigt bereits viele Vorteile durch den modularen und objektorientierten Aufbau. Aspekte der Wartung und Übersichtlichkeit vereinfachen sich so erheblich. Schnittstellen lassen sich leichter implementieren und durch die Einbindung zahlreicher Module (z.B. Numpy, SciPy, Pandas , Mayavi oder PyQt) reduzieren sich die Laufzeiten erheblich und ermöglichen auch die Implementierung von komplexeren Algorithmen sowie von GUIs. Durch das offen zugängliche FE-Daten-Format ergeben sich außerdem viele neue Funktionalitäten, welche z.B. über den FE-PreProzessor (Kapitel 3) oder den FE-Konverter (Kapitel 3) zugänglich sind. Weitere Herausforderungen stellen noch feinere Vernetzungen und Modellierungen dar, um auch detailliertere Crashmodelle generieren zu können. Dafür müssen zukünftig komplexere Vernetzungsalgorithmen sowie Geometriebeschreibungen implementiert werden.

4 Pushing the Frontiers of Information Extraction: A Strategy for Identifying Calls for Action in Texts about War and Conflict

Autor: Katsiaryna Stalpouskaya & Christian Baden

Introduction

Text is one of the key sources of information for social sciences and humanities which, with the rise and development of computational technologies, has been mostly available via digital libraries, archives and websites. It enables researchers to increasingly deal with large scale text corpora that require the use of advanced software tools to process them and extract information. Computational linguistics - a discipline that has emerged on the border of computer science, linguistics and statistics - has achieved certain results in automated text analysis and information extraction, e.g., tools for part-of-speech tagging, grammar parsing, semantic role labelling, sentiment analysis and anaphora resolution have been developed and successfully used in many scientific projects. However, there still exists a gap between technology available and the needs of social sciences: named entity recognizers are incapable of identifying actors, sentiment analysis just provides the overall mood of an expression but is not able to identify the evaluation of information by the utterer, topic modeling tools can only assign a topic to a document, but fall short of measuring its frame.

Calls for action

The current paper addresses these flaws and presents the approach of INFOCORE consortium to overcome this gap. In the focus of INFOCORE's research is the role media plays in violent conflicts and whether it can be used to predict phases of peace and violence. As escalation or de-escalation depends on coordinated social action, media texts calling for specific courses of action play a critical role in understanding the dynamics of violent conflict. To capture what agendas are presented in various kinds of media, we aim to extract calls for action from news coverage - expressions of a request or desire for a specific course of action with the aim of changing the current, improvable or undesirable state, or maintaining the present, desirable state against threatening deterioration. Calls for action consist of three components: a definition of the current state or problem, a motivational component, and a proposed course of action.

Calls for action can be expressed in texts explicitly, using lexical indicators (modal or performative verbs: *"Chad* **has called for** *the humanitarian community to support the government in dealing with the influx of Nigerian refugees"* or *"They* **must** *obey"*), or grammatical markers (imperative sentences: *"***Fight** *them!"*). However, many calls for action are also formulated implicitly, when a piece of text can be interpreted as calling for action only after taking cotext and context into account. It can be achieved by expressing speaker's dissatisfaction (*"[Sb.]* **condemned** *such a motion"*), by using general expressions that something cannot stand (*"Something* **should be done** *about..."*) or rhetorical questions (*"***Can we accept** *such a treatment?"*), or by speaking about desirable states (*"Peace is the only answer"*) [41].

Furthermore, to have the better understanding of the development of the conflict, it is important to know what exactly is being called for. In this study, we distinguish between the following types of calls for action:

- calls for cooperative solutions, such as ceasefire, peace talks, humanitarian help, financial support, etc.: *"Saudi initiative calls for Israeli pullout from all occupied territories in exchange for peace with Arab world"*, *"Eradication of poverty should be the main priority of humanitarian action"*;
- restrictive solutions, such as fighting, aggression, sanctions or prosecution: *"...Fatah and Hamas, which Israel deems a terrorist organizations calling for its destruction"*, *"UN official applauds sentencing of militia leader for war crimes"*, *"The Shahbag campaigners also urged the countrymen to boycott all commercial, industrial, financial and charitable organisations associated with Jamaat"*;
- calls for not doing something: *"We must not lower our guard, at any time, Prime Minister Manuel Valls told Parliament, adding that "serious and very high risks remain"*, *"The Department of State warns U.S. citizens of the risks of travel to eastern Ukraine"*;
- calls to change the current state of affairs with no certain actions specified: *"Now is the time to take action"*.

Thus our classification routine can be broken down into two steps: extracting calls for action in the first, and classifying them in accordance with the developed taxonomy in the second.

Experimental setup

Corpus

In order to automatically extract and classify calls for action from media texts, we have used the advances in machine learning and natural language processing and trained a statistical model to perform these tasks. First of all we have crafted the training corpus by querying online media archives, texts originating from strategic and political communicators as well as social media and downloading documents that met search criteria "war", "violence" or "conflict" published between 01.01 and 01.03.2015. Then the texts were split into sentences and each sentence was labeled as one calling for action or no, and those that are, were classified in accordance with abovementioned taxonomy. The structure of the corpus is presented in Table 4.1. For the current experiments, 80% of corpus was used for training and 20% for testing.

	Total sentences	**Calls for action**	**Valls for action (%)**
Conventional media	2904	955	32.9
Political communication	798	296	37
Strategic communication	418	106	25.4
Social media	880	121	13.8
Total	5000	1478	29

Tabelle 4.1: Number of sentences in total and calls for action within the corpus.

Features

For the first step of classification, we have used three groups of features in our model. First of all, we have used *n*-grams with $n = 1, 2, 3, 4$. For each of the *n*-grams *tf-idf* scores were computed in order to weight up the words that occur seldom but bare more meaning and to account for the document length. The *tf-idf* score can be computed using the following formula:

$$(tf - idf)(t) = \frac{N(t)}{N} * \log\left(\frac{N(s)}{N(s_t)}\right)$$

where $N(t)$ is the number of occurrences of the term t, N – total number of words in the sentence, $N(s)$ – total number of sentences, $N(s_t)$ – total number of sentences where a term t occurs.

Despite being in principal semantic information, calls to act are frequently transmitted by specific grammar relations between words, rather than words themselves. Based on this intuition, we have developed a set of 32 linguistic features that represent the grammar structure of sentences calling for action [1]. The features are mainly checking for the presence of specific 'hint' words in a sentence ("request", "need", "answer", etc.) and disambiguating between examples like *"He* **needs** *to help the refugees"* vs. *"The* **needs** *of all the refugees cannot be met"*, *"They* **request** *a ceasefire"* vs. *"User's* **request** *is being processed"*, *"The tanks* **must be** *withdrawn"* vs. *"It* **must be** *cold"*, *"Peace is* **the only answer***"* vs. "**The only answer** *I have is that I simply didn't know"* by analyzing part-of-speech tags of the 'hint' words, their relationships with other words in the sentence or position within the clause or the sentence.

Finally, data origin, i.e., whether a sentence belongs to traditional or social media, strategic or political communication, has also been used as the third group of features.

Stop-words (functional words as articles, prepositions and auxiliaries) removal and lemmatization or stemming (bringing words to their lemma, i.e., "went" to "go", or stem, i.e., "terrorize" to "terror") – the two common preprocessing steps in computational linguistics – have not been performed as they decreased the performance of the classifier.

[1] The Stanford CoreNLP toolkit [24] has been used for linguistic features extraction.

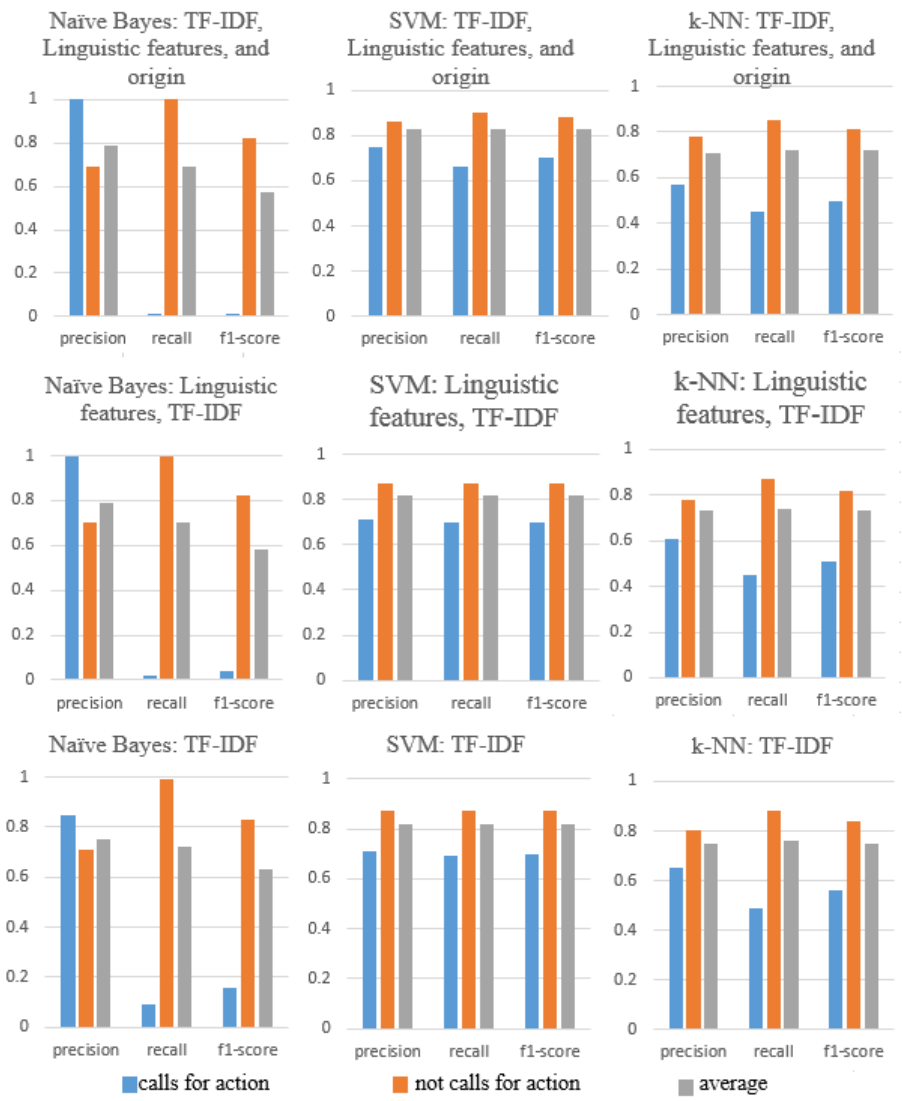


Abbildung 4.1: Performance scores for SVM, Naïve Bayes, and k-Nearest Neighbors classifiers with different sets of features.

Classifiers

Among a great variety of algorithms available, Naïve Bayes, *k*-Nearest Neighbors (kNN) and Support Vector Machines (SVM) have proved to perform the best for text classification tasks [21].The results for three classifiers with different sets of features are shown in Figure 4.1 [2].

Figure 4.1 shows precision, recall and f1-score for three classifiers. The major problem with Naïve Bayes, is that it provides misleadingly high precision for calls for action, but given very low recall, it means that the model captures very few of actual calls for action and most of the sentences that indeed call for something are misclassified. *k*-NN reveals similar behavior with slightly better recall. SVM turns out to be the best performing classifier with reasonably high precision and recall scores for all categories, it also has the highest f1-score which takes into account both precision and recall. SVM reaches an average accuracy of around 80% even without using additional features. The main contribution of linguistic features developed is

[2] The scikit-learn Python library [33] has been used for classification.

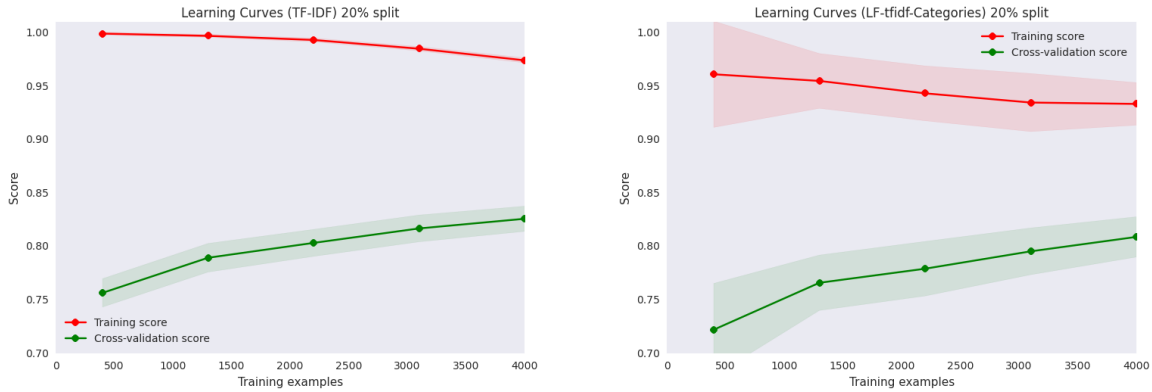


Abbildung 4.2: Learning curves for the classifier using combined features and *tf-idf* only.

that they render the model more powerful by partially overcoming the problem of overfitting, or high variance, which is reflected in learning curves in Figure 4.2:

The learning curves for the model that only uses *tf-idf* scores as features overfits tremendously: the training score remains almost equally high with the increase of the number of training examples, it means that the model overfits training data trying to classify each instance correctly (also outliers and noisy data) rather than finding the optimal general way to separate data and predict labels for new unseen sentences. Generally speaking, a large gap between the curve for the training score and the one for cross-validation score signals about high variance. The learning curves for combined features, in contrast, show less overfitting (the training score decreases with the growth of training set), thus the model is able to generalize better and predict more precisely the labels for the new data, which can be seen from the descending trend of training curve. Learning curves also show, that in order to use *tf-idf* only, more training data is needed to decrease the variance: ideally when training data are enough the scores for training and cross-validation should stay at the same level, the gap between them should be around $0.15 - 0.20$. The model with the combined features is much closer to the ideal state than the one with *tf-idf* only.

Fine-grained classification

In the second round of classification we categorize the sentences that were labelled as calling for action in the first step based on what exactly is being called for. Given the focus on war and violent conflict, we are primarily interested in those calls that can hint at possible developments of a conflict, e.g., escalation or de-escalation. For the current experiment we used two taxonomies. The first one consists of nine categories (see Table 4.2), for the second one we combined some of the categories: calls for de-escalation with calls for support were merged into cooperative treatment, calls for escalation, prosecution and ignorance into restrictive treatment class, and calls for multiple actions, general expressions and other put together into one category. The scores for more detailed and less detailed classifications are shown in Table 4.2 and Table 4.3 respectively.

The scores for the nine-labeled classification are rather low. It can be explained by high ambiguity of natural language, when a lot of contextual and cultural knowledge is needed to correctly assign a label. Even for human coders it appears to be a hard task. Moreover, the

TF-IDF & Data origin	Precision	Recall	f1-score	Support
Peace / De-escalation	0.50	0.62	0.55	34
Violence / Escalation	0.64	0.39	0.48	18
Support	0.36	0.32	0.34	38
Prosecution / Punishment	0.36	0.64	0.46	14
Ignorance / Exclusion	0.40	0.20	0.27	10
General / Rhetorical Q.	0.33	0.17	0.22	18
Calls for Not Doing	0.67	0.35	0.46	23
Multiple	0.22	0.25	0.24	16
Other	0.40	0.53	0.46	51
Average / Total	**0.44**	**0.42**	**0.41**	**222**

Tabelle 4.2: Precision, recall and f1-scores for fine-grained classification for SVM.

TF-IDF & Data origin	Precision	Recall	f1-score	Support
Cooperative Treatment	0.62	0.56	0.58	72
Restrictive Treatment	0.62	0.57	0.59	42
Calls for Not Doing	0.67	0.35	0.46	23
Other	0.57	0.71	0.63	85
Average / Total	**0.60**	**0.59**	**0.59**	**222**

Tabelle 4.3: Precision, recall and f1-scores for 4-labeled classification for SVM.

number of training examples per each category is not enough for the model to make well-weighted decisions. The increase of training data can help the performance which can bee seen from the scores for four-labeled classification.

Conclusion

The present paper mainly addresses two problems. Firstly, it narrows the gap between existing technology, namely the state-of-the-art in text analysis and information extraction, and the needs of social sciences. We have shown how one can extract complex higher-order semantic and pragmatic information highly relevant for communication science research using existing tools and techniques.

Secondly, the core concept of our work, calls for action, can be used in many different fields beyond communication science: extracting calls for action from emails and notes enables automatic generation of to-do lists; identifying recommendations and suggestions in user reviews can help managers or developers to improve their products or services; finally, processing medical documentation and extracting treatment recommendations can help doctors to make decisions on treatment methods in a timely manner.

5 Using Python for Scientific Research

Autor: Dr. Uwe Ziegenhagen

Scientific or statistical research has long been the domain of dedicated programming languages such as R, SPSS or SAS. A few years other competitors entered the arena, among them Python with its powerful SciPy package. The following article introduces SciPy by applying a small subset of its functionality to a well-known dataset.

SciPy

Besides indicating a dedicated collection of numerical algorithms, the term "SciPy" also describes a set of Python core package for scientific research. Relevant in the context of this paper are mainly the following components[1]:

NumPy is the fundament for all kinds of numerical computations as it defines not only the basic data types such as large, multi-dimensional arrays and matrices, but also fundamental operations for these data types.

Matplotlib is the 2D plotting library for the creation of publication quality figures

pandas provides mechanisms and data structures for easier data selection and filtering. Originally developed at AQR Capital Management by Wes McKinney for quantitative analysis of financial data. Since then it has become the defacto standard for data scientific data handling in Python.

Although the necessary packages can be installed manually it is recommended to use a dedicated Python installation, see WinPython (http://winpython.github.io) or Anaconda (http://www.continuum.io).

The "Swiss Banknote" data

This article uses the so-called "Swiss Banknote" data [13]. It consists of seven variables (see figure 5.1) measured for 100 genuine and 100 counterfeit Swiss banknotes:

Length Length of bill in mm

Left Width of left edge in mm

Right Width of right edge in mm

Bottom Bottom margin width in mm

[1] See [40] for information on the additional components.

Top Top margin width in mm

Diagonal Length of image diagonal in mm

Status 0 for genuine, 1 for counterfeit banknote

Abbildung 5.1: A Swiss banknote, source: [13].

Data Loading and Handling

Before one can work with the data it needs to be loaded and checked for obvious errors. As our data is in CSV (comma-separated values) we apply pandas' `read_csv` command to load the data, for data from Excel or even the clipboard pandas provides similar commands. Since the file uses comma as decimal separator and semicolon as column separator the corresponding options need to be set, see Listing 5.1 for the source code.

```
1: import pandas as pd
2: data = pd.read_csv('banknote.csv', sep=';', decimal=',')
```

Listing 5.1: Foo.

With the loaded data pandas creates a so-called "DataFrame" object. `DataFrames` are – beside `Series` – the essential data types in pandas. A `Series` object is a one-dimensional array that can hold any data type (integer, float, string, etc.) labeled with an index. `DataFrames` extend the `Series` by one dimension, thus represent two-dimensional data structures with (potentially, but not necessarily) different data types and a common index.

Exploring the Data

Using the pandas `head()` respectively `tail()` functions the correct application of the formatting instructions be checked, see Listing 5.2. As one can see, 200 data rows have been loaded successfully.

```
IN[5]: data.head()
Out[6]:
     Status  Length   Left  Right  Bottom   Top  Diagonal
0  Genuine   214.8  131.0  131.1     9.0   9.7     141.0
1  Genuine   214.6  129.7  129.7     8.1   9.5     141.7
2  Genuine   214.8  129.7  129.7     8.7   9.6     142.2
3  Genuine   214.8  129.7  129.6     7.5  10.4     142.0
4  Genuine   215.0  129.6  129.7    10.4   7.7     141.8

len(data)
Out[7]: 200
```

Listing 5.2: The `head()` command applied, output grabbed from Spyder Python IDE. All 200 rows have been loaded successfully.

For exploring the data it has become common to use Tukey's "five-number summary" [16] to calculate key indicators for the dataset: minimum, maximum, median as well as the upper and lower quartile. Pandas provides the `describe()` function for this purpose, see Listing 5.3. It shows an excerpt, limiting the variables to `Length`, `Left`, `Right` and `Bottom` using pandas' column filtering mechanisms.

```
IN[1]: summary = data.describe()
IN[2]: summary = summary[['Length', 'Left', 'Right', 'Bottom'
    ]]
IN[3]: summary
Out[4]:
            Length        Left       Right      Bottom
count  200.000000  200.000000  200.000000  200.000000
mean   214.896000  130.121500  129.956500    9.417500
std      0.376554    0.361026    0.404072    1.444603
min    213.800000  129.000000  129.000000    7.200000
25%    214.600000  129.900000  129.700000    8.200000
50%    214.900000  130.200000  130.000000    9.100000
75%    215.100000  130.400000  130.225000   10.600000
max    216.300000  131.000000  131.100000   12.700000
```

Listing 5.3: Summary for the complete data.

A graphical representation of the five-number summary can be created with boxplots. To create a boxplot with the given data we use the seaborn library [43], which is built on top of `matplotlib` to simplify the creation of print-ready plots. Listing 5.4 shows the command, figure 5.2 the resulting diagram.

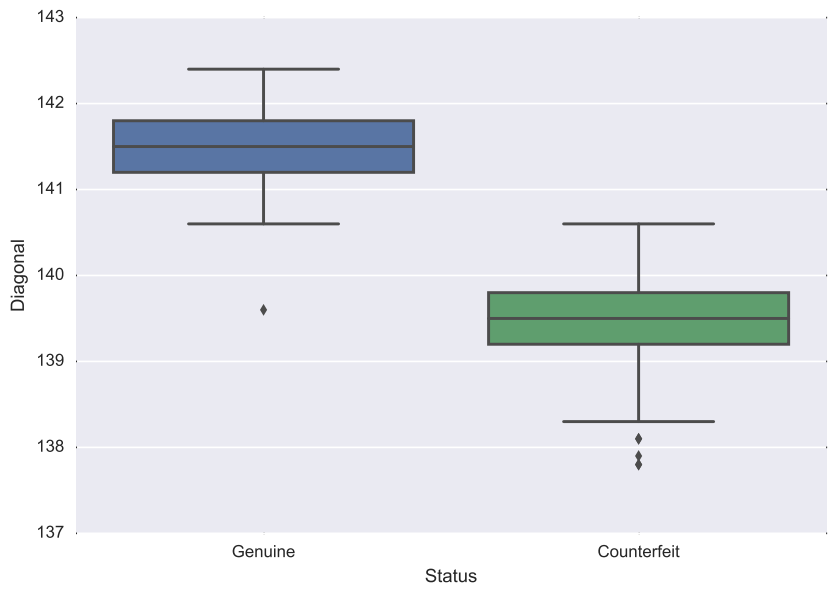


Abbildung 5.2: Boxplot for "Diagonal", grouped by 'Status'.

```
IN[1]: sns.boxplot(x="Status", y="Diagonal", data=data);
```

Listing 5.4: Code to create a grouped boxplot for the variable Diagonal.

Another basic method of graphically exploring the data is to use scatter plots, respectively scatter plot matrices which contain multiple plots. For space reasons the data was restricted to just three variables plus 'Status' as grouping variable. See Listing 5.5 for the code and figure 5.3 for the resulting images, which shows all variables plotted against all variables, with histograms on the diagonals. The scatterplots indicate, that the variable 'Diagonal' might provide a suitable criterium to separate genuine from counterfeit banknote while the other variables do not seem powerful enough to provide such separation, as they show genuine and counterfeit banknotes mixed in point clouds. The boxplot, see figure 5.2, confirms the suitability of the variable 'Diagonal', as it clearly shows that the diagonal length of genuine banknotes is significantly longer than the diagonal length of counterfeit banknotes, with only one exception.

```
IN[1]: scatterdata = data[['Status','Length', 'Left',
       'Diagonal']]
IN[2]: scatter = sns.pairplot(scatterdata, hue="Status")
```

Listing 5.5: Code to create a scatter plot matrix.

Cluster Analysis

As final step in the analysis a simple cluster analysis shall be performed. It is beyond the scope of this paper to give an introduction of this topic – interested readers may consult [20] or [44] – but cluster analysis tries to find groups of "similar" objects in the data. There are

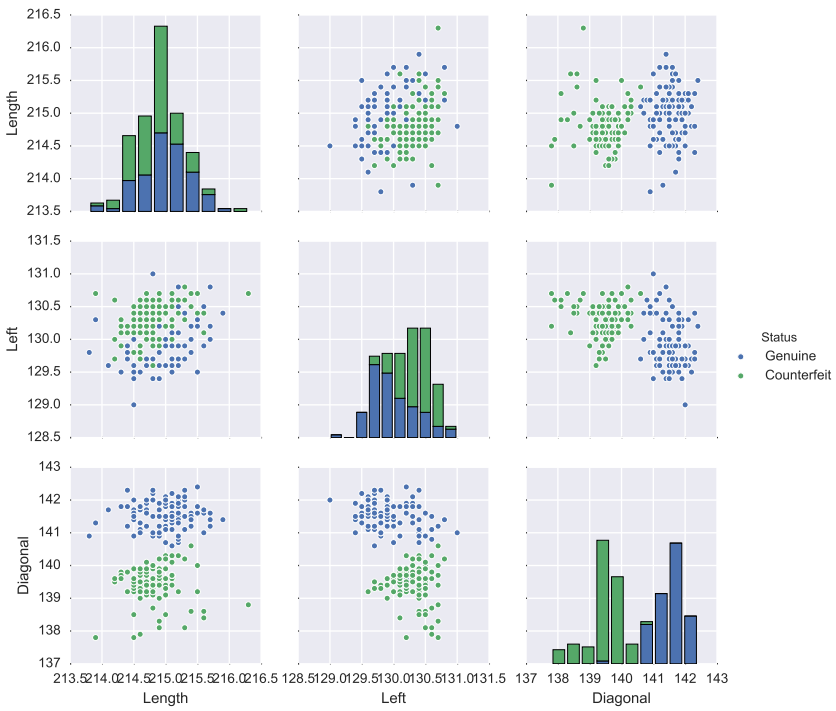


Abbildung 5.3: Scatterplot Matrix for three variables, grouped by 'Status'.

several hundred different algorithms differing e. g. in the way, they determine how they define "similarity". In this paper the K-means clustering algorithm will be used as it is rather simple to explain and work with.

K-means in its most basic form requires the data scientist to define the number K as the number of groups in the data. As for this example two groups – genuine and counterfeit – are relevant, K is set to 2. Two random objects are now selected from the data, these will represent the cluster centers in the initial step. For each of the remaining data point the distances, often the Euclidian distance, between the data point and both cluster centers is calculated. The data points are then assigned to the cluster point for which the calculated distance is minimal.

After the assignment of all data points the new clusters are calculated as (arithmetic) means of the respective cluster centers and the procedures is repeated until the algorithm converges, which means that no reassignments to the other cluster occur respectively the cluster centers do not move anymore.

SciPy provides an own package, `scipy.cluster.vq` for cluster analysis. Listing 5.6 shows an excerpt for the required cluster analysis code, figure 5.4 the resulting scatter plot. Further analysis, by comparing the original status with the computed cluster assignment, revealed one observation to be wrongly assigned.

```
from scipy.cluster.vq import kmeans,vq

data = data[['Length', 'Diagonal']]
# convert to matrix for cluster analysis
clusterData = data.as_matrix()
```

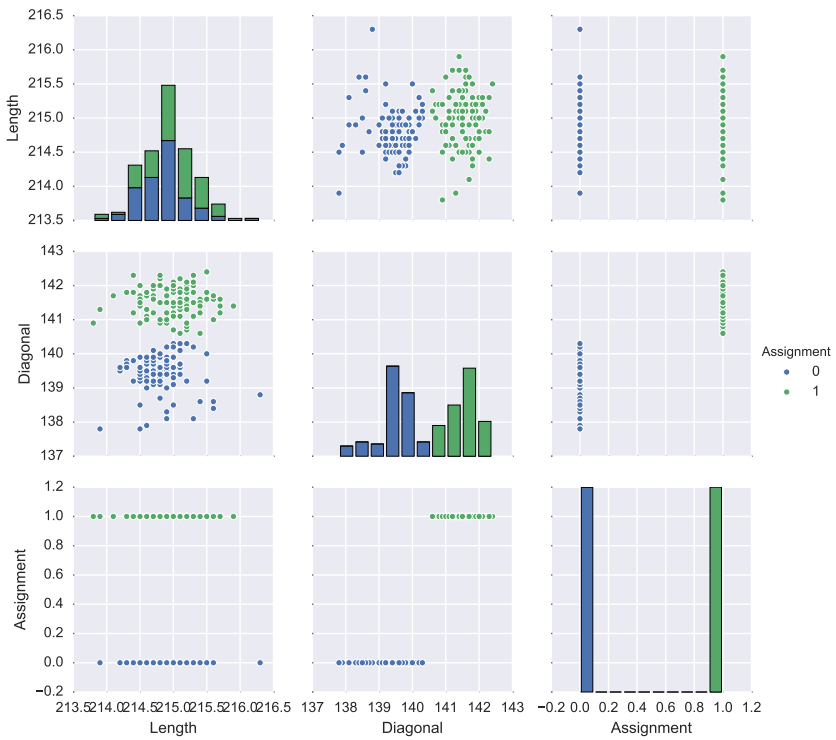


Abbildung 5.4: Scatterplot, color indicates the assignment to a specific cluster.

```

# compute K-Means with K = 2 clusters
centroids,_ = kmeans(data,2)
# assign each observation to a specific cluster
assignment,_ = vq(data,centroids)

# save assignment in original data
data['Assignment'] = assignment

# create scatter plot with assignment
scatter = sns.pairplot(data, hue='Assignment')
```

Listing 5.6: Excerpt from cluster analysis code based on two variables.

Conclusion

With the SciPy packages Python offers a versatile toolbox to efficiently analyze data in all its aspects and thus may present an excellent alternative to existing programming languages. This paper could only utilize a very small subset of the SciPy functionality, interested readers are invited to study the abundant documentation on- and offline.

Literaturverzeichnis

[1] Jake VanderPlas et al. *gatspy: General tools for Astronomical Time Series in Python*. URL: http://www.astroml.org/gatspy/ (besucht am 14. 08. 2016) (zitiert auf Seite 5).

[2] Jake VanderPlas et al. *Machine Learning and Data Mining for Astronomy*. URL: http://www.astroml.org/ (besucht am 14. 08. 2016) (zitiert auf Seite 5).

[3] *Aptly*. 2016. URL: https://www.aptly.info/ (besucht am 01. 08. 2016).

[4] Astropy. *A Community Python Library for Astronomy*. URL: http://astropy.org/ (besucht am 14. 08. 2016) (zitiert auf Seite 4).

[5] Ben Caldwell, Michael Cooper, Loretta Guarino Reid und Gregg Vanderheiden. *Web Content Accessibility Guidelines (WCAG) 2.0*. Dez. 2008. URL: http://www.w3.org/TR/WCAG20/.

[6] *CENDARI GitHub Organisation*. 2016. URL: https://github.com/cendari/ (besucht am 01. 08. 2016).

[7] European Commision. *What are RIs?* 2016. URL: http://ec.europa.eu/research/infrastructures/index_en.cfm?pg=what (besucht am 01. 08. 2016).

[8] *D21-Digital-Index 2015*. 2015. URL: http://www.initiatived21.de/portfolio/d21-digital-index-2015/.

[9] DLR. *Was passiert, wenn Sterne Verstecken spielen?* 2009. URL: http://www.dlr.de/desktopdefault.aspx/tabid-5170/8702_read-20474/ (besucht am 14. 08. 2016) (zitiert auf Seite 2).

[10] DLR-LY. *Common Parametric Aircraft Configuration Schema (CPACS)*. 2016. URL: https://github.com/DLR-LY/CPACS (besucht am 15. 08. 2016) (zitiert auf Seite 7).

[11] DLR-SC. *Remote Component Environment*. 2016. URL: http://www.dlr.de/sc/desktopdefault.aspx/tabid-5625/9170_read-17513/ (besucht am 18. 08. 2016) (zitiert auf Seite 7).

[12] DLR-SC. *The TiGL Geometry Library to process aircraft geometries in pre-design*. 2016. URL: https://github.com/DLR-SC/tigl (besucht am 15. 08. 2016) (zitiert auf Seite 9).

[13] Flury und Riedwyl. *Multivariate Statistics. A Practical Approach*. 1. Aufl. Chapman und Hall, 1988 (zitiert auf Seiten 19, 20).

[14] Blender Foundation. *Blender 3D-Grafiksuite*. 2016. URL: https://www.blender.org/ (besucht am 18. 08. 2016) (zitiert auf Seite 11).

[15] Kelly A. Harper und Jamie DeWaters. "A Quest for website accessibility in higher education institutions". In: *The Internet and Higher Education* 11.3–4 (2008). Special Section of the {AERA} Education and World Wide Web Special Interest Group (EdWeb/SIG), S. 160 –164. ISSN: 1096-7516. DOI: http://dx.doi.org/10.1016/j.iheduc.2008.06.007.

[16] David C. Hoaglin, Frederick Mosteller und John W. Tukey. *Understanding robust and exploratory data analysis*. 1. Aufl. Wiley, 1983 (zitiert auf Seite 21).

[17] Wolfgang Karl Härdle und Léopold Simar. *Applied Multivariate Statistical Analysis*. 4. Aufl. Springer, 2015.

[18] Ansys Inc. *ANSYS FE-Software*. 2016. URL: http://www.ansys.com/ (besucht am 15. 08. 2016) (zitiert auf Seite 7).

[19] *Jenkins*. 2016. URL: https://jenkins.io/ (besucht am 01. 08. 2016).

[20] Leonard Kaufman und Peter Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. 1. Aufl. Wiley, 1990 (zitiert auf Seite 22).

[21] Anthony Khoo, Yuval Marom und David Albrecht. "Experiments with Sentence Classification". In: *Proceedings of the 2006 Australasian language technology workshop* (2006), S. 18–25 (zitiert auf Seite 16).

[22] Gene Kim, Kevin Behr und George Spafford. *The Phoenix Project: A Novel About IT, DevOps, and Helping Your Business Win*. 1st. IT Revolution Press, 2013. ISBN: 9780988262591.

[23] scikit learn. *Machine Learning in Python*. URL: http://scikit-learn.org/ (besucht am 14. 08. 2016) (zitiert auf Seite 4).

[24] Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard u. a. "The Stanford CoreNLP Natural Language Processing Toolkit". In: *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. 2014, S. 55–60 (zitiert auf Seite 15).

[25] Wes McKinney. *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython*. 1. Aufl. O'Reilly, 2012.

[26] *Mobile Internetnutzung – Gradmesser für die digitale Gesellschaft*. 2014. URL: http://www.initiatived21.de/portfolio/mobile-internetnutzung-2014/.

[27] Christopher C. Morphew. "What College and University Websites Reveal About the Purposes of Higher Education". In: *The Journal of Higher Education* 85.4 (2014), S. 499–530. ISSN: 1538-4640. URL: http://muse.jhu.edu/journals/journal_of_higher_education/v085/85.4.saichaie.html.

[28] Vorname Nachname. *Titel der Webseite*. 2014. URL: http://some.url/page (besucht am 13. 10. 2014).

[29] B. Nagel, D. Böhnke, V. Gollnick, P. Schmollgruber, A. Rizzi u. a. *Communication in Aircraft Design: Can We Establish a Common Language?* Brisbane (Australia): 28th Congress of the International Council of the Aeronautical Sciences (ICAS), 2012 (zitiert auf Seite 7).

[30] OGLE. *OGLE-III Catalog of Variable Stars*. URL: http://ogledb.astrouw.edu.pl/~ogle/CVS/ (besucht am 14. 08. 2016) (zitiert auf Seite 3).

[31] OGLE. *The Optical Gravitational Lensing Experiment*. URL: http://ogle.astrouw.edu.pl/ (besucht am 14. 08. 2016) (zitiert auf Seite 3).

[32] pandas. *Python Data Analysis Library*. URL: http://pandas.pydata.org/ (besucht am 14. 08. 2016) (zitiert auf Seite 5).

[33] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion u. a. "Scikit-learn: Machine Learning in Python". In: *JMLR* 12 (2011), S. 2825–2830 (zitiert auf Seite 16).

[34] *Puppet*. 2016. URL: https://puppet.com/ (besucht am 01. 08. 2016).

[35] Open Cascade S.A.S. *OCC Api für Python*. 2016. URL: http://www.pythonocc.org/ (besucht am 15. 08. 2016) (zitiert auf Seite 9).

[36] Open Cascade S.A.S. *Open CASCADE Technology*. 2016. URL: https://www.opencascade.com (besucht am 15. 08. 2016) (zitiert auf Seite 9).

[37] J. Scherer, D. Kohlgrüber, F. Dorbath und M. Sorour. *Finite element based Tool Chain for Sizing of Transport Aircraft in the Preliminary Aircraft Design Phase*. Stuttgart (Germany): 62. DLRK, 2013 (zitiert auf Seiten 7, 12).

[38] J. Scherer und D. Kohlgrüber. *Overview of the versatile options to define fuselage structures within the CPACS data format*. Aachen (Germany): 4th EASN Workshop on Flight Physics & Aircraft Design, 2014 (zitiert auf Seite 7).

[39] SciPy. URL: https://scipy.org/ (besucht am 14. 08. 2016) (zitiert auf Seite 4).

[40] SciPy. *Scientific Computing Tools for Python*. 2016. URL: https://www.scipy.org/about.html (besucht am 15. 08. 2016) (zitiert auf Seite 19).

[41] Katsiaryna Stalpouskaya und Christian Baden. "To Do or Not to Do: the Role of Agendas for Action in Analyzing News Coverage of Violent Conflict". In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*. Hrsg. von Min-Yen Kan. 2015, S. 21–30. ISBN: 978-1-941643-71-6 (zitiert auf Seite 14).

[42] Statsmodels. *Statistical methods and data exploration for Python*. URL: http://statsmodels.sourceforge.net/ (besucht am 14. 08. 2016) (zitiert auf Seite 5).

[43] Michael Waskom. *Seaborn: statistical data visualization*. 2016. URL: https://stanford.edu/~mwaskom/software/seaborn/ (besucht am 18. 08. 2016) (zitiert auf Seite 21).

[44] Ian Witten und Eibe Frank. *Data Mining. Practical Machine Learning Tools and Techniques*. 1. Aufl. Morgan Kaufman, 2005 (zitiert auf Seite 22).

[45] Panayiotis Zaphiris und R Darin Ellis. "Website Usability and Content Accessibility of the top USA Universities." In: *WebNet*. 2001, S. 1380–1385.