Visualisierungsinstitut (VISUS)

Universität Stuttgart
Allmandring 19
D–70569 Stuttgart

Bachelorarbeit Nr. 263

Ein adaptierbares Rahmenwerk für die Annotation von Skalar- und Vektorfelddaten

Marian Thull

Studiengang:	Informatik
Prüfer/in:	Prof. Dr. Daniel Weiskopf
Betreuer/in:	Dipl.-Inf. Rudolf Netzel
Beginn am:	1. Juni 2015
Beendet am:	4. November 2015
CR-Nummer:	I.2.6, I.4.8, I.5.5

Kurzfassung

Die manuelle Annotation von Skalar- und Vektorfelddaten zum Zwecke des überwachten maschinellen Lernens bedeutet einen hohen zeitlichen Aufwand. Zusätzlich verursachen die heute gängigen rechteckigen Selektionsregionen Ungenauigkeiten. Als Reaktion darauf wird ein System vorgestellt, das die Annotation mittels allgemeiner polygonaler Regionen ermöglicht. Es bietet die Möglichkeit, die Visualisierung der Skalar- bzw. Vektorfelder flexibel zu wechseln. Dazu wird ein entsprechendes Plugin-System realisiert. Ebenso ist es möglich, die Berechnungsmethode der Merkmalsvektoren schnell und einfach durch Plugins auszutauschen. Das Rahmenwerk unterstützt die Verwaltung von Annotationsprojekten. In Kombination mit dem Plugin-System für die Visualisierung der zu annotierenden Daten und die Generierung der Merkmalsvektoren ist ein flexibles und leistungsfähiges Rahmenwerk entstanden. Als theoretische Basis werden in dieser Arbeit einige maschinelle Lernverfahren und ihre Evaluation, Grundlagen der Merkmalsvektorkonstruktion und die Vektorfeldvisualisierung mit Line Integral Convolution eingeführt. Darauf folgt eine Beschreibung des entstandenen Systems und seine Auswertung, die den Vorteil der polygonalen Regionen gegenüber den Rechtecken belegen kann. Zum Schluss wird ein Ausblick auf mögliche Verbesserungen des Rahmenwerks gegeben.

Abstract

Manual annotation of scalar and vector field data for supervised machine learning causes a large temporal effort. Additionally, the rectangular regions which are popular for selection today are responsible for inaccuracies. As a reaction a system is introduced that enables annotation with arbitrary polygonal regions. It offers the option to flexibly substitute the visualization of the scalar or vector field. Therefor a corresponding plugin system is realized. Likewise it is possible to substitute the method of feature vector calculation fast and easily through plugins. The framework supports the management of annotation projects. In combination with the plugin system for data visualization and feature vector calculation a flexible and powerful was developed. As a theoretical base, several machine learning algorithms and their evaluation as well as the foundations of feature vector engineering and vector field visualization with line integral convolution are introduced. This is followed by a description of the system and its evaluation, which can verify the advantage of the polygonal regions over the rectangles. Finally an outlook on possible improvements of the framework is given.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Listings

Verzeichnis der Algorithmen

1 Einleitung

Seit den Anfängen der Forschung auf dem Gebiet der künstlichen Intelligenz wird auch versucht, Maschinen das Lernen beizubringen. Bis heute wurden zu diesem Zweck zahlreiche verschiedene Algorithmen und Methoden entwickelt, die zunehmend beeindruckende Leistungen aufweisen können.
Um die Repräsentation realer Objekte zu lernen, benötigen viele Algorithmen annotierte Daten. Das bedeutet, dass die Daten als Paare aus Objektbeschreibung und Klassenzuweisung vorliegen müssen.
Handelt es sich bei den Daten um Skalar- oder Vektorfelddaten, stellt sich die Aufgabe oft wie folgt: Aus den Datensätzen müssen Regionen ausgewählt werden, die einer bestimmten Objektklasse angehören, als Beispiel seien hier Wirbel genannt. Diese müssen in eine für den Algorithmus verwendbare Form übertragen werden, in der Regel sind dies numerische Merkmalsvektoren. Eine große Zahl solcher Merkmalsvektoren in Verbindung mit ihrer jeweiligen Klasse kann schließlich zum Training eines Klassifikators verwendet werden.
Diese Aufgabe birgt drei Probleme, die es zu lösen gilt:

- Die gewöhnlich im Rahmen solcher Anwendungen verwendeten rechteckigen Selektoren sind für komplexere Formen zu ungenau.
- Da die Menge an verfügbaren Trainingsdaten oft einen größeren Einfluss auf die Leistung eines Klassifikators hat als die Qualität des Algorithmus, wird ein immenser zeitlicher Aufwand für die Annotation nötig.
- Die Entwicklung geeigneter Merkmalsvektoren ist kompliziert und daher an häufiges Testen gebunden. Auch hier liegt ein großer Zeitaufwand vor.

Diese Arbeit beschreibt ein Rahmenwerk zur Selektion und Klassenzuweisung von Regionen in Skalar- und Vektorfelddaten und zur Berechnung von auf diesen Regionen basierenden Merkmalsvektorausgabedateien. Um die Ungenauigkeit der Rechtecke zu vermeiden bietet das System die Möglichkeit, die Daten unter Verwendung beliebiger allgemeiner Polygone zu selektieren. Auch runde oder nicht konvexe Objekte können so mit ausreichender Genauigkeit ausgewählt werden.

Um die Zeit, welche das Annotieren in Anspruch nimmt, möglichst klein zu halten, wird für das Rahmenwerk ein Fokus auf Effizienz gelegt. Die Benutzung soll möglichst einfach erfolgen. Sämtliche von der Datenart abhängigen Module des Programms sind über einfaches

Laden von Plugins austauschbar.
Ein solches Plugin-System wird auch für die Berechnung des Merkmalsvektors verwendet. Die für das Testen verschiedener Vektoren benötigte Zeit wird so deutlich reduziert.

Die Arbeit gibt einen Einblick in die Grundlagen des maschinellen Lernens, um die Notwendigkeit geeigneter Trainingsdaten deutlich zu machen und die theoretischen Grundlagen für die am Ende erfolgte Auswertung zur Verfügung zu stellen. Es folgt eine Einführung in die Konstruktion aussagekräftiger Merkmalsvektoren für die Datenrepräsentation. Außerdem werden die Visualisierung von Vektorfelddaten mittels Line Integral Convolution, sowie Auswertungsmethoden für Klassifikatoren besprochen.

Das System wird schließlich anhand einer Systembeschreibung und einer Fallstudie vorgestellt. Die dabei entstandenen Ergebnisse werden analysiert und verglichen. Als Beispielanwendung orientiert sich die Arbeit an der Annotation von Wirbeln in zweidimensionalen Vektorfeldern.

Gliederung

Die Arbeit ist in folgender Weise gegliedert:

Kapitel 2 – Verwandte Arbeiten Dieses Kapitel gibt einen Überblick über Arbeiten, die auf dem Feld der Annotation von Skalar- und Vektorfeldern, sowie im Bereich der Konstruktion von Merkmalsvektoren, veröffentlicht wurden und ordnet das entstandene System ein.

Kapitel 3 – Grundlagen In diesem Kapitel werden die Grundlagen des maschinellen Lernens, die Konstruktion verschiedener Merkmalsvektoren, die Strömungsvisualisierung mit Hilfe der Line Integral Convolution und die Evaluation der Leistungsfähigkeit von Klassifikatoren erklärt.

Kapitel 4 – Systembeschreibung Hier findet eine Anforderungsanalyse statt und die Implementierung des Systems wird besprochen. Dabei wird insbesondere auch auf das entwickelte Plugin-System und die verwendeten Merkmalsvektoren eingegangen.

Kapitel 5 – Fallstudie In diesem Teil der Arbeit wird die Benutzung des System anhand einer Fallstudie exemplarisch dargestellt. Die dabei gewonnenen Ergebnisse werden außerdem verwendet, um die benutzten Merkmalsvektoren sowie die Regionsselektion mit allgemeinen Polygonen im Vergleich zu Rechtecken zu bewerten.

Kapitel 6 – Zusammenfassung und Ausblick Das letzte Kapitel fasst die Arbeit zusammen und gibt einen Ausblick auf weitere Verbesserungsmöglichkeiten.

2 Verwandte Arbeiten

Die Annotation von Regionen in Skalar- oder Vektorfelddaten, sowie die Verwendung der daraus gewonnenen Ergebnisse für Methoden des maschinellen Lernens, ist Gegenstand einiger Forschungsarbeiten und Projekte.
So existieren zahlreiche Rahmenwerke für die Annotation von Videodaten für Computer Vision und Semantic Web Anwendungen. Das Open Video Annotation Project [RDMN15], ViPER [MD] und ANVIL [Kip12] sind einige populäre Beispiele.
Die am häufigsten verwendete Form zur Selektion von Daten ist das Rechteck. Ein Projekt, das dagegen von polygonalen Regionen für die Selektion von Daten Gebrauch macht, ist das Bildannotationswerkzeug LabelMe [RTMF08]. Es wurde für die Annotation großer Mengen von Bildern entworfen, um eine Datenbank für die Forschung im Bereich Computer Vision zu erstellen. Die Annotation übernehmen Freiwillige in einem Online-Interface. Die Annotation von Vektorfelddaten wird durch das Projekt nicht unterstützt.
Yang et al. beschäftigen sich ebenfalls mit der Klassifikation von Bildregionen. Sie schlagen eine Variante von Support Vector Machines vor, die mit annotierten Regionen arbeitet.
Mit der Detektion von Wirbeln in Vektorfelddaten beschäftigen sich Zhang et al. [ZDM+14]. Sie entwickeln ein Boosting-Verfahren, um die Leistungsfähigkeit der Klassifikation zu steigern. Die Annotation der Regionen erfolgt durch einen Experten.
Mit Wirbeln beschäftigen sich auch Daniels II et al. in [DANS10]. Sie entwerfen ein System für die interaktive Identifikation geeigneter Merkmale durch den Nutzer. Zum Einsatz kommt dafür eine texturbasierte Visualisierung, die Abbildung geschieht hier aber bereits in einem geclusterten Attributsraum. Die verwendeten Daten sind dreidimensionale Vektorfelddaten. In ihrer Arbeit beschäftigen sich die Autoren außerdem mit Merkmalsdesign für die Klassifizierung von Wirbeln.

Die vorliegende Arbeit überträgt die Flexibilität und einfache Benutzbarkeit der vorhandenen Annotationswerkzeuge für Videos und Bilder auf die Domäne der Vektorfelddaten, um die Erstellung einer ausreichend großen Datenbasis für das Training von maschinellen Lernverfahren zu erleichtern. Dabei sind, wie für Bilddaten bereits geschehen, polygonale Regionen selektierbar.

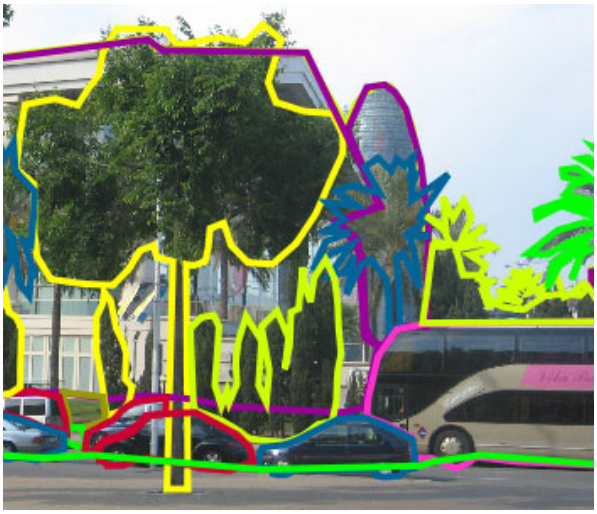

Abbildung 2.1: Regionsauswahl in LabelMe. Quelle: http://labelme.csail.mit.edu/

3 Grundlagen

In diesem Kapitel werden die Algorithmen und Konzepte, die der entwickelten Anwendung und der Auswertung zugrunde liegen, erläutert. Abschnitt 3.1 bietet eine Einführung in die Methoden des maschinellen Lernens und stellt einige dafür wichtige Algorithmen vor. Abschnitt 3.2 beschäftigt sich mit der Konstruktion von Merkmalsvektoren für eine aussagekräftige Repräsentation von Daten, die von maschinellen Lernverfahren genutzt werden kann. Im darauf folgenden Abschnitt 3.3 wird die Visualisierung von Vektorfelddaten besprochen, die für das Annotieren solcher Daten unerlässlich ist. Schließlich widmet sich Abschnitt 3.4 der Auswertung von Lernverfahren. Die darin vorgestellten Konzepte kommen in Kapitel 5 bei der Bewertung zum Einsatz.

3.1 Maschinelles Lernen

Ein System, das lernt, sei es ein Lebewesen oder eine Maschine, muss sehr allgemein gesprochen, im Stande sein, aus Erfahrungen Wissen zu gewinnen. Dieses Wissen wird dann eingesetzt, um Vorhersagen über die Zukunft zu machen, oder die Leistung des Systems in Konfrontation mit einer neuen Situation zu verbessern.
Im Falle des maschinellen Lernens bedeutet das meist, aus einem gegebenen Trainingsdatensatz ein Modell für die Daten zu ermitteln. Ein solches hat die Form einer Funktion, die von Ein- auf Ausgabedaten abbildet. Ziel ist dabei eine gute Generalisierung. Diese zeichnet sich dadurch aus, dass das Modell sowohl für die vorhandenen Trainingsdaten, als auch für zukünftige, beim Lernen unbekannte Testdaten einen möglichst kleinen Fehler bei der Vorhersage der Ausgabe macht.

Unter dem Oberbegriff „Maschinelles Lernen“ werden einige verschiedene Lernszenarien zusammengefasst. Eine wichtige Abgrenzung der Szenarien stellen hier die vorhandenen Ausgabedaten dar.

Beim **unüberwachten Lernen** liegen die verfügbaren Daten ohne Ausgabewerte vor: $D = \{\vec{x}_i\}_{i=1}^{N}$. Das Ziel bei dieser Form des Lernens ist oft das Clustering der Daten (Einteilung in Gruppen mit ähnlichen Eigenschaften) oder das Erlernen einer impliziten Repräsentation in einer niedrigeren Dimension, beispielsweise mit dem Ziel der Kompression von Daten.

Im Gegensatz dazu liegen beim **überwachten Lernen** die Trainingsdaten mit dazugehörigen Ausgabewerten vor: $D = \{\vec{x}_i, y_i\}_{i=1}^{N}$. Die Ausgabe kann dabei kontinuierlich oder diskret sein. Zielsetzung ist hier für gewöhnlich das Lernen einer Funktion, die Eingabewerte auf eine Ausgabe abbildet. Im Falle von diskreten Ausgabewerten spricht man hier auch von Klassifizierung.
Überwachtes Lernen beinhaltet immer auch die Gefahr von Überanpassung. Trainingsdaten spiegeln in den meisten Fällen nicht genau die Funktion wider, die die Daten erzeugt hat, sondern sind durch Rauschsignale gestört. Ein überwachtes Lernverfahren läuft Gefahr, eben diese Störsignale mit zu lernen. Dies wird bei einem Großteil der vorhandenen Algorithmen verhindert, indem die Komplexität des gelernten Modells beschränkt oder eine hohe Komplexität bestraft wird (oft Regularisierung genannt).

Ein dritter Fall ist das **halb überwachte Lernen**, bei dem ein kleiner Teil der Daten mit (unter Umständen unsicheren) Ausgabewerten vorliegt, ein Großteil jedoch ohne. Ausgangslage ist oft eine eigentlich ungenügende Menge an gegebenen Ausgabewerten, zum Beispiel wegen zu hohen Aufwands der Erstellung solcher Datensätze. Unter Verwendung von zusätzlichen Daten ohne Ausgabewerte kann das erzielte Ergebnis oft verbessert werden.

Anwendungen des maschinellen Lernens finden sich bei Regierungsorganisationen [Hic13], in verschiedenen Bereichen der Wirtschaft [Var14] sowie in Forschung und Wissenschaft [MD01]. Beispiele reichen von Gesichts- und Spracherkennung über Spam Filter bis hin zum Bildverstehen.

In dieser Arbeit kommt maschinelles Lernen in Form der Klassifizierung zum Einsatz. Das heißt, die Trainingsdaten liegen als

$$D = \{\vec{x}_i, y_i\}_{i=1}^{N};\ \vec{x}_i \in \mathbb{R}^n;\ y_i \in Y \subset \mathbb{N}$$

vor, wobei Y die Menge aller Klassen ist. Zu erlernen ist dabei in der Regel eine Funktion der Form

$$F:\ \mathbb{R}^n \to Y.$$

Da die meisten Algorithmen nicht direkt mit der Eingabe $\vec{x}$ arbeiten, sondern die Daten mit Hilfe eines Merkmalsvektors repräsentieren (siehe auch Abschnitt 3.2), wird beispielsweise eine Funktion ϕ definiert, die den Eingabevektor in einen Merkmalsraum der Dimension k abbildet:

$$\phi : \mathbb{R}^n \to \mathbb{R}^k$$

Im Folgenden sind einige wichtige Algorithmen des maschinellen Lernens beschrieben. Die **logistische Regression** ist eine Anpassung gängiger Regressionsverfahren an das Problem der Klassifizierung. **Support Vector Machines** (SVMs) sind sogenannte „Large Margin Classifier“, das heißt, sie teilen den Merkmalsraum anhand einer Entscheidungsgrenze, so dass jedes Beispiel des Trainingsdatensatzes einen möglichst hohen Abstand zu der Entscheidungsgrenze hat. **Entscheidungsbäume** lernen eine Reihe von Regeln, um Daten zu

klassifizieren.
Meta-Lernverfahren wie **Boosting** oder **Bagging** können die Ergebnisse vorhandener Lernalgorithmen weiter verbessern, sie werden am Ende des Abschnitts behandelt.

3.1.1 Logistische Regression

Die Logistische Regression ist ein statistisches Analyseverfahren, das verwendet wird, um eine Klassifizierungsfunktion zu lernen. Sie ist bei geeigneter Wahl des Merkmalsvektors sehr leistungsfähig, da zum Beispiel auch nichtlineare Merkmale verwendet werden können. Der Einfachheit halber wird hier nur der binäre Fall mit zwei Klassen betrachtet, die Trainingsdaten liegen also in der Form

$$D = \{\vec{x}_i, y_i\}_{i=1}^{N};\ y_i \in \{-1, 1\}$$

vor. Dementsprechend ist die zu lernende Funktion

$$F:\ \mathbb{R}^n \to \{-1, 1\}.$$

Für die Regression wird F durch eine reellwertige Diskriminanzfunktion

$$f:\ \mathbb{R}^n \times \{-1, 1\} \to \mathbb{R}$$

dargestellt, die folgende Bedingung erfüllen muss:

$$F:\ \vec{x} \mapsto \operatorname*{argmax}_y\ f(\vec{x}, y)$$

Das impliziert, dass f einen hohen Wert hat, wenn y die korrekte Klasse zu $\vec{x}$ ist und einen niedrigen sonst. Eine solche Funktion kann recht einfach parametrisiert werden. O.B.d.A. nehmen wir an, dass $f(\vec{x}, -1) = 0$. Außerdem gehen wir davon aus, dass die Daten auf einen beliebigen Merkmalsvektor $\phi(\vec{x}) \in \mathbb{R}^k$ abgebildet werden. Aus diesem Vektor kann ein neuer Merkmalsvektor $\phi(\vec{x}, y)$ folgendermaßen berechnet werden:

$$\phi(\vec{x}, y) = [y = 1]\ \phi(\vec{x})$$

Die Diskriminanzfunktion f kann dann linear in den Eigenschaften parametrisiert werden:

$$f(\vec{x}, y) = \sum_{i=1}^{k} \phi_j(\vec{x}, y)\beta_j = \phi(\vec{x}, y)^{\mathsf{T}}\vec{\beta}$$

Die Parameter β_j können dabei als Maß für die Wichtigkeit bestimmter Eigenschaften für die Zugehörigkeit zur Klasse 1 betrachtet werden. Die Funktion f kann auch als Wahrscheinlichkeit der Klassenzugehörigkeit interpretiert werden

$$p(1|\vec{x}) = \sigma(f(\vec{x}, 1)) = \frac{e^{f(\vec{x},1)}}{1 + e^{f(\vec{x},1)}}\ \text{ mit }\ \sigma(z) = \frac{e^z}{1 + e^z}$$

wobei man σ die logistische Sigmoidfunktion nennt. Ein optimales $\vec{\beta}$ kann nun gefunden werden, indem die Kostenfunktion

$$K(\vec{\beta}) = -\sum_{i=1}^{n} \log p(y_i, \vec{x}_i) + \lambda \|\vec{\beta}\|^2$$

minimiert wird. $\lambda\|\vec{\beta}\|^2$ ist hierbei ein Regularisierungsterm um Überanpassung zu verhindern. Das gelernte $\vec{\beta}$ ergibt sich also aus

$$\vec{\beta}_{opt.} = \underset{\vec{\beta}}{\operatorname{argmin}}\ K(\vec{\beta})$$

Mangels analytischer Lösung dieser Gleichung wird $\vec{\beta}_{opt.}$ durch iterative Methoden, wie beispielsweise das Newton-Verfahren, berechnet. Unter Zuhilfenahme der gelernten Parameter lässt sich ein neues Element $\vec{x}$ durch die Berechnung folgender Formel berechnen:

$$\hat{y} = sign(\phi(\vec{x})^{\mathsf{T}} \vec{\beta}_{opt.})$$

3.1.2 Support Vector Machines

Support Vector Machines legen eine Entscheidungsgrenze in Form einer Hyperebene durch den Merkmalsraum fest, anhand welcher Eingabewerte klassifiziert werden. Die Besonderheit von SVMs liegt hierbei darin, dass die Lage der Grenze so gewählt wird, dass alle erlernten Beispiele einen Mindestabstand von ihr haben. Dieser Abstand wird Rand (eng.: margin) genannt und hängt nur von den Punkten ab, die der Entscheidungsgrenze am nächsten liegen. Die Hyperebene lässt sich anhand dieser Punkte vollständig beschreiben, weshalb man sie Stützvektoren ($\rightarrow$ Support Vectors) nennt.
Wie schon bei der logistischen Regression betrachten wir nur den dualen Fall mit

$$D = \{\vec{x}_i, y_i\}_{i=1}^{N};\ y_i \in \{-1, 1\}.$$

Die gesuchte Hyperebene kann durch einen Normalenvektor $\vec{\beta}$ und ein Bias β_0 beschrieben werden, so dass die Zugehörigkeit zu einer Klasse mit

$$f(\vec{x}) = sign(\vec{x}^{\mathsf{T}} \vec{\beta} + \beta_0)$$

entschieden werden kann. Der Normalenvektor wird bestimmt, indem das folgende Minimum berechnet wird:

$$\min_{\vec{\beta}, \vec{\xi}, \beta_0} \|\vec{\beta}\|^2 + C \sum_{i=1}^{n} \xi_i \ \text{ mit } \ C > 0 \ \text{ so dass } y_i f(\vec{x}_i) \geq 1 - \xi_i;\ \xi_i \geq 0$$

Die Nebenbedingung gibt an, dass alle Instanzen des Trainingsdatensatzes korrekt, bis auf eine Überschussvariable ξ_i, klassifiziert werden müssen. Diese Variable nimmt einen Wert größer

null an, wenn die Instanz falsch klassifiziert wurde und ist nötig, da nicht alle Daten linear separierbar sind. Das Optimierungsproblem beinhaltet zwei zu minimierende Dimensionen: Zum einen maximiert sie den margin (äquivalent zur Minimierung der quadratischen Norm von $\vec{\beta}$, siehe [Vap82]) und zum anderen minimiert sie den Fehler. Das Zulassen kleiner Fehler kann als Regularisierung betrachtet werden, die Konstante C regelt ihr Maß.
Das obige Problem wird für gewöhnlich in seiner dualen Form gelöst, dazu wird $\vec{\beta}$ als Linearkombination aus den Eingabewerten des Trainingsdatensatzes geschrieben:

$$\vec{\beta} = \sum_{i=1}^{n} \alpha_i y_i \vec{x}_i$$

Die Variable α_i ist dabei ein Indikator, der angibt ob $y_i f(\vec{x}_i) \geq 1 - \xi_i$ gilt. Die duale Form kann unter Verwendung von Lagrange-Multiplikatoren [Ber99] gewonnen werden und führt zu

$$\max_{\alpha} -\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \vec{x}_i^{\mathsf{T}} \vec{x}_j + \sum_{i=1}^{n} \alpha_i \text{ so dass } 0 \leq \alpha_i \leq C; \sum_{i=1}^{n} \alpha_i y_i = 0.$$

Da das Optimierungsproblem in der letzten Schreibweise nur noch $\vec{x}_i^{\mathsf{T}} \vec{x}_j$ verwendet kann an dieser Stelle sehr einfach ein Wechsel in den Merkmalsraum vollzogen werden, indem stattdessen $\phi(\vec{x}_i)^{\mathsf{T}} \phi(\vec{x}_j)$ verwendet wird. Es ist sogar möglich an Stelle des Skalarprodukts eine beliebige Kernelfunktion $k(\vec{x}_i, \vec{x}_j)$ zu verwenden. Arbeitet diese in einem implizit höherdimensionalen Raum, ist als Ergebnis im Eingaberaum auch eine nicht-lineare oder nicht zusammenhängende Entscheidungsgrenze möglich. Die angepasste Klassifizierungsfunktion nimmt dann diese Form an:

$$f(\vec{x}) = sign\left(\sum_{i=1}^{n} \alpha_i y_i k(\vec{x}_i, \vec{x}) + \beta_0 \right)$$

3.1.3 Entscheidungsbäume

Im Gegensatz zu logistischer Regression und Support Vector Machines wird bei Entscheidungsbäumen keine mathematische Funktion erlernt. Stattdessen wird ein Binärbaum aus logischen Regeln erstellt, um Eingabedaten zu klassifizieren. Jeder Knoten des Baums enthält eine Regel, während in den Blättern jeweils eine der Klassen steht. Um eine Instanz zu klassifizieren, wird an der Wurzel begonnen und abhängig vom Wahrheitswert der Regel in Hinblick auf die Eingabe entschieden, welcher Kindknoten als nächstes gewählt wird. Wird ein Blatt erreicht, wird die entsprechende Klasse ausgegeben.
Der bekannteste Algorithmus für die Erstellung eines solchen Baums ist CART (classification and regression tree) [BFOS84]. CART unterteilt den Merkmalsraum in k durch den Baum definierte disjunkte rechteckige Regionen R. Um eine bisher ungeteilte Region R_j aufzuteilen,

wird eine Dimension a des Merkmalsraum und ein Schwellenwert t nach folgendem Prinzip gewählt:

$$\min_{a,t}\left(\min_{c_1}\sum_{i:\phi(x_i)\in R_j\wedge\phi(x_i)_a\leq t}(y_i-c_1)^2+\min_{c_2}\sum_{i:\phi(x_i)\in R_j\wedge\phi(x_i)_a>t}(y_i-c_2)^2\right)$$

Bildlich kann man sich das so vorstellen, dass entlang jeder Dimension eine Hyperebene bewegt wird, welche die Region in zwei Teile aufteilt. In diesen beiden Teilen wird jeweils die „Ähnlichkeit“ der Klassen der darinliegenden Instanzen bewertet. Die beste Aufteilung wird dann als Regel der Form $\phi(x)_a > t$ übernommen. Sind in einer Region alle Daten der gleichen Klasse zugeordnet, wird ein Blatt dieser Klasse erstellt, anstatt eine erneute Aufteilung vorzunehmen.
Die maximale Tiefe des Entscheidungsbaumes kann entweder vorher festgelegt oder mittels Pruning verringert werden. Dafür wird iterativ der Knoten entfernt, der am wenigsten zu

$$\sum_{i=1}^{n}(y_i-f(\phi(x_i)))^2$$

beiträgt. Wie stark der Baum auf diese Art beschnitten wird ist mit den Regularisierungsmethoden anderer Algorithmen vergleichbar. Nach Konstruktion des Entscheidungsbaums kann f zur Klassifizierung genutzt werden:

$$f(x)=\sum_{j=1}^{k}c_j[x\in R_j]$$

c_j ist die Klasse, die dem Blatt R_j zugeordnet wurde.

3.1.4 Boosting

Sowohl Boosting als auch Bagging zählen zu den Meta-Lernverfahren. Sie arbeiten mit Hilfe anderer Lernverfahren, indem sie die Ausgaben mehrerer gelernter Modelle gewichtet mitteln. Diese Modelle sind oft mit recht einfachen Lernverfahren wie Entscheidungsbäumen gewonnen.
Beim Boosting können die verwendeten Modelle verschiedenen Typs sein. Man spricht bei den einzelnen Klassifikatoren oft von schwachen Klassifikatoren, ihre einzeln betrachtete Leistungsfähigkeit ist meist mäßig. Die zum Training eingesetzten Daten werden beim Boosting für jedes Modell anders gewichtet. Ein populärer Vertreter dieser Methode ist **AdaBoost** [FS97], das erstmals 1997 von Robert E. Schapire und Yaov Freund beschrieben wurde. Dieser Algorithmus ist hier beschrieben.
Die Ausgangslage ist wieder ein binäres Klassifizierungsproblem mit Trainingsdaten $D = \{\vec{x}_i, y_i\}_{i=1}^{N};\ y_i \in \{-1, 1\}$. Gegeben sei ein Algorithmus um Klassifikatoren der Form $G(x) \in$

$\{-1, 1\}$ zu trainieren. Ziel ist es, eine Reihe von Klassifikatoren $G_1, ..., G_M$ auf gewichteten Instanzen der Trainingsdaten zu trainieren, um einen neuen Klassifikator zu erhalten:

$$G(x) = sign \sum_{m=1}^{M} \alpha_m G_m(x)$$

Die Gewichtung der Daten einer jeden Instanz wird dabei jeweils aus dem Fehler der vorher trainierten Instanz gewonnen. Falsch klassifizierte Trainingsdaten werden dabei stärker gewichtet, während eine korrekte Einschätzung zu keiner Veränderung führt. Algorithmus 3.1 zeigt den Algorithmus, mit dem die Klassifikatoren und Gewichte erstellt werden. Außerdem werden die Gewichte $\alpha_1, ..., \alpha_M$ ermittelt, die den Beitrag der einzelnen Lerner zum neuen Klassifikator bestimmen. Die so gewonnenen G_m und α_m können dann zur Berechnung von

Algorithmus 3.1 AdaBoost

Input: D
Output: $G_{1,...,M}$; $\alpha_{1,...,M}$
for all i=1,...,M **do**
  $w_i = \frac{1}{n}$
end for
for all m=1,...,M **do**
  Trainiere G_m auf dem mit w_i gewichteten Datensatz
  $fehler_m = \frac{\sum_{i=1}^{n} w_i [y_i \neq G_m(x_i)]}{\sum_{i=1}^{n} w_i}$
  $\alpha_m = \log(\frac{1-fehler_m}{fehler_m})$
  for all j=1,...,M **do**
    $w_i = w_i \exp(\alpha_m [y_i \neq G_m(x_i)])$
  end for
end for

$G(x)$ und damit zur Klassifizierung verwendet werden.

3.1.5 Bagging

Auch Bagging verwendet einen Satz von Klassifikatoren des selben Typs, im Gegensatz zum Vorgehen beim Boosting werden die Trainingsdaten aber nicht gewichtet. Stattdessen wird jeder Klassifikator auf einer Bootstrap-Stichprobe trainiert. Diese ist eine gleichverteilt zufällige Auswahl aus dem Datensatz. Sie hat die gleiche Größe wie der Trainingsdatensatz, Doppeltziehungen sind aber erlaubt. Bei Vorhandensein von M Klassifikatoren, die jeweils eine Diskriminanzfunktion $f_m(x)$ trainieren, erhält man die Diskriminanzfunktion des Bagging-Verfahrens durch einfaches Mitteln:

$$f(x) = \frac{1}{M} \sum_{m=1}^{M} f_m(x)$$

Random Forests

Ein bekanntes Beispiel für Bagging, das auch in Kapitel 5 zum Einsatz kommt, sind die Random Forests. Ein Random Forest wird mittels Bagging aus Entscheidungsbäumen gewonnen. Zusätzlich zur Randomisierung des Trainingsdatensatzes wird außerdem bei der Aufteilung einer Region die zu teilende Dimension nur aus einer zufälligen, oft sehr kleinen Teilmenge der Merkmale gewählt.

3.2 Merkmalsvektoren

Als Merkmalsvektor bezeichnet man im Bereich des maschinellen Lernens einen Vektor aus numerischen Merkmalen, die ein Objekt repräsentieren. Dies ist nötig, da viele Lernalgorithmen, beispielsweise die in Abschnitt 3.1, auf der Verwendung von numerischen Vektoren beruhen. Die Merkmale können dabei von einfachen Dingen, wie der Position im Raum, bis hin zu komplexen zusammengesetzten Berechnungen reichen.

Ein guter Merkmalsvektor enthält Eigenschaften, die für den konkreten Anwendungsfall von Bedeutung sind und eine möglichst gute Unterscheidung der Klassen ermöglichen. Die Erstellung eines solchen geeigneten Merkmalsvektors für eine bestimmtes Lernszenario ist daher oft ein aufwendiger Prozess, der Wissen über die Domäne, sowie ausgiebiges Testen erfordert.
Einen alternativen Ansatz stellt das automatisierte Lernen geeigneter Merkmale durch spezielle Algorithmen dar. Analog zur Definition in Abschnitt 3.1 wird auch hier zwischen überwachtem und unüberwachtem Lernen unterschieden, abhängig davon, ob die Trainingsdaten klassifiziert sind.
Ein populäres Beispiel für überwachtes Merkmalslernen sind künstliche neuronale Netze. Diese können durch die Wahl von ausreichend kleinen verdeckten Schichten so konfiguriert werden, dass sie eine Repräsentation der Eingabe in kleinerer Dimension lernen [KSH]. Diese Repräsentation kann als Merkmalsvektor interpretiert werden.
Eine Anwendung von unüberwachtem Merkmalslernen ist die Hauptkomponentenanalyse (englisch: Principal Component Analysis (PCA)). Sie basiert auf dem Ansatz, die vorhandenen Daten so in einen Merkmalsraum niedrigerer Dimension abzubilden, dass dabei der Informationsverlust minimal ist. Das heißt, dass sich die Daten nach Rückabbildung möglichst wenig von ihrem Original unterscheiden dürfen. Da die dadurch neu gewonnenen Merkmale Eigenvektoren der Kovarianzmatrix sind, spricht man bei diesen abhängig von der Anwendung beispielsweise von „Eigengesichtern“.

Im Folgenden werden einige Beispiele für manuell gewonnene Merkmale vorgestellt. Die letzten vier sind zum Teil speziell für die Erkennung von Wirbeln in Vektorfeldern konstruiert und kommen auch in Kapitel 5 zum Einsatz.

3.2.1 Lineare und polynomielle Merkmale

Die einfachste Methode um Merkmalsvektoren zu erstellen sind lineare Merkmale. Geht es beispielsweise um Positionen im Raum kann einfach der Positionsvektor als Merkmalsvektor verwendet werden. Ein Bild oder Bildausschnitt lässt sich als Vektor mit allen Farb- oder Helligkeitswerten der Pixel darstellen, dies ist natürlich auch auf andere Skalar- oder Vektorfelder übertragbar. Möchte man die Korrelation der Komponenten mit berücksichtigen, können auch polynomielle Merkmale verwendet werden. Am Beispiel von einer Position $\vec{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ kann ein quadratischer Merkmalsvektor $\phi(\vec{x})$ wie folgt berechnet werden:

$$\phi(\vec{x}) = \begin{pmatrix} x_1 \\ x_2 \\ x_1^2 \\ x_2^2 \\ x_1 x_2 \end{pmatrix}$$

Dies lässt sich analog auf Polynome höheren Grads erweitern, erzeugt bei hochdimensionalen Ausgangsdaten wie zweidimensionalen Vektorfeldern aber schnell zu große Merkmalsvektoren.

3.2.2 Histogramme

Eine Möglichkeit Vektorfelder oder Teile von Vektorfeldern zu beschreiben, ist ein Histogramm über die Eigenschaften der darin enthaltenen Datenpunkte anzufertigen. Mögliche Größen wären beispielsweise Richtung, Magnitude oder partielle Ableitungen des Felds. Eine solche Größe wird in eine feste Anzahl an Behältern (eng.: bins) aufgeteilt, in welche die Datenpunkte einsortiert werden. Das resultierende Histogramm wird als Merkmalsvektor verwendet.

3.2.3 Histogrammvarianz

Berechnet man wie im vorangegangenen Unterabschnitt beschrieben ein Histogramm über eine Größe des Vektorfelds, kann dessen Varianz als Merkmal verwendet werden. Das wird am Beispiel der Strömungsrichtung für die Erkennung von Wirbeln deutlich: Im Idealfall eines kreisrunden Wirbels sind alle Richtungen gleich häufig vorhanden, die Varianz des Histogramms nimmt den Wert null an. Je stärker eine oder mehrere Richtungen dominieren, desto größer wird die Varianz. Ein Nachteil dieser Methode der Unterscheidung ist die ebenso erhöhte Varianz bei Wirbeln, die eine eher längliche Form aufweisen.

3.2.4 Magnitude des Durchschnittsvektors

Ein weiteres für die in Kapitel 5 klassifizierten Wirbel relevantes Merkmal ist die Magnitude der Durchschnittsgeschwindigkeit. Dafür wird über alle im (Teil-)Vektorfeld enthaltenen Datenpunkte ein Durchschnittsvektor berechnet und dessen Betrag als Merkmal verwendet. Diese Größe ist interessant, da ein Wirbel durch die Kreisbewegung der Strömung meist einen Durchschnittsvektor mit niedriger Magnitude hat, während zum Beispiel eine laminare Strömung eine hohe Magnitude aufweist. Das Merkmal eignet sich daher zur Unterscheidung. Nicht zu verwechseln ist die Eigenschaft mit der Durchschnittsmagnitude, die Auskunft über die durchschnittliche Fließgeschwindigkeit gibt, für Wirbel allerdings eine eher geringe Aussagekraft besitzt.

3.2.5 Fluss

Der Fluss gibt an, wie viel Masse in einen Punkt x_c hinein und aus ihm hinaus transportiert wird. Die Berechnung orientiert sich an [DANS10]. Um den Fluss zu ermitteln, wird für alle anderen in der Region vorhandenen Punkte x_j das Skalarprodukt aus dem Vektor v_j an dieser Stelle und der Differenz der Punkte x_c und x_j berechnet. Es beschreibt damit, wie stark der Fluss an dieser Stelle zu x_c hin oder von ihm weg strömt. Die so ermittelten Skalarprodukte werden schließlich aufsummiert:

$$Flux(x_c) = \sum_j v_j^{\mathbb{T}}(x_j - x_c)$$

Damit dieses Maß für die Beschreibung von einem Teilvektorfeld aussagekräftig wird, muss x_c sinnvoll gewählt werden. Der Betrag des Flusses wird minimal, wenn alle Vektoren der anderen Punkte senkrecht zum Differenzvektor stehen. Dies ist genau dann der Fall, wenn x_c im Zentrum eines (runden) Wirbels liegt. Daher kann als Merkmal zur Klassifizierung von Wirbeln der betragsmäßig minimale Fluss über alle Punkte verwendet werden:

$$Flux_{area} = \min_{x \in area} Flux(x)$$

Eine zweite Möglichkeit besteht darin, einen kritischen Punkt zu suchen und seinen Fluss zu bewerten. Ein solcher Punkt zeichnet sich dadurch aus, dass seine Flussgeschwindigkeit den Wert null hat. Weil nicht jeder untersuchte Ausschnitt einen kritischen Punkt aufweisen muss, kann für x_c beispielsweise der Punkt mit der niedrigsten Geschwindigkeit verwendet werden.
Da das Zentrum eines Wirbels ein kritischer Punkt ist, stellt dieses Vorgehen sicher, dass bei Vorhandensein eines Wirbels immer dessen Zentrum bewertet wird. Der Unterschied zur Verwendung des Minimums ist hier, dass dies auch sichergestellt ist, wenn der Wirbel nicht ausreichend rund ist.

3.3 Strömungsvisualisierung

Vektorfelddaten, wie sie zum Beispiel aus Strömungssimulationen gewonnen werden, besitzen keine inhärente für Menschen verständliche Darstellungsform. Um solche Daten annotieren zu können ist es daher unerlässlich, sie mit Visualisierungsverfahren in eine für den Benutzer interpretierbare Form zu überführen. Eine Möglichkeit dies zu erreichen besteht darin, die Stromlinien sichtbar zu machen. Stromlinien sind Kurven deren Ableitung an jedem Punkt der Geschwindigkeit des Vektorfelds entspricht. Sie stellen damit die Pfade dar, auf denen sich masselose Partikel durch das Vektorfeld bewegen würden.
Eine Visualisierungsmethode, die darauf beruht, ist die „Line Integral Convolution“ (LIC) oder Linienintegralfaltung. Sie zählt zu den texturbasierten Vektorfeldvisualisierungen und wurde ursprünglich in [CL93] vorgeschlagen. Die Formulierung orientiert sich hier aber an [Wei09]. LIC verwendet als Ausgangstextur ein weißes Rauschen N und faltet dieses mit Hilfe eines Faltungskernes entlang der Stromlinien. Im einfachsten Fall wird ein Box-Kernel verwendet, welcher die Werte einfach mittelt. Die Faltung führt zu ähnlichen Grauwerten entlang von Stromlinien, während die Korrelation entgegen der Linien gering ist. Für den menschlichen Betrachter wird so ein feldlinienartiges Bild sichtbar. Ein Beispiel ist in Abbildung 3.1 zu sehen.

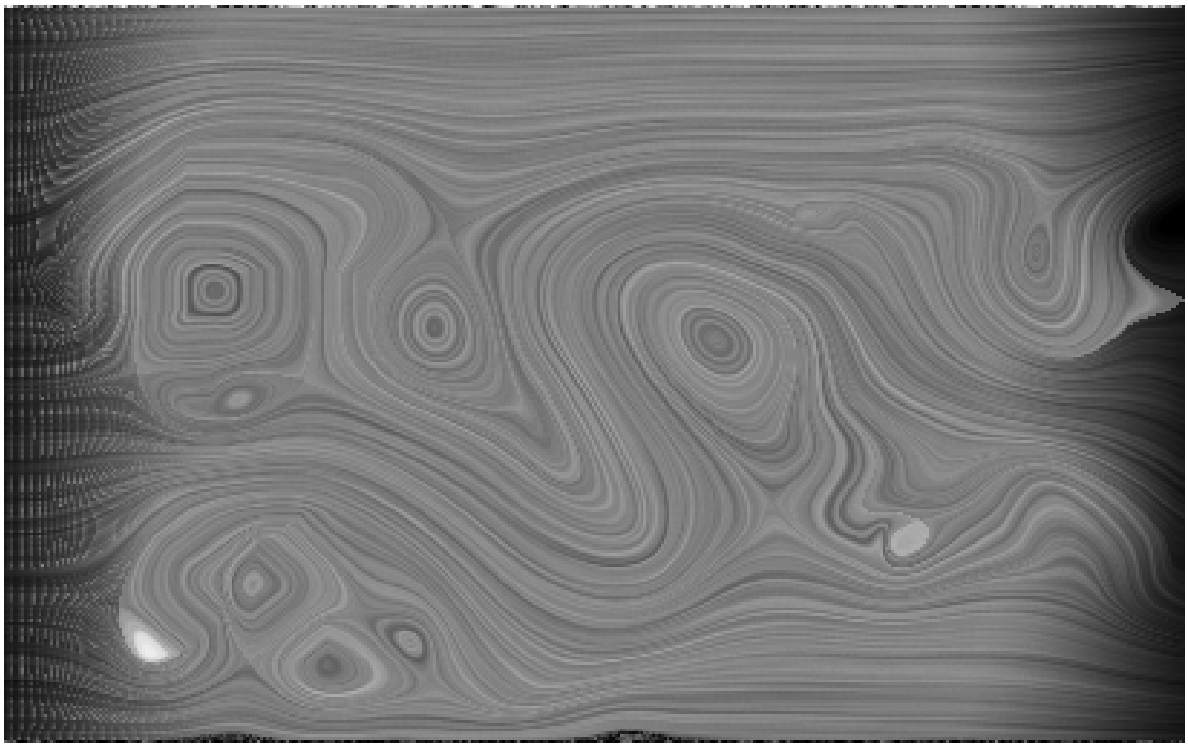

Abbildung 3.1: Visualisierung mittels Line Integral Convolution.

Um den Grauwert des Pixels der Visualisierung D an der Position $\vec{x}$ bestimmen zu können, muss zuerst eine Stromlinie $\sigma(s)$ berechnet werden, die mit der Bogenlänge s parametrisiert ist und die Position $\vec{x}$ enthält ($\sigma(s_0) = \vec{x}$). Dafür kann die Differentialgleichung

$$\frac{\mathrm{d}\sigma(\tau)}{\mathrm{d}\tau} = v(\sigma(\tau))$$

mit der Anfangsbedingung $\sigma(\tau_0) = \vec{x}$ gelöst werden, wobei v das Vektorfeld ist. Der Parameter τ entspricht nicht notwendigerweise der Bogenlänge, berechnet man die Stromlinie im normierten Vektorfeld, ist dies aber erfüllt:

$$\frac{\mathrm{d}\sigma(s)}{\mathrm{d}s} = \frac{v(\sigma(s))}{|v(\sigma(s))|}$$

Die Anfangsbedingung ist hier entsprechend $\sigma(s_0) = \vec{x}$. Die Berechnung des Grauwerts erfolgt dann durch die Auswertung des Integrals

$$D(\vec{x}) = \int_{s_0-L_s}^{s_0+L_e} k(s - s_0)N(\sigma(s))\mathrm{d}s.$$

k ist hierbei ein Faltungskern mit dem Träger $[-L_s, L_e]$. Die in Kapitel 4 beschriebene Anwendung verwendet hierfür einen Gauß-Filter wie er zum Beispiel in [NA02] beschrieben wird, da dieser ein besseres spektrales Verhalten als ein Box-Kernel erzielt.

Für die Annotation der Daten kann außer den Stromlinien auch die Lage von kritischen Punkten $\vec{x}_c$ interessant sein. Diese sind in einem Vektorfeld einfach zu identifizieren, da für sie gilt: $v(\vec{x}_c) = 0$. Bei Bedarf können diese Punkte zusätzlich markiert werden.

3.4 Evaluationsverfahren

Da die Wahl eines Algorithmus und der entsprechenden Parameter, sowie die Konstruktion eines geeigneten Merkmalsvektors sich meist sehr schwierig gestalten und das Bewerten verschiedener Ansätze auf theoretischem Weg kaum möglich ist, kann beim maschinellen Lernen für gewöhnlich nicht auf eine aussagekräftige und gut vergleichbare Auswertung verzichtet werden.
Das allgemeinste und vermutlich am weitesten verbreitete Verfahren ist hierbei die Verwendung eines Trainings- und eines ebenfalls vorklassifizierten Testdatensatzes. Das Modell wird mit dem Trainingsdatensatz erlernt und seine Leistung danach mit Hilfe des Testdatensatzes bewertet. Das kann zum Beispiel mit einer Konfusionsmatrix und diversen daraus abgeleiteten Größen geschehen. Das Moving Window Verfahren ist dagegen eigentlich der Anwendung von erlernten Modellen entnommen, kann aber auch zur Evaluation eingesetzt werden.

Tabelle 3.1: Beispiel einer Konfusionsmatrix

Klassifiziert als:	Klasse A	Klasse B
Klasse A	122	13
Klasse B	7	33

3.4.1 Cross Validation

Die Cross Validation (eng.: Kreuzvalidierung) ist ein Sonderfall der Verwendung von Trainings- und Testdatensatz. Sie arbeitet nur auf einem großen Datensatz. Bei der Evaluation wird folgendermaßen vorgegangen: Ein fester Parameter k legt fest, in wie viele gleichmächtige Teilmengen der Datensatz zerlegt wird. Jeder dieser Datensätze wird dann einmal als Testdatensatz verwendet, während die Vereinignung über die restlichen $k-1$ Teilmengen als Trainingsdatensatz fungiert. Die auf diese Art gewonnenen Ergebnisse werden zum Schluss gemittelt und als Gesamtergebnis verwendet.

Cross Validation kommt auch häufig zum Einsatz, um Parameter von Lernalgorithmen automatisch zu wählen. Dafür wird für eine große Zahl möglicher Parameterwerte unter Verwendung von Cross Validation der mittlere Fehler über alle k Testdatensätzen ermittelt. Der Wert, welcher den niedrigsten durchschnittlichen Fehler zur Folge hat, wird für den Parameter verwendet.

3.4.2 Konfusionsmatrix

Ein verbreitetes Evaluationsverfahren im maschinellen Lernen ist die Verwendung einer sogenannten Konfusionsmatrix und daraus resultierender Maße. In einer solchen Matrix entspricht eine Dimension der tatsächlichen, die andere der vom Modell bestimmten Klasse. Die Einträge entsprechen Aussagen der Form: „x Instanzen der Klasse a wurden als Klasse b erkannt“. Tabelle 3.1 zeigt beispielhaft eine Konfusionsmatrix. Im Falle eines dualen Klassifizierungsproblems enthält die Diagonale also die Zahl der korrekt klassifizierten Instanzen (**true positive (TP)** und **true negative (TN)**) während die beiden anderen Einträge die Zahl der falsch klassifizierten Instanzen enthält (**false positive (FP)** und **false negative (FN)**). Aus dieser Matrix lassen sich Größen mit statistischer Bedeutung ableiten, von denen einige hier beschrieben sind.

Sensitivität

Die Sensitivität oder True-Positive-Rate (auch: recall) gibt an, wie groß der Anteil der korrekt als positiv erkannten Instanzen an der Menge aller positiven Instanzen ist. Sie berechnet sich wie folgt:

$$\text{TP-rate} = \frac{TP}{TP + FN}$$

False-Negative-Rate

Die False-Negative-Rate ist entsprechend zur Sensitivität definiert und gibt an, wie groß der Anteil der falsch als negativ klassifizierten Instanzen an der Menge aller positiven Instanzen ist:

$$\text{FN-rate} = \frac{FN}{TP + FN} = 1 - \text{TP-rate}$$

Spezifität

Die Spezifität oder True-Negative-Rate gibt an, wie hoch der Anteil der korrekt als negativ erkannten Instanzen an der Menge aller negativen Instanzen ist:

$$\text{TN-rate} = \frac{TN}{TN + FP}$$

False-Positive-Rate

Auch zur Spezifität gibt es eine entsprechende zweite Rate. Die False-Negative-Rate gibt an, wie groß der Anteil der falsch als positiv klassifizierten Instanzen an der Menge aller negativen Instanzen ist:

$$\text{FP-rate} = \frac{FP}{TN + FP} = 1 - \text{TN-rate}$$

Genauigkeit und Trennfähigkeit

Die Genauigkeit oder Precision gibt an, wie viele der als positiv klassifizierten Instanzen, dies tatsächlich auch sind. Sie ist damit ein Maß für die Relevanz einer positiv-Bewertung. Berechnet wird sie durch:

$$\text{Precision} = \frac{TP}{TP + FP}$$

Analog dazu ist die Trennfähigkeit oder Negative Prediction Value (NPV) definiert:

$$\text{NPV} = \frac{TN}{TN + FN}$$

Korrekt- und Falschklassifikationsrate

Die Korrektklassifikationsrate (KKR) gibt an, wie hoch der Anteil der korrekt klassifizierten Instanzen an der Gesamtzahl der Instanzen ist. Analog dazu ist die Falschklassifikationsrate (FKR) definiert:

$$\text{KKR} = \frac{TP + TN}{TP + FP + TN + FN} \text{FKR} = \frac{FP + FN}{TP + FP + TN + FN} = 1 - \text{KKR}$$

F-Maß

Das F-Maß kombiniert Genauigkeit und Sensitivität in dem es das harmonisches Mittel bildet.

$$F = \frac{2 \times \text{Precision} \times \text{TP-rate}}{\text{Precision} + \text{TP-rate}}$$

3.4.3 Moving Window

Das Moving Window Verfahren findet häufig Anwendung, wenn Bilder oder vergleichbare Daten, auf bestimmte Merkmale untersucht werden sollen. Es ist auch auf beliebige Vektorfelddaten anwendbar. Das Vorgehen bei der Detektion von Objekten innerhalb solcher Daten ist für gewöhnlich zuerst den zu untersuchenden Bereich als Merkmalsvektor zu repräsentieren und ihn anschließend anhand des Vektors von einem zuvor trainierten Klassifikator einordnen zu lassen.
Wurde dieser Klassifikator für Klassen erstellt, die räumlich begrenzten Merkmalen des Skalar- oder Vektorfelds entsprechen, ergibt sich eine etwas komplexere Problemstellung. Statt nur die Existenz eines solchen Merkmals festzustellen ist nämlich in der Regel auch die Position und Größe des Merkmals von Interesse.
Eine Methode, die dieses Problem löst, ist das Moving Window Verfahren. Dabei werden Teilausschnitte des Felds separiert und dann klassifiziert. Diese Ausschnitte haben die Form von Rechtecken, die iterativ verkleinert werden. Für jede Rechteckgröße, wird das Rechteck über alle voneinander verschiedenen möglichen Positionen im Gesamtfeld gelegt. Für jede dieser Positionen wird dann ein Merkmalsvektor berechnet und eine Klassifizierung vorgenommen.
Bildlich kann man sich ein Fenster vorstellen, durch das der Klassifikator auf die Daten blickt. Dieses Fenster wird über den gesamten Datensatz bewegt und danach verkleinert, um

von vorne zu beginnen. Von dieser Vorstellung des Verfahrens stammt auch der Name der Methode.
Wird ein Ausschnitt als ein Objekt von Interesse klassifiziert, ist sofort auch dessen Position und Größe bekannt. Da beim Moving Window Verfahren viele Ausschnitte betrachtet werden, die räumlich nah beieinander liegen und eine ähnliche Größe haben, führt ein gesuchtes Objekt meist zu einer Vielzahl von positiven Klassifizierungen. Eine Weiterverarbeitung der gewonnenen Ergebnisse ist deshalb in der Regel unerlässlich. Eine Möglichkeit, die gewonnen Ergebnisse zu visualisieren, wird im nächsten Abschnitt besprochen.

Prediction Ratio

Hält man für jeden Punkt des Felds fest, in wie vielen klassifizierten Ausschnitten er enthalten war und wie viele dieser Ausschnitte als positiv klassifiziert wurden, kann man einen Quotienten berechnen, der Auskunft darüber gibt, wie wahrscheinlich es ist, dass besagter Punkt Teil eines gesuchten Objekts ist.

$$PR(p) = \frac{|\{\text{Rechteck } R \mid p \in R \wedge G(\phi(R)) = 1\}|}{|\{\text{Rechteck } R \mid p \in R\}|}$$

Berechnet man PR für jeden Punkt des Felds und färbt die Visualisierung des Felds proportional dazu, kann sichtbar gemacht werden, auf welchen Bereichen der Klassifikator besonders häufig ein positives Ergebnis zurückgegeben hat.
Denkbar ist auch, mittels Analyse der Gradienten von PR oder mit Hilfe von einem festen Schwellenwert Bereiche einzugrenzen, die dann als gefundene Objekte ausgegeben werden.

4 Systembeschreibung

In diesem Kapitel wird das System beschrieben, das im Rahmen dieser Arbeit entstanden ist. Dazu wird in Abschnitt 4.1 eine Anforderungsanalyse erstellt, aus deren Ergebnissen die in Abschnitt 4.2 beschriebene Umsetzung resultiert. In 4.3 wird näher auf das entwickelte Plugin-System eingegangen, während sich Abschnitt 4.4 den verwendeten Merkmalsvektoren widmet.

4.1 Anforderungsanalyse

Ziel der Entwicklung ist ein Programm, dass die Annotation von Skalar- und Vektorfeldern, sowie die Erstellung einer Merkmalsvektorausgabedatei ermöglicht. An das Rahmenwerk sind dabei einige Anforderungen gestellt, die hier erläutert werden. Der Fokus liegt dabei zum einen auf einem simplen und effizienten Arbeitsablauf und zum anderen auf der Flexibilität und Wiederverwendbarkeit in Hinsicht auf die speziellen Anforderungen des einzelnen Nutzers.

1. Um das OpenCV Framework [Ope] verwenden zu können und um eine einfache Weiterentwicklung oder Anpassung zu ermöglichen, soll das System in C++ unter Verwendung von Qt [Qt] entstehen.
2. Es muss einen Auswahlmechanismus für Daten geben und die Verwaltung mehrerer eingelesener Datensätze soll möglich sein. Damit verschiedene Datentypen unterstützt werden können, soll das System Plugins unterstützen, welche die Datei einlesen.
3. Die geladenen Daten müssen in eine repräsentative Abbildung überführt werden. Diese Visualisierung soll ebenso durch ein Plugin übernommen werden, so dass der Nutzer beliebige Visualisierungsmethoden implementieren und mit dem Rahmenwerk verwenden kann.
4. Es muss ein Mechanismus implementiert werden, der die Selektion und Klassenzuweisung von Regionen im Datensatz erlaubt. Das Auswählen von Regionen soll mit Hilfe beliebiger polygonaler Formen geschehen.
   Außerdem soll es, wie schon bei den eingelesenen Daten, die Möglichkeit zur Verwaltung der Annotationen geben.

5. Projekte sollen zur späteren Fortsetzung der Arbeit gespeichert und geladen werden können. Ein Projekt beinhaltet dabei eine Liste der ausgewählte Dateien, sowie alle darin selektierten Regionen und ihre Klassen.
6. Das System soll auf Wunsch eine Merkmalsvektorausgabedatei erzeugen, die anhand der annotierten Regionen berechnet wird. Dies soll durch ein drittes Plugin umgesetzt werden, damit Nutzer leicht mit verschiedenen Designs experimentieren und diese vergleichen können.

4.2 Umsetzung

Aus der Anforderungsanalyse hat sich das in Abbildung 4.1 gezeigte System entwickelt. Das Bild zeigt einen Screenshot des Programms während der Verwendung und soll hier zur Erläuterung der Benutzung dienen.

Markierung 1 zeigt ein Listenelement, in dem die geladenen Dateien des aktuellen Projekts zu sehen sind. Hier kann der Nutzer auswählen, in welcher Datei er Annotationen vornehmen möchte. Es ist außerdem möglich die angewählte Datei mittels Tastaturbefehl oder über die Menüleiste aus dem Projekt zu entfernen.

Markierung 2 kennzeichnet ein zweites Listenelement, in dem die Regionen aufgezählt sind, die in der aktuellen Datei selektiert wurden. Diese sind durchnummeriert und zusätzlich nach der jeweiligen ihnen zugeordneten Klasse benannt.
Über ein Kontrollkästchen neben jedem Eintrag kann ausgewählt werden, ob die entsprechende Region in die Visualisierung gezeichnet werden soll. Wie schon bei der Dateiliste können auch hier einzelne Regionen aus dem Projekt entfernt werden.

Markierung 3 zeigt die Visualisierung der Daten. Um sowohl kleine als auch größere Regionen exakt selektieren zu können, ist es möglich in die Abbildung hinein- und aus ihr heraus zu zoomen.

Markierung 4 kennzeichnet die Menüleiste sowie die Auswahl der Selektionsart. Letztere bietet die Wahl zwischen rechteckiger und polygonaler Selektion und kann zusätzlich auch über Tastaturkürzel gesteuert werden. Die Menüleiste bietet neben der schon genannten Möglichkeit Dateien oder Regionen aus dem Projekt zu entfernen, Optionen zum Laden, Speichern und Neuanlegen von Projekten, zur Auswahl neuer Daten, zum Schreiben von Merkmalsvektorausgabedateien und zum Importieren von Plugins.

Markierung 5 vertritt den Selektions- und Klassenzuordnungsmechanismus. Die Region wird durch Mausklicks auf die gewünschten Eckpunkte ausgewählt. Dieser Prozess wird durch Auswahl des letzten Eckpunkts mit einem Rechtsklick im Falle der Polygone oder durch Auswahl eines zweiten Punktes bei Rechtecken beendet (die Wahl der linken oberen und rechten unteren Ecke genügt).

Ein Dialog fordert zur Eingabe des Klassennamens auf und lässt zusätzlich eine Farbe für die Klasse wählen, falls diese bis zu diesem Zeitpunkt noch nicht existiert.

Das oben beschriebene System führt zu dem in Abbildung 4.2 dargestellten Arbeitsablauf. Gestrichelte Pfeile entsprechen hierbei optionalen Arbeitsschritten, während die durchgezogenen Pfeile Schritte beschreiben, die zur Erstellung einer Ausgabedatei in Form von Merkmalsvektoren unerlässlich sind.

Nachdem ein neues Projekt erstellt oder ein bereits vorhandenes geladen wurde, muss unter Umständen das Plugin getauscht werden, das für das Laden der Dateien zuständig ist. Dies ist der Fall, wenn das Dateiformat nicht durch das Standard-Plugin unterstützt ist. Danach können beliebig viele Dateien in das Projekt geladen werden.
Der nächste Schritt im Arbeitsablauf ist die Selektion und Klassenzuweisung der für den Nutzer relevanten Regionen. Selbstverständlich können auch während des Annotierens neue Daten nachgeladen werden.
Ist die aktuelle Visualisierungsmethode an einigen Stellen zu ungenau, oder braucht der Nutzer aus anderen Gründen eine alternative Darstellungsform, kann er das Visualisierungs-Plugin tauschen. Die Daten und Annotationen bleiben erhalten, lediglich die Darstellung des Skalar- oder Vektorfelds wird geändert.
Falls der Nutzer die Arbeit unterbrechen muss, kann er das Projekt speichern und zu einem späteren Zeitpunkt wieder laden. Er kann die Arbeit dann genau an dem Punkt fortsetzen, an dem er aufgehört hat. Möchte er eine Merkmalsvektorausgabedatei erzeugen, kann er das jederzeit im Arbeitsablauf tun. Benötigt er einen anderen Merkmalsvektor muss er nur das entsprechende Plugin austauschen und erneut eine Ausgabedatei erzeugen.

4.3 Plugin-System

Plugins in C++ werden gewöhnlich in Form von dynamic linked libraries (DLLs) zur Verfügung gestellt. Eine solche Bibliothek implementiert einige Funktionen, die das Hauptprogramm dann ausführt. Dafür hat die DLL eine Funktion, die dem Programm als Einstiegspunkt dient, um die Bibliothek zu laden. Die übrigen Funktionen müssen über ihren Namen aufgerufen werden, der aber aufgrund des sogenannten *name mangling* in der Regel nicht dem originalen Funktionsnamen entspricht, sondern durch den Compiler generiert wird.
Bei der Verwendung von DLLs können insbesondere beim Einsatz verschiedener Compiler, aber auch beim Einsatz verschiedener Generationen des gleichen Compilers, Probleme auftreten, die das Laden des Plugins verhindern.
Zum einen unterscheidet sich das name mangling bei verschiedenen Compilern und zum anderen hat die Standardbibliothek von C++ keinen einheitlichen Binärstandard, was zu Fehlern bei der Verwendung von nicht grundlegenden Datentypen führen kann. Beides umgeht das eingesetze Plugin-System, um die Benutzung möglichst unanfällig für Fehler zu gestalten.

Um das name mangling zu verhindern, verwendet das Hauptprogramm jeweils nur eine Funktion der Plugins, die einen Zeiger auf ein Objekt zurückgibt, welches das Interface implementiert. Diese Funktion wird für den Compiler als C-Funktion deklariert, was das mangling verhindert.
Damit das so erhaltene Objekt normal verwendet werden kann, dürfen dessen vom Hauptprogramm aufgerufenen Methoden nur grundlegende Datentypen verwenden. Diese Einschränkung führt zu den eher unhandlichen Interfaces 4.1, 4.2 und 4.3. Um die Arbeit mit den Schnittstellen zu erleichtern, enthält jede Interface-Header-Datei zusätzlich eine abstrakte Klasse, die einen Großteil der Funktionen implementiert und die darin übergebenen Daten in sinnvollere Datentypen überführt. Ein Plugin-Autor kann, statt direkt das Interface zu implementieren, dann von dieser Klasse erben und muss nur noch die eigentliche Funktionalität programmieren.

Listing 4.1 Interface des Lade-Plugins

```
class ILoader {
public:
      virtual ~ILoader() {}

      virtual void load_file(char*) = 0;
      virtual int get_x_dim() = 0;
      virtual int get_y_dim() = 0;
      virtual int get_vec_dim() = 0;
      virtual double get_data_at(int y, int x, int vec) = 0;
      virtual void release() = 0;
};
```

Listing 4.2 Interface des Visualisierungs-Plugins

```
class IVisualizer {
public:
      virtual ~IVisualizer() {}

      virtual void pass_data_dim(int width, int height, int vec) = 0;
      virtual void pass_data_at(int y, int x, int vec, double data) = 0;
      virtual void calc_visualization() = 0;
      virtual int get_x_dim() = 0;
      virtual int get_y_dim() = 0;
      virtual int get_color_at(int y, int x) = 0;
      virtual void release() = 0;
};
```

Das Standard-Plugin für die Visualisierung, das auch in Kapitel 5 zum Einsatz kommt, implementiert Line Integral Convolution, wie sie in Kapitel 3 beschrieben wird. Die verwendeten Merkmalsvektoren sind in Abschnitt 4.4 erläutert.

Listing 4.3 Interface des Merkmalsvektor-Plugins

```
class IFeatureVec {
public:
      virtual ~IFeatureVec() {}

      virtual void pass_num_of_areas(int n) = 0;
      virtual void pass_area_label(int area_index, char* label) = 0;
      virtual void pass_vector(int area_index, int y, int x, double vec0, double vec1)
          = 0;
      virtual void save_feature_vector(char* file_name) = 0;

      virtual void release() = 0;
};
```

4.4 Verwendete Merkmalsvektoren

Um den als Standard verwendeten Merkmalsvektor möglichst allgemein und problemunabhängig zu gestalten, kommt hier ein Histogramm zum Einsatz, wie es in Abschnitt 3.2 beschrieben wird. Als aufzuteilende Domäne kommt die Richtungen des Geschwindigkeitsvektors zum Einsatz. Sie wird in acht gleichgroße Bereiche v_i eingeteilt, die jeweils einen Winkel von 45° einschließen. Das resultierende Histogramm H wird normiert, so dass die Summe über alle Richtungen den Wert eins annimmt:

$$H'(v_i) = \frac{H(v_i)}{\sum_i H(v_i)}$$

Dies verhindert den Einfluss der Größe des Bereichs, dessen Merkmalsvektor berechnet wird.

Als Vergleichsmerkmalsvektor wird in Kapitel 5 ein spezifisch für Wirbel entwickelter Vektor eingesetzt. Dieser enthält die Magnitude der Durchschnittsgeschwindigkeit, den Fluss und die Histogrammvarianz des oben beschriebenen Histogramms. Alle drei Größen sind gemäß der Definiton in Abschnitt 3.2 implementiert.

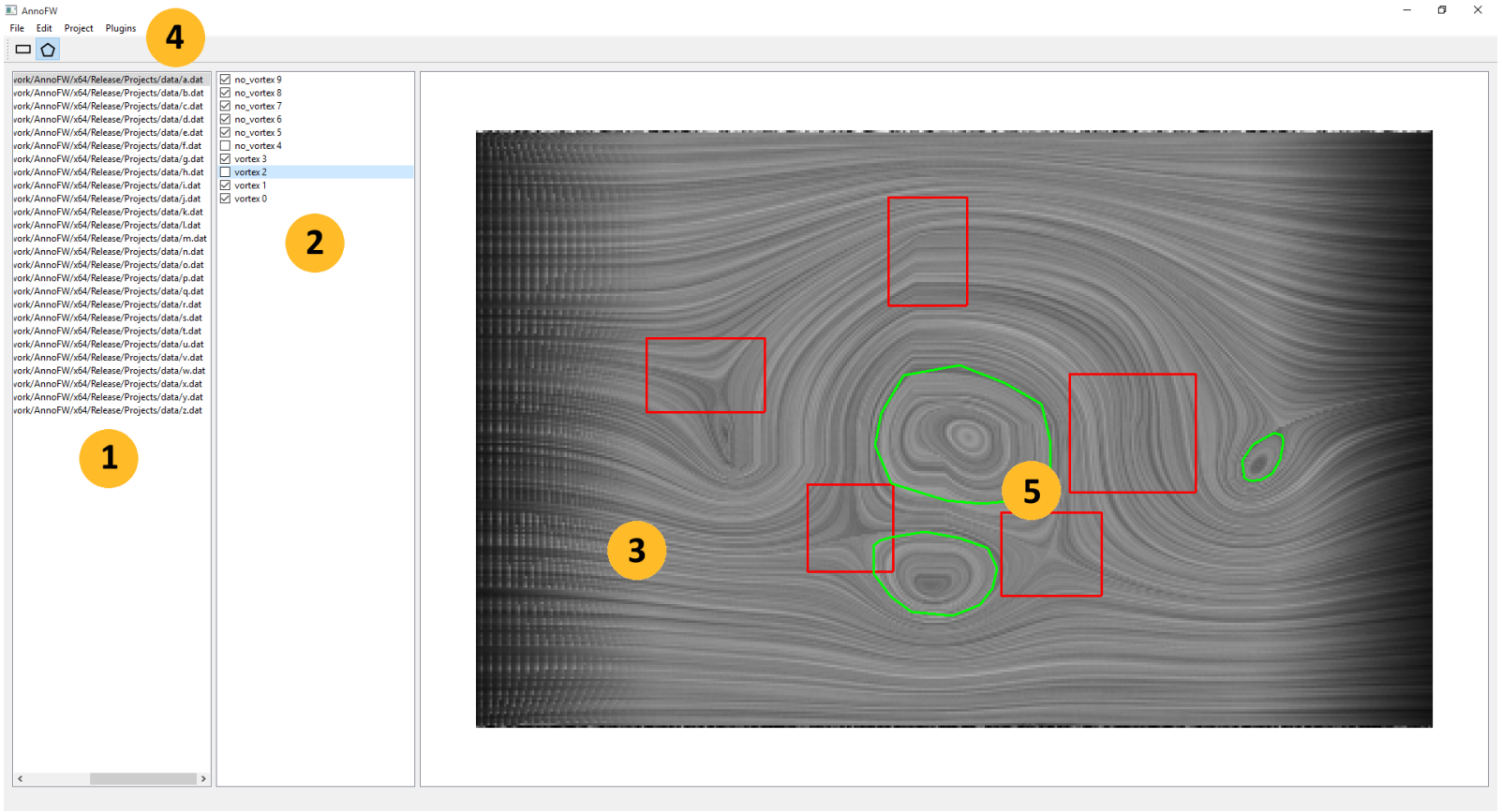

Abbildung 4.1: Benutzeroberfläche des Programms. 1: Übersicht der geladenen Dateien, 2: Verwaltung der annotierte Regionen, 3: Visualisierung der Daten, 4: Menüleiste und Auswahl der Selektionsmethode, 5: Annotierte Regionen

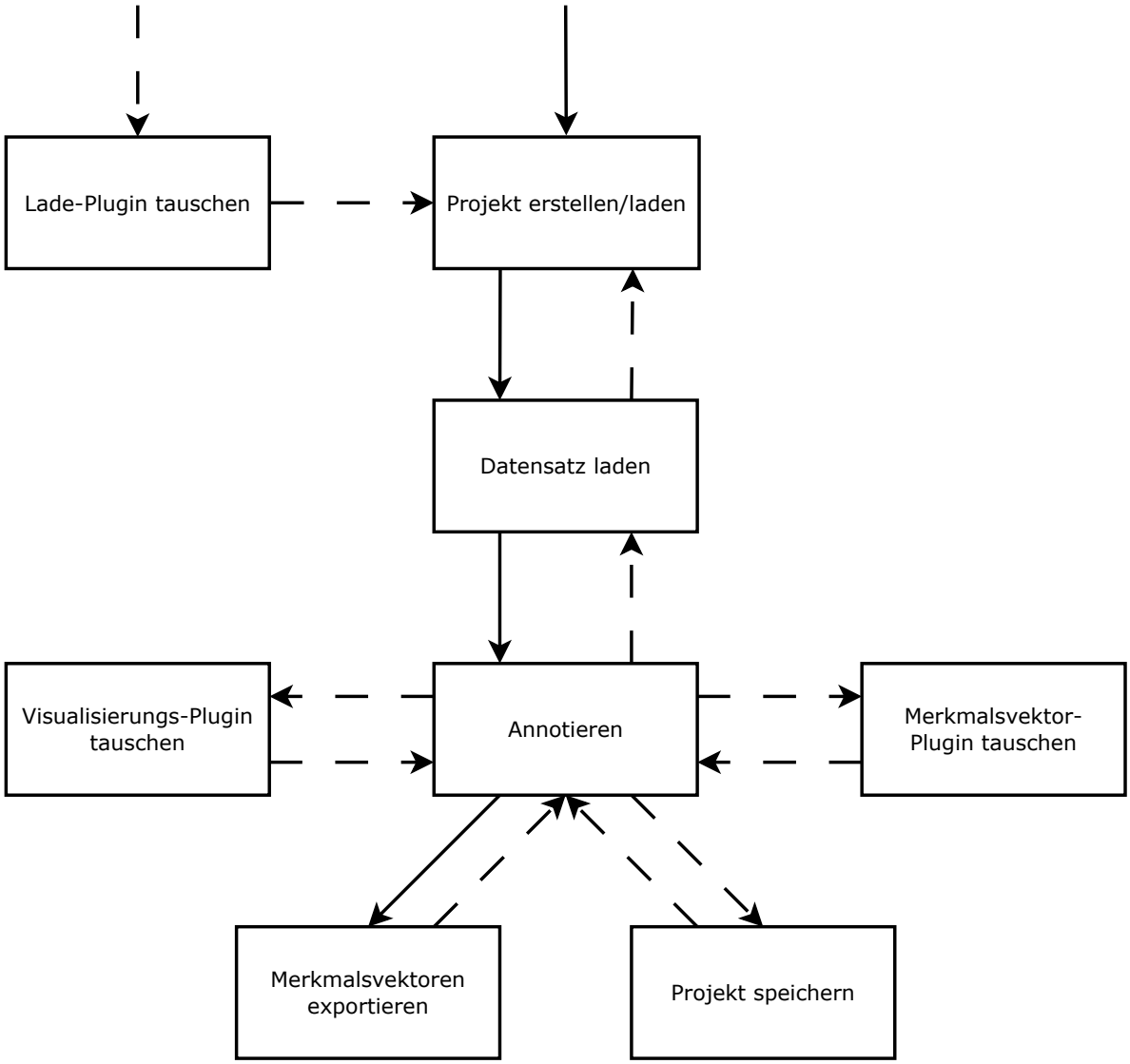

Abbildung 4.2: Arbeitsablauf. Gestrichelte Pfeile sind optionale Arbeitsschritte.

5 Fallstudie

Die Fallstudie gibt einen Einblick in die Arbeitsweise mit dem entwickelten System und bespricht damit ermittelte Ergebnisse für die Erkennung von Wirbeln.
Ziel ist, einen Satz von 26 Dateien mit Daten aus Strömungssimulationen mit Positiv- und Negativbeispielen für Wirbel zu annotieren. Das soll einmal mit rechteckigen und ein zweites mal mit allgemein polygonalen Regionen geschehen. Für die annotierten Bereiche sollen dann der histogrammbasierte und der spezifische Merkmalsvektor (siehe Abschnitt 4.4) exportiert werden.
Die Merkmalsvektorausgabedateien sollen dann verwendet werden, um mit Methoden des maschinellen Lernens Klassifikatoren zu trainieren und diese zu bewerten. Dafür kommt die Bibliothek Weka zum Einsatz [HFH+09].

Initialisieren des Projekts Bevor mit der eigentlichen Annotation begonnen werden kann, muss ein Projekt erstellt werden. Das explizite Öffnen eines neuen Projekts ist aber nur nötig, wenn ein anderes bereits geöffnet ist, da das System beim Start ein leeres Projekt erstellt. Die Vorbereitung beschränkt sich daher alleine auf die Auswahl der Daten.

Annotation der Regionen Sobald die Daten geladen sind, kann mit der Selektion relevanter Regionen begonnen werden. Im konkreten Fall bedeutet dies, dass in der Visualisierung sichtbare Wirbel möglichst genau durch Polygone eingegrenzt und anschließend mit einem entsprechenden Label versehen werden. Zusätzlich müssen Regionen markiert werden, in denen keine Wirbel sind, um die Lernverfahren mit Negativbeispielen zu versorgen. Das System bietet die Möglichkeit für die beiden Klassen kontrastierende Farben zu wählen, so dass der Nutzer stets den Überblick behalten kann. Zu sehen ist dies auch in Abbildung 4.1.

Speichern der Merkmalsvektorausgabedatei Ist die Annotation vollständig, speichert der Nutzer eine Merkmalsvektorausgabedatei. Der histogrammbasierte Standardmerkmalsvektor entspricht hier schon einem der gewünschten Vektoren. Um den zweiten, spezifischen Merkmalsvektor zu berechnen und zu speichern, tauscht der Nutzer lediglich das Plugin aus und lässt erneut eine Ausgabedatei speichern.

Annotation mit Rechtecken Um auf möglichst einfachem Weg vergleichbare Ergebnisse mit ausschließlich rechteckigen Regionen zu erhalten, speichert der Nutzer das Projekt für spätere Verwendungszwecke und beginnt dann, die annotierten Regionen zu ersetzen. Dazu markiert er jeweils eine möglichst ähnliche rechteckige Region und löscht dann die nicht mehr erwünschte polygonale Variante. Hat er dies mit allen Regionen gemacht, speichert er wie vorher die beiden unterschiedlichen Ausgabedateien.

Auswertung der Ausgabe Die Auswertung geschieht mit externen Programmen wie in den Abschnitten 5.1 und 5.2 beschrieben.

5.1 Cross Validation & Konfusionsmatrix

Die durch die beiden Beispiel-Plugins erstellten Ausgabedateien liegen im „Attribute-Relation File Format“ (.arff) vor. Dieses Format kann von Weka direkt gelesen werden. Um bei der Auswertung die Abhängigkeit von einem bestimmten Lernalgorithmus zu verhindern, wird jede der vier Ausgabedateien mit drei verschiedenen Methoden getestet:

Logistische Regression Die logistische Regression kommt mit einem Regularisierungsparameter von 10^{-8} zum Einsatz.

Random Forest Für das Random Forest Verfahren werden jeweils 100 Modelle verwendet. Die Erstellung erfolgt beim Histogramm-Merkmalsvektor unter Berücksichtigung von vier zufälligen Merkmalen, beim spezifischen Merkmalsvektor werden aufgrund der niedrigeren Anzahl an Merkmalen nur zwei zufällig ausgewählt.

AdaBoost AdaBoost kommt mit Entscheidungsbäumen als Klassifikatoren zum Einsatz. Wie schon bei den Random Forests werden jeweils 100 von ihnen verwendet.

Alle Kombinationen aus Ausgabedatei und Lernalgorithmus wurden mit Cross Validation getestet. Für den Parameter k wurde der Wert zehn gewählt, das heißt, es wurden jeweils neunzig Prozent als Trainings- und die restlichen zehn Prozent als Testdatensatz verwendet. Die Tabelle 5.1 enthält die Ergebnisse bei Verwendung des Histogramm-Merkmalsvektors, während Tabelle 5.2 die des spezifischen Vektors beinhaltet. Anstatt die Konfusionsmatrix für jeden Test abzubilden sind hier schon die in Kapitel 3 eingeführten Maße berechnet. Diese bieten eine bessere Vergleichbarkeit, da sie leichter interpretierbar sind.

Tabelle 5.1: Ergebnisse unter Verwendung des Histogramm-Vektors

	Polygon			Rechteck		
	Log. Regr.	R. Forest	AdaBoost	Log. Regr.	R. Forest	AdaBoost
KKR	90,14 %	94,84 %	94,84 %	89,91 %	93,43 %	93,19 %
FKR	9,86 %	5,16 %	5,16 %	10,09 %	6,57 %	6,81 %
TP-rate	0,95	0,97	0,96	0,94	0,94	0,95
FP-rate	0,14	0,07	0,06	0,14	0,07	0,09
Genauigkeit	0,85	0,92	0,93	0,85	0,92	0,90
F-Maß	0,90	0,95	0,95	0,90	0,93	0,93

Tabelle 5.2: Ergebnisse unter Verwendung des spezifischen Merkmalsvektors

	Polygon			Rechteck		
	Log. Regr.	R. Forest	AdaBoost	Log. Regr.	R. Forest	AdaBoost
KKR	89,91 %	90,85 %	89,67 %	86,62 %	86,15 %	84,04 %
FKR	10,09 %	9,15 %	10,33 %	13,38 %	13,85 %	15,96 %
TP-rate	0,92	0,91	0,90	0,90	0,87	0,84
FP-rate	0,12	0,10	0,10	0,16	0,14	0,16
Genauigkeit	0,87	0,89	0,88	0,83	0,84	0,82
F-Maß	0,89	0,90	0,89	0,86	0,85	0,83

Interpretation der Ergebnisse

Auch wenn die Ergebnisse durchweg als akzeptabel einzustufen sind, lassen sich deutliche Unterschiede zwischen den verwendeten Merkmalsvektoren, sowie zwischen der Verwendung von rechteckigen und der von allgemeinen polygonalen Regionen ausmachen.
Im Folgenden wird die Leistung der Merkmalsvektoren und die der verwendeten Regionsarten separat verglichen.

Vergleich der Merkmalsvektoren

Entgegen der intuitiven Erwartung zeigt der Merkmalsvektor auf Basis des Richtungshistogramms durchweg eine höhere Korrektklassifikationsrate als der eigens für Wirbel entworfene zweite Merkmalsvektor.
Besonders gravierend sind die Unterschiede in Kombination mit rechteckigen Regionen und Random Forest oder AdaBoost als Wahl des Lernalgorithmus. Der Gründe hierfür sind wahrscheinlich zweierlei Natur: Zum einen hat der spezifische Merkmalsvektor eine sehr

niedrige Dimension, was den Vorteil der Randomisierung bei Random Forests und der Gewichtung bei AdaBoost stark reduziert. Zum anderen lassen rechteckige Regionen mehr nicht zum eigentlichen Wirbel gehörende Störsignale zu, welche die Leistung der Klassifikation einschränken. Darauf geht der nächste Unterabschnitt genauer ein.

Vergleich der Regionsarten

Vergleicht man die Ergebnisse der Merkmalsvektoren, die mit allgemeinen polygonalen Regionen berechnet wurden, mit denen der rechteckigen Regionen, zeigt sich erwarteterweise eine klare Überlegenheit der Polygone. Sie zeigen mit beiden Merkmalsvektoren und allen Lernalgorithmen sowohl eine höhere Korrektklassifikationsrate, als auch eine höhere Genauigkeit und Sensitivität (TP-rate). Am deutlichsten fallen die Unterschiede aus, wenn der spezifische Merkmalsvektor verwendet wird.
Grund dafür ist mit hoher Wahrscheinlichkeit die schlechte Eignung von Rechtecken für die Eingrenzung von Wirbeln. Da Wirbel meist eine runde Form aufweisen, liegen in den Ecken der Rechtecke oft Bereiche, die eigentlich nicht mehr dem Wirbel zuzuordnen sind. Je nach Anteil dieser Störsignale an der Region können die berechneten Merkmale dadurch signifikant verfälscht werden. Besonders anfällig sind hier die Maße des Flusses und der Magnitude des Durchschnittsvektors. Da beide für Wirbel im Idealfall gegen null gehen, führen Störsignale bei relativer Betrachtung zu großen Änderungen.
Da beide Maße im spezifischen Merkmalsvektor verwendet werden, schneidet dieser dann auch in Kombination mit rechteckigen Regionen besonders schlecht ab.

5.2 Ergebnisse Moving Window

Eine Möglichkeit, die Leistung eines Klassifikators visuell darzustellen, ist das in Abschnitt 3.4.3 vorgestellte Moving Window Verfahren unter Verwendung der Prediction Ratio. Bei der Auswertung der verschiedenen Ausgabedateien wird deutlich, dass sich die Leistung nicht stark genug unterscheidet, um sie mit den Ergebnissen des Moving Window Verfahrens zu vergleichen.
Es zeigt sich aber auch, dass die trainierten Klassifikatoren die vorhandenen Wirbel größtenteils erkennen können. Ein Beispiel mit guter Detektionsleistung findet sich in Abbildung 5.1, während Abbildung 5.2 einen Datensatz zeigt, in dem die Methode wenig Rückschlüsse zulässt. Beide wurden unter Verwendung polygonaler Regionen, des Histogramm-Merkmalsvektors und Random-Forests erstellt. Die rote Färbung korreliert hier nicht linear mit der Prediction Ratio, sondern wird durch die Kubikwurzel ermittelt. Das hat zum Ziel, dass kleinere Bereiche, die aufgrund der verwendeten Fenstergrößen meist eine niedrigere Ratio aufweisen, nicht durch große Bereiche nahezu „unsichtbar“ gemacht werden.

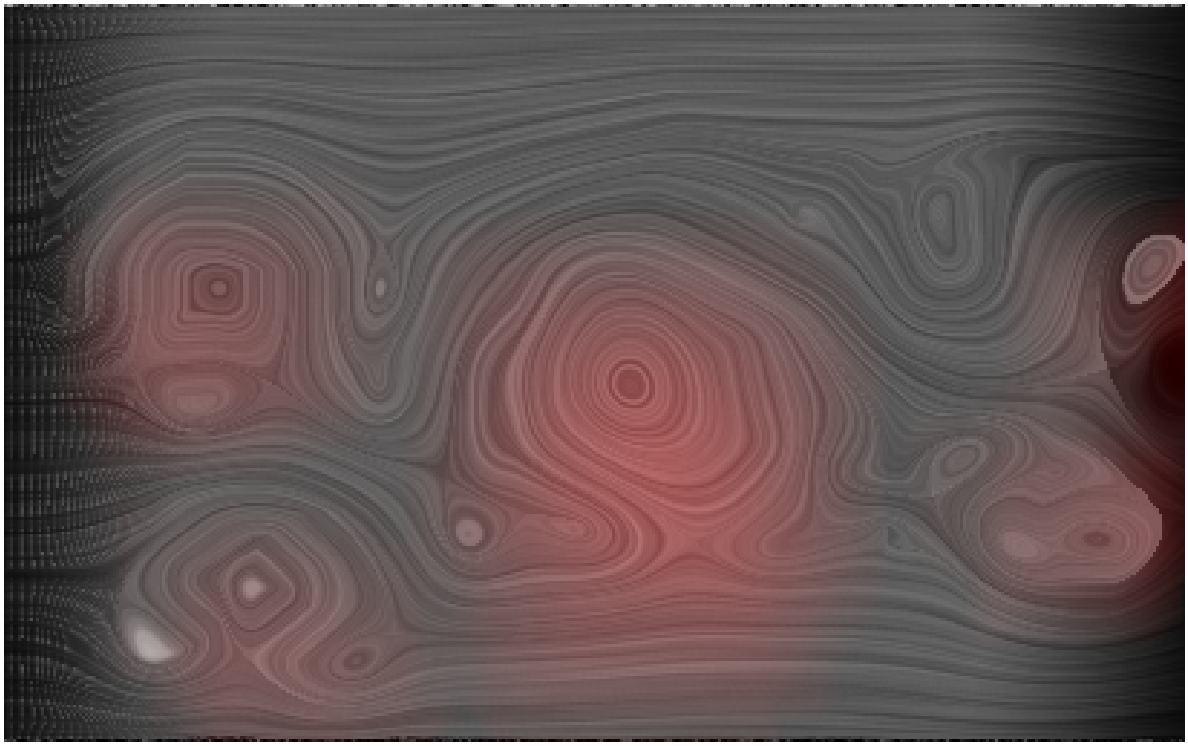

Abbildung 5.1: Ergebnis einer Anwendung des Moving Window Verfahrens. Der Klassifikator wurde mit Random Forests unter Verwendung polygonaler Regionen und des histogrammbasierten Merkmalsvektors trainiert.

Nach Einführung von Kriterien, um die Prediction Ratio in abgegrenzte Bereiche zu überführen, sollten die Klassifikatoren also im Stande sein, Wirbel in unbekannten Daten zu identifizieren.

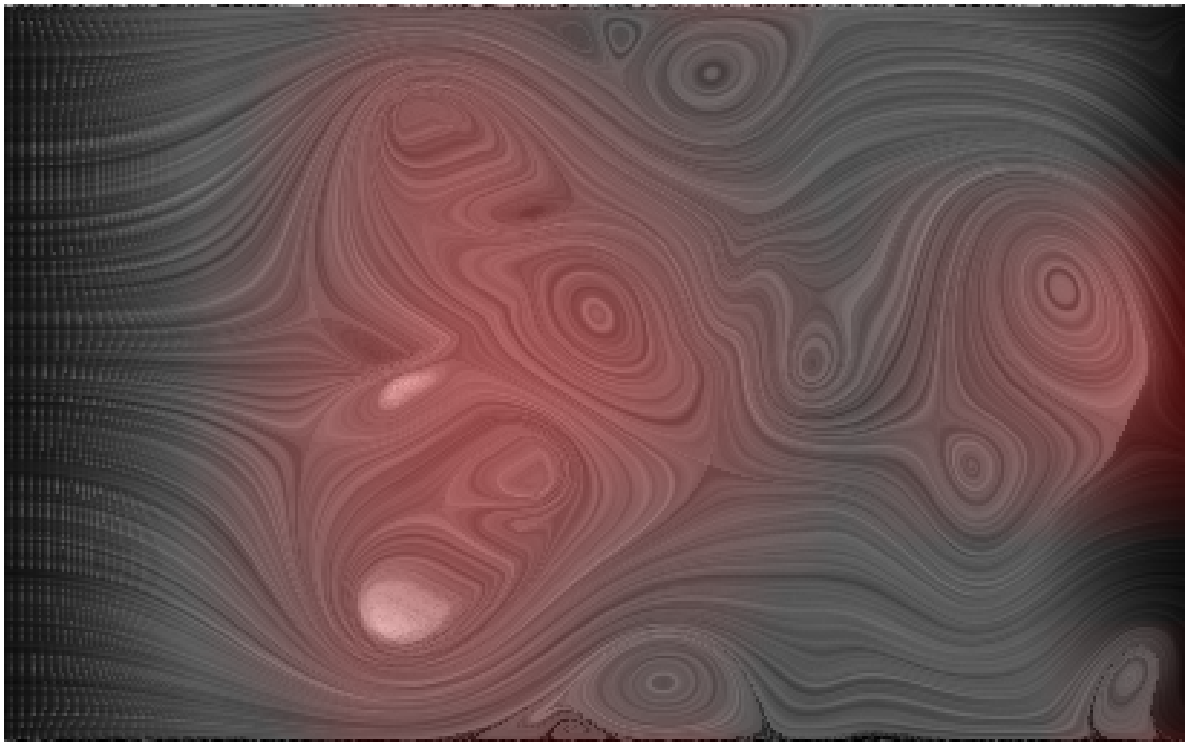

Abbildung 5.2: Ergebnis einer weiteren Anwendung des Moving Window Verfahrens. Die Trainingsbedingungen sind identisch zu den in Abbildung 5.1 beschriebenen.

6 Zusammenfassung und Ausblick

In dieser Arbeit wurde ein System zur Annotation von Skalar- und Vektorfelddaten vorgestellt. Die Ausgangslage war dabei der hohe zeitliche Aufwand, der nötig ist um Daten für überwachtes maschinelles Lernen zu generieren, sowie die durch rechteckige Selektionsregionen verschuldete Ungenauigkeit der Merkmalsvektoren.
Nach Einführung der zugrundeliegenden Algorithmen des maschinellen Lernens und einem Einblick in die Vektorfeldvisualisierung, die Merkmalsvektorkonstruktion und einige gängige Methoden zur Evaluation von Klassifikatoren wurde das entwickelte System vorgestellt.
Es arbeitet mit allgemeinen Polygonen zur Auswahl der Regionen und besitzt ein Plugin-System, das den einfachen Austausch des Lademoduls, des Visualisierungsverfahrens sowie des Moduls zur Erstellung einer Merkmalsvektorausgabedatei ermöglicht. Die Arbeitsabläufe bei der Annotation werden auf diese Art vereinfacht. Insbesondere der Zeitaufwand für das Testen und Vergleichen verschiedener Merkmalsvektoren sinkt, da für ein Annotationsprojekt mehrere Merkmalsvektorausgabedateien durch simples Austauschen des Plugins generiert werden können.
Im anschließenden Vergleich zeigte sich ein eindeutiger Vorteil der Verwendung von polygonalen Regionen. Es wurde auch deutlich, dass diese Überlegenheit vom verwendeten Merkmalsvektor und dem eingesetzten maschinellen Lernverfahren abhängig ist.

Ausblick

Das beschriebene System lässt Raum für einige Erweiterungen und Verbesserungen. So wäre es beispielsweise wünschenswert, auch dreidimensionale Daten annotieren zu können. Eine Aggregation kurzer Zeitreihen und die Möglichkeit eine solche zu annotieren wäre ebenso eine denkbare Erweiterung.
Es existieren bereits einige Verfahren, die automatisch Objekte extrahieren. Interessant wäre daher eine Kombination der manuellen Annotation mit automatisch erkannten Vorschlägen. Dies könnte auch als Plugin umgesetzt werden, um den Nutzer die volle Kontrolle zu ermöglichen.
Schließlich sehe ich noch die Möglichkeit, das Training der Klassifikatoren in die Anwendung zu integrieren und bei Hinzufügen neuer Annotationen live zu aktualisieren, um dem Nutzer Feedback über die Auswirkungen zu geben. Möglich wäre das zum Beispiel durch das Einbinden einer Bibliothek wie Weka.

Literaturverzeichnis

[Ber99] D. P. Bertsekas. *Nonlinear Programming: Second Edition*. Belmont, Massachusetts: Athena Scientific, 1999 (Zitiert auf S. 17).

[BFOS84] L. Breiman, J. H. Friedman, R. A. Olshen und C. J. Stone. *Classification and Regression Trees*. Belmont, CA: Wadsworth International Group, 1984 (Zitiert auf S. 17).

[CL93] B. Cabral und L. C. Leedom. „Imaging Vector Fields Using Line Integral Convolution“. In: *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '93. 1993, S. 263–270 (Zitiert auf S. 23).

[DANS10] J. Daniels II, E. W. Anderson, L. G. Nonato und C. T. Silva. *Interactive Vector Field Feature Identification*. Oktober 2010 (Zitiert auf S. 11, 22).

[FS97] Y. Freund und R. E. Schapire. „A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting“. In: *Journal of Computer and System Sciences* 55 (1997), S. 119–139 (Zitiert auf S. 18).

[HFH+09] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann und I. H. Witten. „The WEKA Data Mining Software: An Update“. In: *SIGKDD Explorations* 11 (2009) (Zitiert auf S. 37).

[Hic13] M. Hickins. *How the NSA Could Get So Smart So Fast*. The Wall Street Journal. Juni 2013 (Zitiert auf S. 14).

[Kip12] M. Kipp. „ANVIL: A Universal Video Research Tool“. In: *Handbook of Corpus Phonology*. Oxford University Press, 2012 (Zitiert auf S. 11).

[KSH] A. Krizhevsky, I. Sutskever und G. E. Hinton. *ImageNet Classification with Deep Convolutional Neural Networks* (Zitiert auf S. 20).

[MD] D. Mihalcik und D. Doermann. *The Design and Implementation of ViPER* (Zitiert auf S. 11).

[MD01] E. Mjolsness und D. DeCoste. „Machine Learning for Science: State of the Art and Future Prospects“. In: *Science* 293 (Sep. 2001), S. 2051–2055 (Zitiert auf S. 14).

[NA02] M. Nixon und A. Aguado. *Feature Extraction and Image Processing*. Butterworth Heinmann/ Newnes, Jan. 2002 (Zitiert auf S. 24).

[Ope] OpenCV. URL: http://opencv.org/ (Zitiert auf S. 29).

[Qt] Qt. URL: http://www.qt.io/ (Zitiert auf S. 29).

[RDMN15] D. C. Robles, P. Desenne, L. Muellner und G. Nagy. *Open Video Annotation Project*. http://www.openvideoannotation.org. 2015 (Zitiert auf S. 11).

[RTMF08] B. C. Russell, A. Torralba, K. P. Murphy und W. T. Freeman. „LabelMe: a datadata and web-based tool for image annotation". In: *International Journal of Computer Vision* 77.1–3 (Mai 2008), S. 157–173 (Zitiert auf S. 11).

[Vap82] V. Vapnik. *Estimation of Dependences Based on Empirical Data: Springer Series in Statistics (Springer Series in Statistics)*. 1982 (Zitiert auf S. 17).

[Var14] H. R. Varian. „Big Data: New Tricks for Econometrics". In: *Journal of Economic Perspectives* 28.2 (2014), S. 3–28 (Zitiert auf S. 14).

[Wei09] D. Weiskopf. „Iterative Twofold Line Integral Convolution for Texture-Based Vector Field Visualization". In: *Mathematical Foundations of Scientific Visualization, Computer Graphics, and Massive Data Exploration*. Mathematics and Visualization. 2009, S. 191–211 (Zitiert auf S. 23).

[ZDM+14] L. Zhang, Q. Deng, R. Machiraju, A. Ragarajan, D. Thompson, D. Walters und H.-W. Shen. „Boosting Techniques for Physics-Based Vortex Detection". In: *Computer Graphics Forum* 33.1 (2014), S. 282–293 (Zitiert auf S. 11).

Alle URLs wurden zuletzt am 30. 10. 2015 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift