Ruprecht-Karls-Universität Heidelberg

Hochschule Heilbronn

Studiengang Medizinische Informatik

Bachelorarbeit

Entwurf und Implementierung einer Benutzeroberfläche
zur Klassifikation von Mustern

Konstantin Kinzel

22. September 2013

Referent: Prof. Dr. Rolf Bendl, Hochschule Heilbronn

Korreferent: Prof. Dr. Martin Haag, Hochschule Heilbronn

Betreuer: Dipl.-Inform. Med. Andrea Fränzle, DKFZ Heidelberg

Eidesstattliche Erklärung

Ich erkläre hiermit, dass ich die vorliegende Bachelorarbeit selbstständig und unter der ausschließlichen Verwendung der angegebenen Literatur und Hilfsmittel verfasst habe.

Die vorliegende Arbeit wurde keiner anderen Prüfungsbehörde vorgelegt.

(Konstantin Kinzel)

Heidelberg, 22. September 2013

Entwurf und Implementierung einer Benutzeroberfläche zur Klassifikation von Mustern

Zusammenfassung - In dieser Arbeit wurde am Deutschen Krebsforschungszentrum (DKFZ) in Heidelberg eine Anwendung für das Trainieren von Klassifikatoren, das Klassifizieren und für die Durchführung eines Leave-One-Out-Tests entwickelt. Auf der Grundlage von Methoden aus einem bereits existierenden Programm, das zum automatischen Erkennen von osteolytischen und osteoblastischen Läsionen bei Patienten mit einem multiplen Myelom genutzt wird, wurde eine Benutzeroberfläche implementiert, die das aufwendige und manuelle Anpassen an neue Fragestellungen erleichtern soll. Zusätzlich soll eine Visualisierung von Merkmalen dem Benutzer beim Trainieren eines Klassifikators die Auswahl verdeutlichen und vereinfachen. Damit die Anwendung auch für andere Klassifikationsaufgaben genutzt werden kann, wurde eine Erweiterungsmöglichkeit der grafischen Benutzeroberfläche auf Basis von XML integriert.

Design and implementation of a graphical user interface for pattern recognition

Abstract - This thesis was written in cooperation with the German Cancer Research Center (DKFZ) in Heidelberg. It approaches the design and development of an application to train classifiers, the classification itself, as well as the execution of a Leave-One-Out-Test. Based on methods of an existing program which is used to automatically detect osteolytic and osteoblastic lesions of patients with a multiple myeloma, a user interface was implemented to ease the effortful and manual adjustment to new problems. Furthermore, a visualization of features should clarify and simplify the choices of the user to train the classifiers. To use the application for other tasks of classification, a possible enhancement of the user interface was integrated on XML basis.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

DKFZ ..... *Deutsches Krebsforschungszentrum*
GUI ..... *Graphical User Interface*
UI ..... *User Interface*
XML ..... *Extensible Markup Language*

1. Einleitung

Die Erkennung und Klassifikation anatomischer und pathologischer Strukturen ist eine grundlegende Aufgabe der medizinischen Bildverarbeitung. Klassifikationsverfahren versuchen auf Basis von Intensitätswerten und -verteilungen Pixel oder Voxel eines Bildes zu bestimmten Strukturen oder Gewebeklassen zuzuordnen. In der Abteilung Medizinische Physik in der Strahlentherapie am DKFZ Heidelberg wurden in einem Forschungsprojekt Klassifikationsverfahren entwickelt, um osteolytische und osteoblastische Läsionen bei Patienten mit einem multiplen Myelom (Plasmozytom) automatisch erkennen zu können. Dazu wurde eine Bibliothek mit unterschiedlichen Algorithmen erstellt, die für die verschiedenen Klassifikationsaufgaben zusammengefasst und konfiguriert wurde.

Klassifikations- und Mustererkennungsverfahren sind prinzipiell universell konzipiert und werden auch in vielen anderen Anwendungsgebieten, wie zum Beispiel der Spracherkennung oder der Biometrie eingesetzt. Die Verfahren müssen aber für die jeweiligen Anwendungsgebiete konfiguriert und trainiert werden, bevor sie ihre Aufgabe erfüllen können. Diese Anpassungen sind häufig mit sehr viel Aufwand verbunden, da der Programmcode händisch in eine Konsole oder einen Eingabebereich eingegeben und ausgeführt werden muss. Um die existierenden Verfahren leichter an neue Aufgabenstellungen anpassen zu können, soll eine grafische Benutzeroberfläche entwickelt werden, die den Vorgang der manuellen Änderung ersetzen soll und stattdessen einen gewissen Automatismus umsetzen und die Arbeit, wie zum Beispiel die Parametrisierung von Funktionen oder das Auswählen einer Datei über einen Browser erleichtern soll. Mit Hilfe von Visualisierungen sollen zusätzlich abstrakte Daten in vereinfachter Form dargestellt werden. Eine der weiteren zentralen Rollen stellt die Erweiterbarkeit dar. Diese soll außerhalb einer Entwicklungsumgebung möglich sein, damit andere Klassifikationsaufgaben das Programm nutzen können.

Für die Entwicklung solcher Graphical User Interfaces (GUI) gibt es spezielle GUI Designer, wie zum Beispiel QT, welche der GUI-Erstellung passende Elemente anbieten, um die Entwicklung zu vereinfachen. Das in dieser Arbeit verwendete Framework und die vertiefenden Grundlagen der Mustererkennung werden in Kapitel 2 erläutert. Mustererkennung fasst in diesem Zusammenhang Gegenstände mit gleichartigen Eigenschaften zu Gruppen zusammen. Im darauffolgenden Kapitel wird auf die Ziele und Anforderungen der Anwendung eingegangen und es wird exemplarisch ein Arbeitsablauf durchgeführt. Daran anschließend können zusätzliche Informationen über die Implementierung erhalten werden. Zum Abschluss wird in Kapitel 4 eine Diskussion und ein Ausblick auf zukünftige Erweiterungen gegeben.

1.1. Hintergrund

In dieser Arbeit wurde eine GUI für Aufgabenstellungen in der Mustererkennung entwickelt. Für die Mustererkennung existiert eine große Anzahl an Anwendungsgebieten, wie zum Beispiel die Texterkennung oder die Iriserkennung in der Biometrie. Das für diese Arbeit entwickelte Programm findet Einsatz in der medizinischen Bildverarbeitung.

Mustererkennung wird in der medizinischen Bildverarbeitung verwendet, um Aufgaben, beispielsweise die Zuordnung von Transversalschichten aus computertomographischen Daten zu einzelnen Körperregionen, zu bearbeiten. Diese Schichtklassifikation setzt jedoch bestimmte Eigenschaften, wie zum Beispiel die Körperbreite voraus. Diese wird zur Einteilung verwendet [1][2]. Weitere Anwendungsbeispiele können im Werk [3] nachgeschlagen werden.

Damit Aufgaben, wie die Schichtklassifikation, auch in annehmbarer Zeit gelöst werden können, werden Programme entwickelt, die die Mustererkennung vereinfachen und automatisieren.

Ein ähnliches Programm wurde bereits in der Studienarbeit von Andrea Fränzle in MATLAB implementiert [4].

1.2. Problemstellung und Motivation

Zum automatischen Erkennen von Läsionen bei Patienten mit einem multiplen Myelom wurden in einem Forschungsprojekt Algorithmen für Klassifikationsverfahren entwickelt und in einem Programm umgesetzt. Da sich das Anpassen des Programmes als umständlich und aufwändig erweist, soll dieses um eine Benutzeroberfläche, auf Basis der bereits bestehenden Algorithmen, erweitert werden. Dabei ist darauf zu achten, dass die Implementierung möglichst dynamisch ist und eine leichte Erweiterbarkeit gewährleistet wird.

Der Ansporn für die Entwicklung einer grafischen Benutzeroberfläche ist die Zeitersparnis für den Benutzer, die Visualisierung der Ergebnisse und die Verbesserung der Benutzbarkeit der Methoden. Dabei spielen Punkte wie die Erweiterbarkeit, d.h. die Anpassung an eine weiterführende Problemstellung mit einem möglichst geringen Aufwand, auch eine wichtige Rolle.

2. Material und Methoden

In den folgenden Unterkapiteln werden die Grundlagen der Mustererkennung und die erforderlichen Begriffe erläutert. Hierbei wird unter anderem auf die Phasen und die verschiedenen Verfahren der Mustererkennung eingegangen. Anschließend folgt eine Beschreibung des QT Frameworks und dessen Komponenten.

2.1. Begriffserklärung

Merkmal

Merkmale sind die Eigenschaften eines Objektes, die zur Einteilung der Objekte zu einer Klasse gegeben sein müssen.

Klasse

Eine Klasse ist eine Zusammenfassung von Objekten auf Grund von gleichartigen Eigenschaften.

Muster

Mit einem Muster werden ein oder mehrere Signale in Zusammenhang gebracht, welche für den Mediziner interessant sind. Ein Muster besteht entweder aus einer Struktur oder einem Trend (Veränderung einer Struktur über einen Zeitraum) [5]. Ein Beispiel hierfür wäre ein QRS-Komplex eines Herzens, welcher durch ein Elektrokardiogramm abgebildet wird.

Klassifikation

Unter Klassifikation versteht man die Zuordnung eines Musters zu einer bestimmten Klasse.

2.2. Grundlagen der Mustererkennung

In diesem Abschnitt wird die Mustererkennung vorgestellt. Hierzu werden in Kapitel 2.2.1 allgemeine Grundlagen und die dazugehörenden Phasen der Mustererkennung näher erläutert. In den darauffolgenden Kapiteln wird zu Anfang die Fehlerrate und im Anschluss einige Verfahren, die für eine Klassifikation verwendet werden können, behandelt.

2.2.1. Mustererkennung

Mustererkennung reicht von der Identifikation von einem Muster, bis hin zur Zuordnung von einem Muster zu einer Klasse. Identifikation stellt in diesem Zusammenhang das Auffinden eines Elements mit bestimmten Eigenschaften unter

mehreren Elementen dar. Die Mustererkennung kann unterschiedlichen Zwecken dienen. Diese wären zum Beispiel: Klassifikation, Erkennung oder Abschätzung von Merkmalen [6]. Unter einer Klassifikation versteht man das Zusammenschließen von Gegenständen/Objekten durch Eigenschaften (Merkmale/Attribute) zu bestimmten gleichartigen Gruppen (Klassen).

Exemplarisch für das Trainieren und Klassifizieren eines Musters kann man ein Kleinkind nehmen. Bereits ein Kleinkind kann Form, Farbe und Größe unterscheiden. Diese Fähigkeit wird im Laufe des Wachstums von klein auf trainiert und immer weiter perfektioniert. Wenn einem Kind ein Feuerwehrauto gezeigt wird, kann man dem Kind sagen, dass dies ein Feuerwehrauto ist, da es rot ist und Blaulicht hat. Sollte ein weiteres rotes Auto vorbeifahren, kann man durch das Fahrzeug erklären, dass es kein Feuerwehrauto ist, da es kein Blaulicht besitzt. Nach mehrfachen Wiederholungen ist das Kleinkind soweit trainiert, dass es ein Feuerwehrauto von einem normalen Auto, mit einer gewissen Fehlerrate, selbstständig unterscheiden kann und somit durch die trainierten Eigenschaften (Farbe und Blaulicht) in einem begrenzten Maße klassifizieren und somit die Fahrzeuge in unterschiedliche Gruppen einteilen kann.

Überträgt man dieses Beispiel nun auf Klassifikations- und Mustererkennungsverfahren in der Forschung, verhält sich das Trainieren und Klassifizieren ähnlich, jedoch mit der Problematik, dass zum einen die Klassifikationskriterien vorab nicht bekannt sind und dass die Merkmale in der Regel in der Trainingsphase auf Basis bekannter richtiger Zuordnungen ermittelt werden, nach denen Klassifikationen mit einem möglichst geringen Fehler erfolgen können.

2.2.1.1. Phasen der Mustererkennung

Um sicherzustellen, dass ein Training ausreichend war und das Klassifikationsverfahren zuverlässig arbeitet, bedarf es einiger Testdurchläufe.

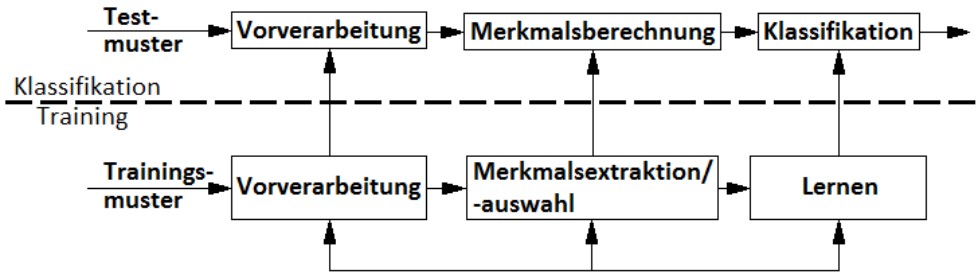


Abbildung 1: Schematische Darstellung von Mustererkennungsphasen (nach [5])

Wie in Abbildung 1 zu erkennen ist, besitzen sowohl der Trainings-, als auch der Testmodus die Vorverarbeitung. Die Vorverarbeitung ist ein wichtiger Teilschritt. Es werden Störungen entfernt, um die Bildqualität zu verbessern, das Muster wird normalisiert und es findet eine Präzisierung des Musters statt. Im Trainingsmodus werden nach der Vorverarbeitung die Merkmale der Muster festgelegt und der

Klassifikator wird mit den Merkmalen trainiert. Anschließend können im Testmodus die Muster mit den Merkmalen klassifiziert werden, um die Zuverlässigkeit der ermittelten Merkmale zur Klassifikation neuer Daten zu bestimmen. In Abhängigkeit der erreichten korrekten Klassifikationsrate müssen die vorangegangenen Schritte wiederholt werden, bis ein zufriedenstellendes Klassifikationsergebnis erreicht ist [5].

2.2.2. Klassifikatoren

In der Mustererkennung gibt es unterschiedliche Klassifikationsprobleme. In diesem Absatz wird eine Auswahl an Verfahren näher vorgestellt. Es werden zwei unterschiedliche Verfahrensarten unterschieden: Verfahren die zur Klassifizierung dienen und Verfahren die zur Ermittlung der Fehlerrate eines Klassifikators verwendet werden.

2.2.2.1. Klassifikationsverfahren

Da es eine Vielzahl an Klassifikationsverfahren gibt, werden in diesem Unterkapitel nur die für diese Arbeit interessantesten Verfahren vorgestellt.

Nächster-Nachbar-Verfahren

Bei dem Nächster-Nachbar-Verfahren spielt der nächste Nachbar bzw. die nächsten Nachbarn eine wichtige Rolle. Bei diesem Verfahren wird unterschieden zwischen dem (einen) nächsten Nachbarn und den k-nächsten Nachbarn. Ein neues Objekt wird der Klasse zugeordnet, welche durch das (die) Nachbarobjekt(e) bestimmt ist, das die ähnlichste(n) Merkmalausprägung(en) besitzt.

Entscheidungsbäume

Dieses sehr schnelle Verfahren trennt Objekte gemäß ausgewählter Merkmale der Reihe nach auf. Dieser Vorgang wird solange durchgeführt, solange bis alle Elemente in Blattendpunkten enden. Ein Beispiel mit verschiedenen „Ball“-Typen wird in Abbildung 2 verdeutlicht.

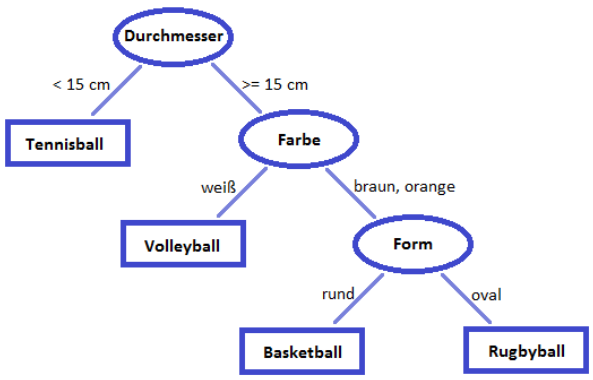


Abbildung 2: Beispiel für einen Entscheidungsbaum

Random Forest

Das von L. Breiman [7] vorgestellte Verfahren verwendet n-Entscheidungsbäume anstelle von nur einem einzelnen Klassifikator. Hierbei werden n-Entscheidungsbäume erstellt. Der Aufbau der Entscheidungsbäume findet nach bestimmten Regeln statt und verwendet festgelegte Parameter. Exemplarisch für den Aufbau eines solchen Entscheidungsbaumes wurde aus [1] ein Beispiel zur Erstellung von einem Binärbaum entnommen.

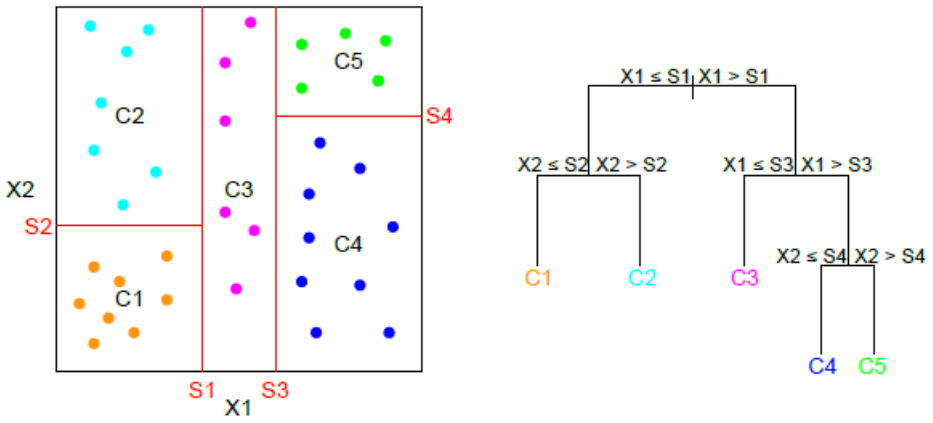


Abbildung 3: Beispiel für den Aufbau eines Entscheidungsbaumes in Random Forest [1]

Jeder dieser Entscheidungsbäume liefert über das zu klassifizierende Muster ein Klassifizierungsergebnis, was dazu führt, dass n-Klassifizierungsergebnisse pro Muster vorhanden sind. Daraus wird dem zu klassifizierenden Muster das am häufigsten vorkommende Klassifizierungsergebnis zugewiesen. Ein ausführliches Beispiel für die Anwendung von Random Forest in der medizinischen Bildverarbeitung findet man in [1].

2.2.2.2. Verfahren zur Ermittlung der Fehlerrate

Die Fehlerrate, welche durch diese Verfahren abgeschätzt wird, ermöglicht das Bewerten eines Klassifikators. Hierbei wird der Klassifikator mit Daten bzw. Objekten, deren Klassenzugehörigkeit bekannt ist, die aber möglichst nicht für das Training des Klassifikators verwendet wurden, getestet und das Ergebnis mit den tatsächlich klassifizierten Klassen verglichen. Die Fehlerrate wird berechnet, indem die Anzahl der falsch klassifizierten Ergebnisse durch die Anzahl der abgeglichenen Klassen geteilt wird. Je niedriger die Fehlerrate, umso besser ist ein Klassifikator. Nachfolgend werden drei Verfahren zur Ermittlung der Fehlerrate erläutert, wobei das zweite Verfahren, die Leave-One-Out-Methode in dieser Arbeit zum Einsatz kommt.

Holdout

Bei dem Holdout-Verfahren werden die Datensätze in Trainings- und Testdaten eingeteilt. Der eine Teil der Datensätze wird verwendet, um den Klassifikator zu

trainieren und der andere Teil wird dann auf Basis des Trainings klassifiziert. Dieses Verfahren ist nur bei einer großen Anzahl von Datensätzen möglich.

Leave-One-Out

Die Leave-One-Out-Methode, auch Jackknife genannt, ist ein Verfahren zur Ermittlung der Fehlerrate eines Klassifikators. Dieses Verfahren ist ein Spezialfall der Kreuzvalidierung. Hierbei wird für jeden Datensatz ein Durchlauf ausgeführt, bei dem jeweils ein Datensatz aus der Datensatzmenge zum Testen herausgenommen und mit den restlichen Datensätzen trainiert wird.

Leave-K-Out

Dieses Verfahren wird auch Kreuzvalidierung genannt. Zu Beginn werden die Datensätze in Mengen von k-Elementen aufgeteilt. Anschließend werden für die Anzahl der Teildatensatzmengen Testdurchläufe durchgeführt, wobei immer eine andere Teildatensatzmenge zum Trainieren und die restlichen Datensatzmengen zum Testen genommen werden.

Nachdem eine Übersicht über die unterschiedlichen Verfahren und Methoden und somit die Funktionen, die in dieser Bachelorarbeit in der Implementierung genutzt wurden, gegeben wurde, folgt im Kapitel 2.3 eine Beschreibung zur Entwicklung der eigentlichen Oberfläche und der verwendeten Werkzeuge.

2.3. Oberflächenentwicklung in C++

Eine Benutzeroberfläche lässt sich auf unterschiedliche Arten realisieren. Da in der Abteilung Medizinische Physik in der Strahlentherapie am DKFZ Heidelberg, vorwiegend Visual Studio als Entwicklungsumgebung verwendet wird, wurde diese Entwicklungsumgebung in dieser Arbeit ebenfalls eingesetzt, um mögliche Konflikte beim späteren Nutzen und Erweitern zu vermeiden. Zusätzlich wurde das Framework QT verwendet. Es bietet mit Komponenten, wie dem QT Designer eine Möglichkeit, Benutzeroberflächen zusammenzustellen oder detailliertere Konfigurationen, wie zum Beispiel das Nutzen des Signal-/Slot-Konzepts, durchzuführen (2.3.3). Da eine Internationalisierung zwar nicht gefordert, aber für diese Arbeit angemessen erschien, bot QT mit dem QT Linguist eine ideale Umsetzungsmöglichkeit. Die Benutzung von QT Linguist wird in Kapitel 2.3.4 dargelegt und im Ergebnisteil beim Workflow aufgezeigt.

Im Folgenden wird die Realisierung mit dem Framework QT und deren Komponenten vorgestellt.

2.3.1. QT Framework

QT ist eine C++-Klassenbibliothek, die zur Entwicklung von Anwendungen und Benutzeroberflächen genutzt wird. Hierbei bietet QT C++-Entwicklern die Möglichkeit, mit Hilfe des QT Designers einfach und schnell Benutzeroberflächen zusammenzustellen. Von Vorteil für den Entwickler ist, dass vergleichbar wenig Wissen bezüglich des Hintergrunds der grafischen Benutzeroberflächen benötigt wird. In dieser Arbeit wurde QT als zusätzliche Erweiterung in Visual Studio 2010 verwendet.

QT stellt eine eigene Entwicklungsumgebung zur Verfügung. Um QT Komponenten in anderen Entwicklungsumgebungen, wie zum Beispiel Visual Studio, verwenden zu können, braucht es ein Add-In.

2.3.2. QT Visual Studio Add-In

Mit Hilfe dieser Erweiterung ist es möglich in der gewählten Entwicklungsumgebung QT Komponenten zu benutzen. Ein zusätzlicher Menüpunkt in der Leiste von Visual Studio erlaubt den direkten Zugriff auf die unterschiedlichen QT Komponenten, wie zum Beispiel den QT Linguist (2.3.4) oder den QT Designer.

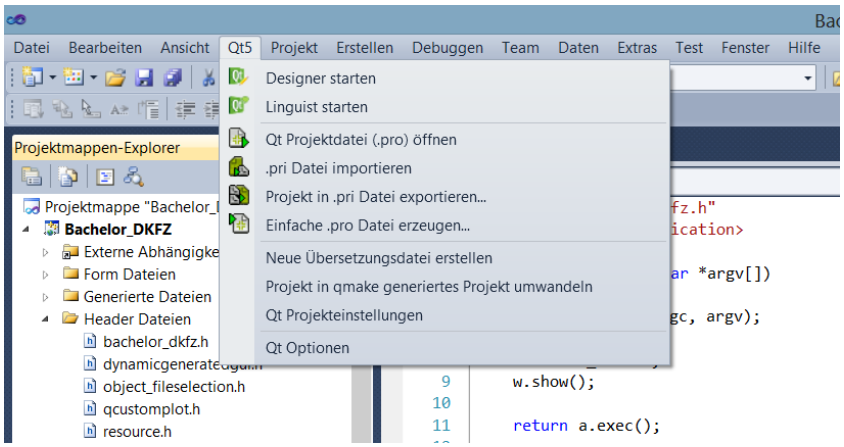


Abbildung 4: Visual Studio mit QT Add-In

2.3.3. QT Designer

In diesem Designer lassen sich Benutzeroberflächen zusammensetzen ohne manuelles Schreiben des Programmcodes (Abbildung 5). Um im Nachgang eine Oberfläche zu bearbeiten, kann die gespeicherte Benutzeroberfläche mit dem QT-Designer wiederhergestellt und modifiziert werden. Für die Feinabstimmung lässt sich der Programmcode auch manuell ändern. Dies ist für erfahrene Programmierer ein wertvolles Tool, da die Präzisierung im QT Designer meist sehr mühevoll ist.

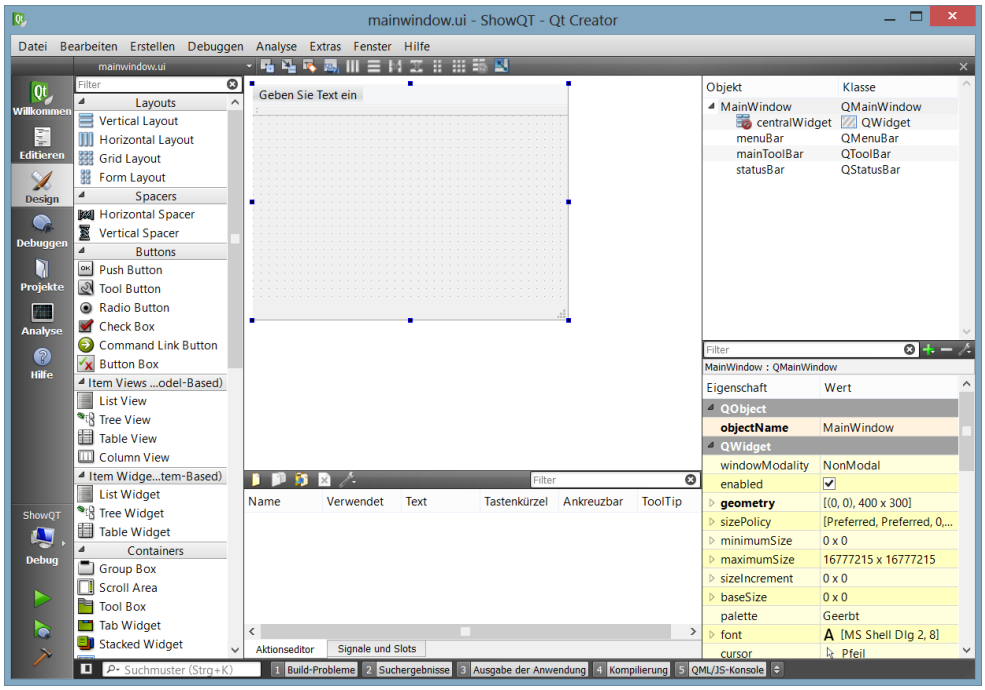


Abbildung 5: QT Designer Arbeitsoberfläche

Wenn Funktionen oder das von QT bereitgestellte und in der Anwendung verwendete Signal-/Slot-Konzept genutzt werden soll, so werden diese außerhalb von dem QT Designer implementiert. Das Signal/Slot-Konzept dient dazu Benutzeraktionen (Signale, z.B. das Drücken einer Schaltfläche) mit einer Aktion (Slot, z.B. eine Programmfunktion) zu verknüpfen. Nähere Ausführungen zum Signal/Slot-Prinzip sind in der ausführlichen Dokumentation von QT [8] vorhanden.

Funktionen für die Oberflächenelemente, sowie weitere Klassen und Methoden für die Anwendung, können in Visual Studio implementiert und erweitert werden (Abbildung 6).

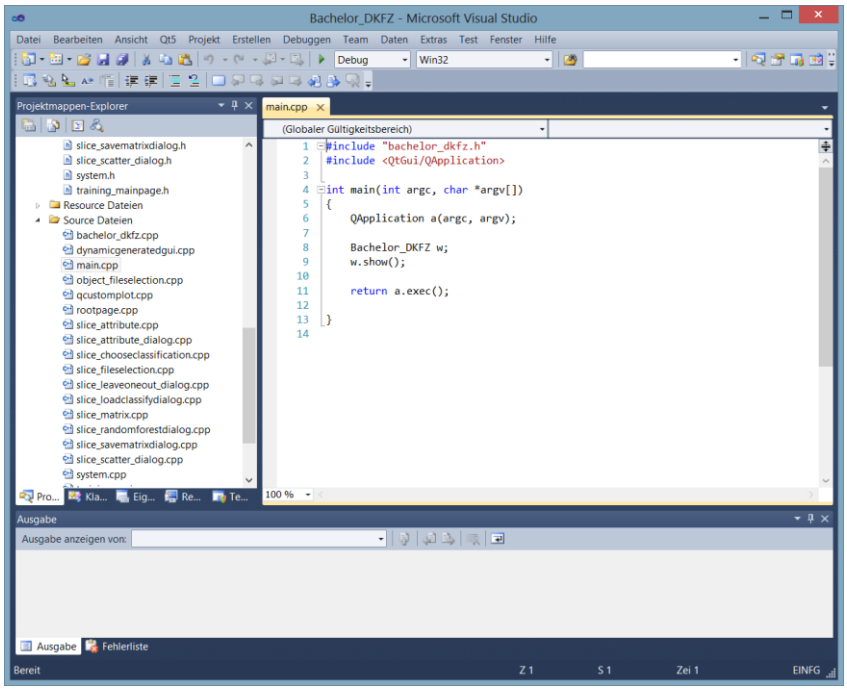


Abbildung 6: Visual Studio 2010 Ansicht

Weitere Informationen und ein umfassendes Handbuch zum QT Designer können auf der Webseite [9] nachgelesen werden.

2.3.4. QT Linguist

QT Linguist ist ein Werkzeug zur Ergänzung einer Internationalisierung. Mit QT Linguist ist es möglich, die vom Anwender in QT entwickelte Applikation in andere Sprachen zu übersetzen und während des Betriebs die Sprache zu wechseln. Im ersten Arbeitsschritt werden durch den *lupdate*-Befehl alle zu übersetzenden Texte herausgefiltert. Anschließend werden sie dem Benutzer zur weiteren Verarbeitung bereitgestellt. (siehe Abbildung 7)

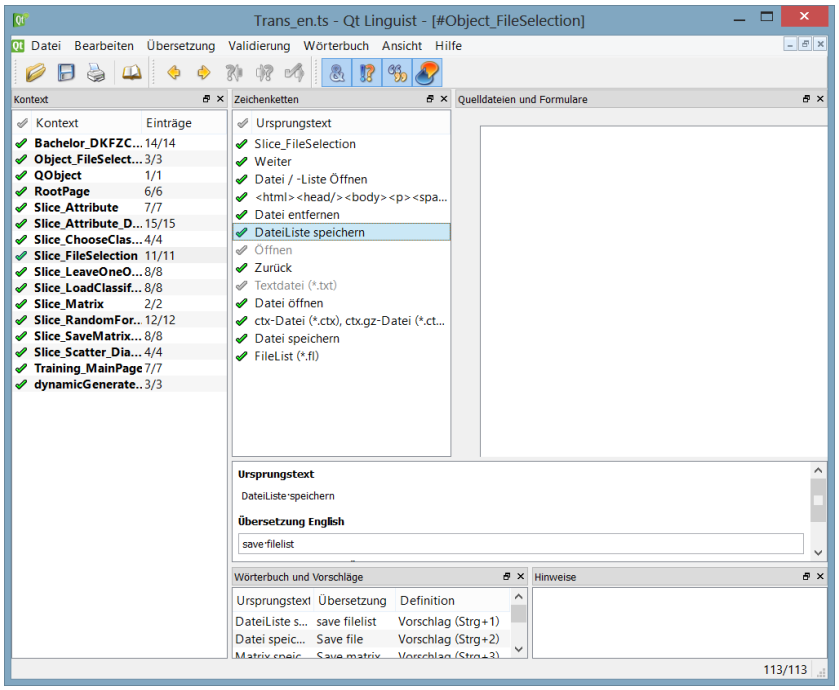


Abbildung 7: QT Linguist Arbeitsoberfläche

Der Benutzer vergibt für jeden Text eine Übersetzung. Mit dem *lrelease*-Befehl wird eine Übersetzungsdatei erstellt, die wiederum von QT eingelesen und verarbeitet werden kann. Anschließend kann die Übersetzung in einer Klasse bzw. Methode verwendet werden. Genauere Informationen und eine ausführliche Dokumentation zu QT Linguist sind auf [10] zu finden.

3. Ergebnisse

Das Ergebnis dieser Arbeit besteht aus einer C++-Anwendung und entstand auf Grundlage eines existierenden Programmes, welches zum automatischen Erkennen von osteolytischen und osteoblastischen Läsionen bei Patienten mit einem Plasmozytom genutzt wird. Das bereits vorhandene Programm basiert auf einer Bibliothek, welches in einem Forschungsprojekt am DKFZ Heidelberg erstellt wurde. Dieses wurde in dieser Bachelorarbeit mit einer Benutzeroberfläche erweitert, damit die Nutzung der Bibliothek effizienter, einfacher und mit weniger Aufwand verbunden ist.

Um die Aufgabe näher zu präzisieren, wird im folgenden Unterkapitel die Problematik in einer Anforderungsanalyse formuliert (3.1). In Anknüpfung an die Grundlagen der Anforderungsanalyse werden die für diese Arbeit erstellten Benutzeroberflächen und die damit verbundenen Funktionen erläutert. Hierzu werden die grafischen Oberflächen mit Hilfe eines Beispiel-Workflows vorgestellt (3.2). Dem Kapitel 3.2 folgend, wird die Implementierung und Architektur in Form von Sequenzdiagrammen und UML-Klassendiagrammen dargelegt. Im Anschluss wird die Erweiterbarkeit und deren Benutzung beschrieben (3.4).

3.1. Anforderungsanalyse

In den folgenden Kapiteln wird eine Anforderungsanalyse für das benötigte System ausgearbeitet. Im ersten Abschnitt (3.1.1) wird festgehalten, welche Personen Einfluss auf die Anforderungen oder das System haben (Stakeholder). Der nächste Punkt der Anforderungsanalyse erfragt die Ziele, die von den Stakeholdern angestrebt werden. Anschließend folgt ein Use Case, welches mit Hilfe eines Use Case-Diagramms die Systemaufgaben näher bringt (3.1.3). Abschließend werden in Kapitel 3.1.4.1 und 3.1.4.2 die funktionalen und nicht-funktionalen Anforderungen festgelegt.

3.1.1. Stakeholder

Als Stakeholder für diese Arbeit standen mir die Doktorandin Andrea Fränzle und der Arbeitsgruppenleiter Prof. Dr. Rolf Bendl, der Arbeitsgruppe Therapieplanung - Entwicklung der Abteilung Medizinische Physik in der Strahlentherapie am DKFZ Heidelberg zur Verfügung. Diese beiden Personen waren ausschlaggebend für die Anforderungen, Ziele und Funktionen, die für diese Anwendung verwirklicht wurden.

3.1.2. Ziele

Die Aufgabenstellung umfasst den Entwurf und die Entwicklung einer Benutzeroberfläche, deren Ziel es ist, die Arbeit mit bereits implementierten Verfahren zur Mustererkennung zu erleichtern und eine Möglichkeit zu bieten, generierte Daten zu veranschaulichen. Diese Klassifikationsverfahren wurden im

Rahmen von einem Forschungsprojekt in der Abteilung Medizinische Physik am DKFZ Heidelberg entwickelt und in einem Programm umgesetzt. Da der Anpassungsaufwand des Programms an neue Fragestellungen hoch ist, soll ein Graphical User Interface erstellt werden, um die Adaption zu vereinfachen und das Programm anderen Klassifikationsaufgaben zugänglich zu machen. Die bereits vorhandenen Klassen und Methoden, die die Programmlogik beinhalten, sollen in die grafische Benutzeroberfläche eingebunden und mit benötigten Funktionen erweitert werden, damit ein fehlerfreies Arbeiten gewährleistet ist. Zusätzlich soll eine Visualisierung bestimmter Attribute verwirklicht werden.

3.1.3. Use Case

Der Anwendung sollen drei verschiedene Modi zugrunde liegen, die unterschiedliche Aufgaben erfüllen. Diese Modi werden in der folgenden Abbildung 8 dargestellt.

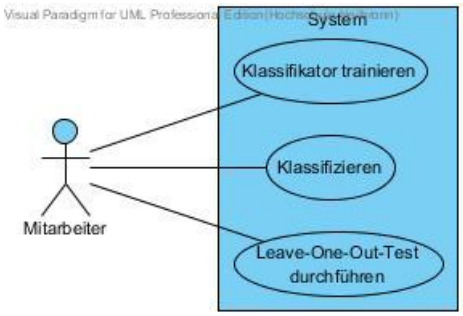


Abbildung 8: Anforderungs-Use Case

Jeder dieser Modi soll ein Graphical User Interface besitzen und durch zielgerichtete Interaktion mit dem System, sollen die jeweiligen Aufgaben erledigt werden können.

3.1.4. Anforderungen

Der folgende Abschnitt beschäftigt sich mit den funktionalen und den nicht-funktionalen Anforderungen für das System. Im ersten Teil werden die gewünschten Funktionalitäten und das Verhalten definiert. Der zweite Unterabschnitt beschäftigt sich mit den nicht-funktionalen Anforderungen, d.h. mit einigen Vorgaben zur Implementierung, wie zum Beispiel die zu nutzenden Werkzeuge.

3.1.4.1. Funktionale Anforderungen

Die Anwendung soll zum Klassifizieren, zum Trainieren von Klassifikatoren und zur Durchführung eines Leave-One-Out-Tests genutzt werden. Der Leave-One-Out-Test dient hierbei zur Überprüfung der Brauchbarkeit von Klassifikatoren und der Klassifizierungen. In jedem dieser Modi, muss eine Mindestanzahl an Datensätzen geladen und verarbeitet werden. Hierbei sollte gewährleistet sein, dass Datensätze in einem gewissen Format (*.ctx) aufgerufen werden können. Exemplarisch muss eine Klassifikationsart (Schichtklassifikation) und ein Klassifikationsverfahren (Random

Forest) implementiert werden. Da bei der Verwendung der Klassifikationsverfahren Parameter definiert werden müssen, soll der Anwender diese in angemessener Form eingeben können, sodass das Verfahren durchgeführt werden kann. Damit nach Abschluss des Trainierens des Klassifikators die trainierten Merkmale besser ausgewertet werden können, sollen diese visualisiert und miteinander verglichen werden.

Eine wichtige Anforderung stellt die Erweiterbarkeit dar. Diese soll realisiert werden, um die Anwendung einfach und ohne Entwicklungsumgebung an verschiedene Klassifikationsaufgaben anpassen zu können. Hierbei sollen die Merkmale, welche beim Trainieren eines Klassifikators und beim Leave-One-Out-Test zur Verfügung stehen, dynamisch erweiterbar sein. Falls die hinzugefügten Merkmale bestimmte Parameter benötigen, sollen diese automatisch beim Auswählen abgefragt und anschließend verarbeitet werden.

3.1.4.2. Nicht-funktionale Anforderungen

Da sich in der Abteilung Medizinische Physik in der Strahlentherapie die Programmiersprache C++ etabliert hat, ist eine der Voraussetzungen die Implementierung dieser Anwendung in C++. Passend zu der zur Verfügung gestellten Bibliothek und der Programmiersprache C++, empfiehlt sich das Benutzen der Entwicklungsumgebung Visual Studio. Dies erlaubt ein fehlerfreies Anbinden und Benutzen der Bibliothek.

Darüber hinaus wurde das Framework QT für die Entwicklung der Benutzeroberflächen nahe gelegt, da sich dieses Framework und deren Komponenten bereits in früheren Bachelor- und Studienarbeiten als zeitgemäß und nutzbringend herausgestellt haben.

3.2. Grafische Benutzeroberfläche

Um die Arbeitsweise des Programms zu verdeutlichen, wird ein Beispielarbeitsablauf durchgeführt. Mit Hilfe von diesem Beispiel-Workflow werden die unterschiedlichen Modi, ausgehend von dem Startbildschirm, erklärt. Da sich die Modi nur in einigen Punkten voneinander unterscheiden, werden im Unterkapitel "Klassifizieren" (3.2.3) und "Leave-One-Out-Test" (3.2.4) nur die Unterschiede grafisch belegt. Damit die Anwendung nicht nur von deutschsprachigen Personen genutzt werden kann, wurde in Kapitel 3.2.5 eine Internationalisierung erarbeitet.

3.2.1. Startbildschirm

Beim Starten der Anwendung zeigt sich der folgende Startbildschirm.

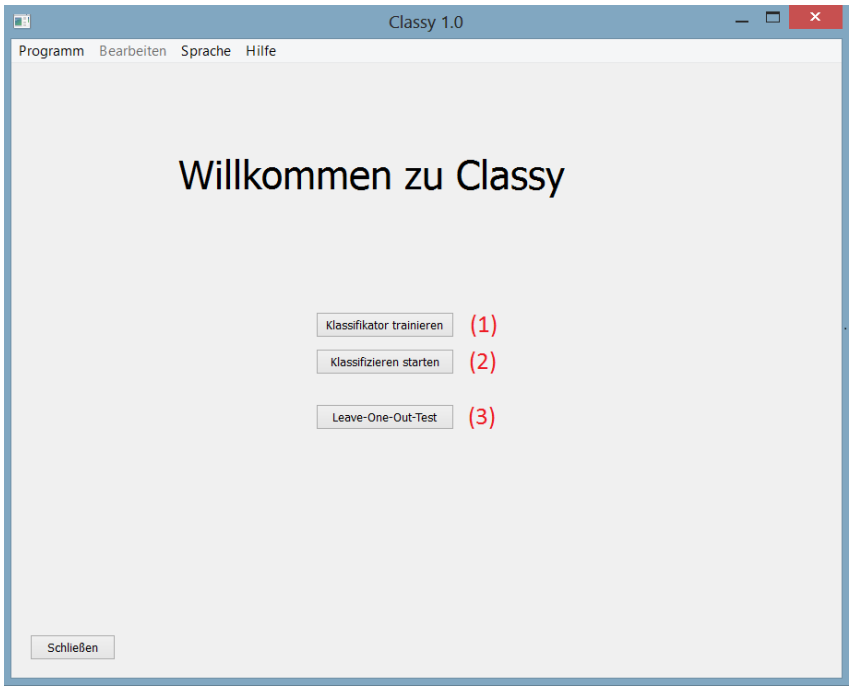


Abbildung 9: Startbildschirm der Anwendung

Die Anwendung besteht aus drei Modi: Klassifikator trainieren (1), Klassifizieren (2) und dem Leave-One-Out-Test (3). Die Modi werden in den kommenden Unterkapiteln näher präzisiert. Abhängig von dem aktuellen Modus werden die Oberflächen an den Modus angepasst.

Von dieser Oberfläche können alle Modi betreten werden. Sobald man einen der Modi fertiggestellt hat oder abbricht, gelangt man wieder zum Anfangsbildschirm zurück.

Um einen Modus zu verlassen oder die Anwendung neu zu starten, wird in der Menüleiste erst der Menüpunkt "Programm" und dann "Startseite" ausgewählt (Abbildung 10). Die Menüleiste steht in der Anwendung in allen Hauptoberflächen zur Verfügung.

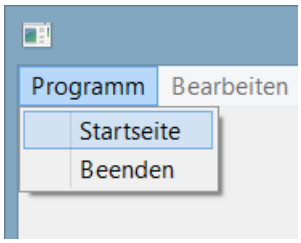


Abbildung 10: Untermenü Programm

3.2.2. Klassifikator trainieren

Um einen Klassifikator zu trainieren, muss man den Modus "Klassifikator trainieren" auswählen. Das Fenster zum Auswählen von einem der drei Klassifikatoren erscheint.

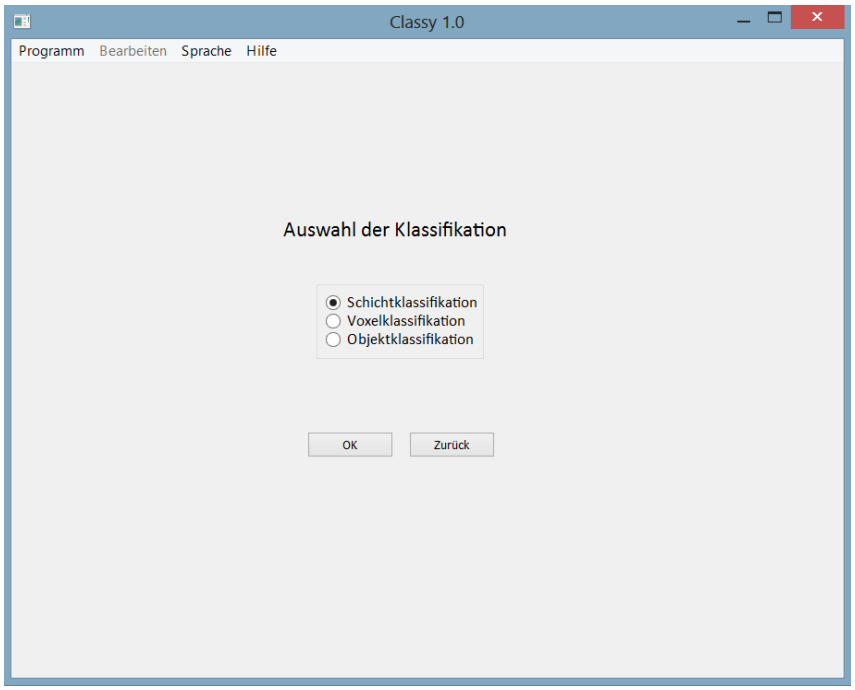


Abbildung 11: Auswahl der Klassifikation

Standardmäßig ist die Schichtklassifikation selektiert, da diese exemplarisch im Rahmen dieser Arbeit implementiert wurde. Die anderen Klassifikationsarten sind äquivalent zu implementieren.

Wird durch das Auslösen des "ok"-Buttons fortgefahren, werden die Datensätze geladen. In Abbildung 12 wurden, durch die "Datei / -Liste öffnen"-Funktion, bereits mehrere Daten geladen. Zusätzlich existiert eine Funktion "Dateiliste speichern". Eine Dateiliste besteht aus einer oder mehreren Dateien. Falls mehrere Datensätze in die Auswahl geladen wurden und beim nächsten Programmstart nicht wieder einzeln geladen werden sollen, kann man die komplette Liste speichern und beim Wiederaufruf über "Datei / -Liste öffnen" die Liste laden.

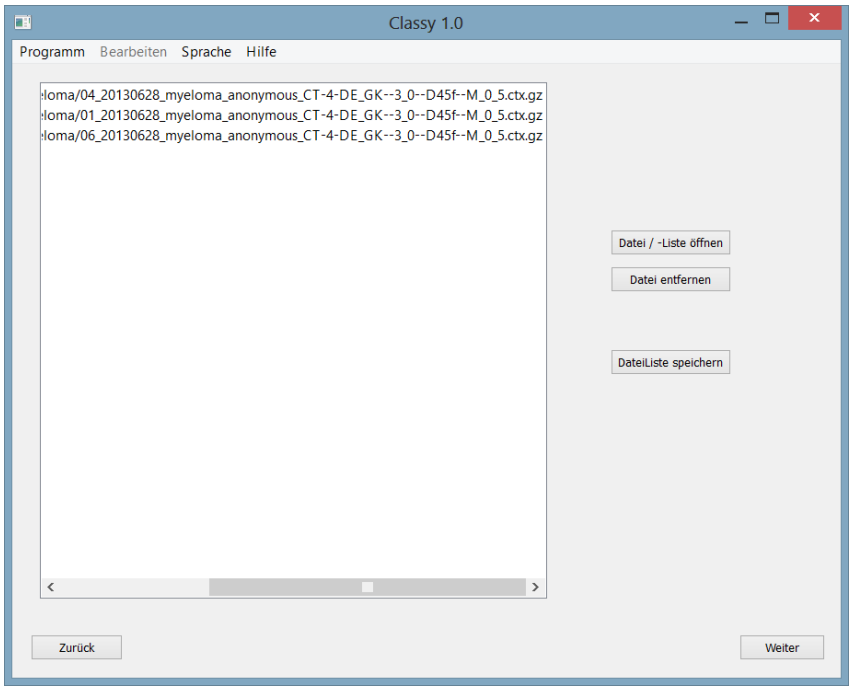


Abbildung 12: Fenster zum Laden der Datensätze

Der Benutzer gelangt nun zur Auswahl der Merkmalsliste. Auf dieser Oberfläche lassen sich Merkmale auswählen und hinzufügen (Abbildung 13). Einige Merkmale benötigen eine zusätzliche Eingabe, welche durch ein Dialogfenster mit den benötigten Spezifizierungen realisiert wurde (Abbildung 14). Eine Besonderheit dieser Dialoge ist die Erweiterbarkeit, die in dem Unterkapitel 3.4 näher angesprochen wird.

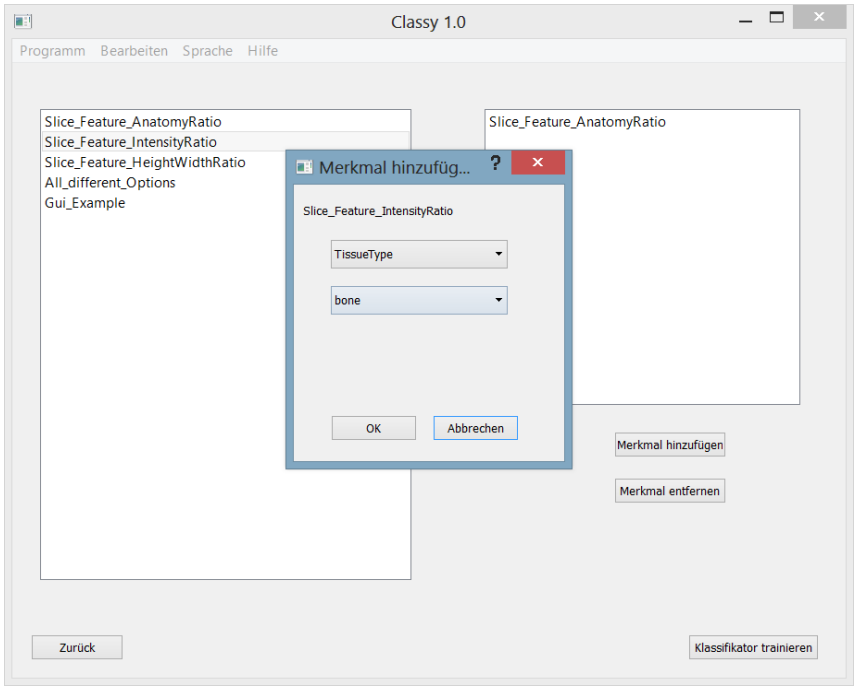


Abbildung 13: Merkmale hinzufügen/entfernen

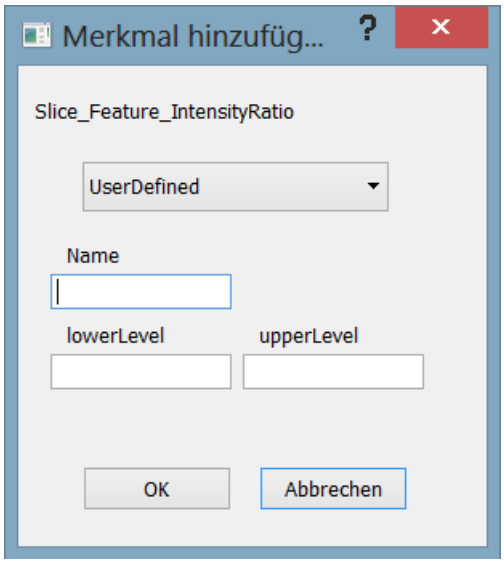
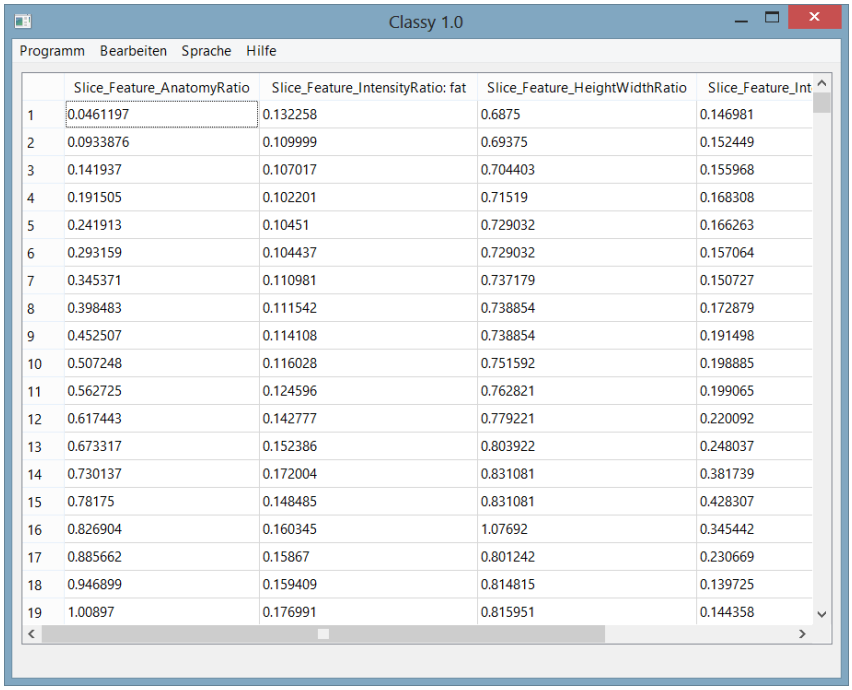


Abbildung 14: Benutzerdefinierte Merkmalsergänzung

Nach Eingabe der gewünschten Merkmale erscheint eine Tabelle mit allen trainierten Merkmalen, wobei der Index (= erste Spalte) dem Musterindex entspricht.



	Slice_Feature_AnatomyRatio	Slice_Feature_IntensityRatio: fat	Slice_Feature_HeightWidthRatio	Slice_Feature_Int
1	0.0461197	0.132258	0.6875	0.146981
2	0.0933876	0.109999	0.69375	0.152449
3	0.141937	0.107017	0.704403	0.155968
4	0.191505	0.102201	0.71519	0.168308
5	0.241913	0.10451	0.729032	0.166263
6	0.293159	0.104437	0.729032	0.157064
7	0.345371	0.110981	0.737179	0.150727
8	0.398483	0.111542	0.738854	0.172879
9	0.452507	0.114108	0.738854	0.191498
10	0.507248	0.116028	0.751592	0.198885
11	0.562725	0.124596	0.762821	0.199065
12	0.617443	0.142777	0.779221	0.220092
13	0.673317	0.152386	0.803922	0.248037
14	0.730137	0.172004	0.831081	0.381739
15	0.78175	0.148485	0.831081	0.428307
16	0.826904	0.160345	1.07692	0.345442
17	0.885662	0.15867	0.801242	0.230669
18	0.946899	0.159409	0.814815	0.139725
19	1.00897	0.176991	0.815951	0.144358

Abbildung 15: Merkmalsmatrix im Modus "Klassifikator trainieren"

Über den Menüpunkt "Bearbeiten" kann der Benutzer weitere Funktionen aufrufen. Die Funktionen sind in Abbildung 16 zu sehen.

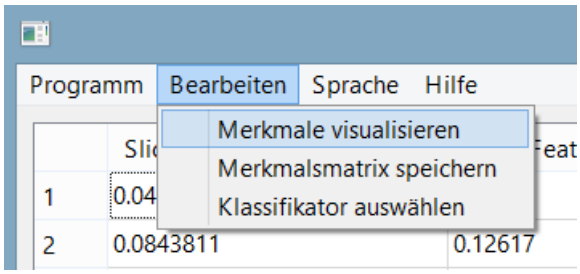


Abbildung 16: Bearbeiten-Menüpunkt und folgende Aktionen

Will der Benutzer Merkmale veranschaulichen, kann er dies über den Unterpunkt "Merkmale visualisieren" erreichen. Es öffnet sich ein neues Fenster mit gelisteten Merkmalen auf der rechten und einem leeren Graphen auf der linken Seite. Zur Darstellung muss der Benutzer genau zwei Merkmale auswählen und es wird pro Achse ein Merkmal mit den Werten aus der vorherigen Merkmalsmatrix gesetzt. Die Farben stellen die unterschiedlichen Klassen der jeweiligen Muster dar, wobei durch mehrfaches Drücken des "anzeigen"-Buttons die Farbe der Punkte verändert werden kann, um eine gute Unterscheidungsmöglichkeit der Klassen zu bewirken.

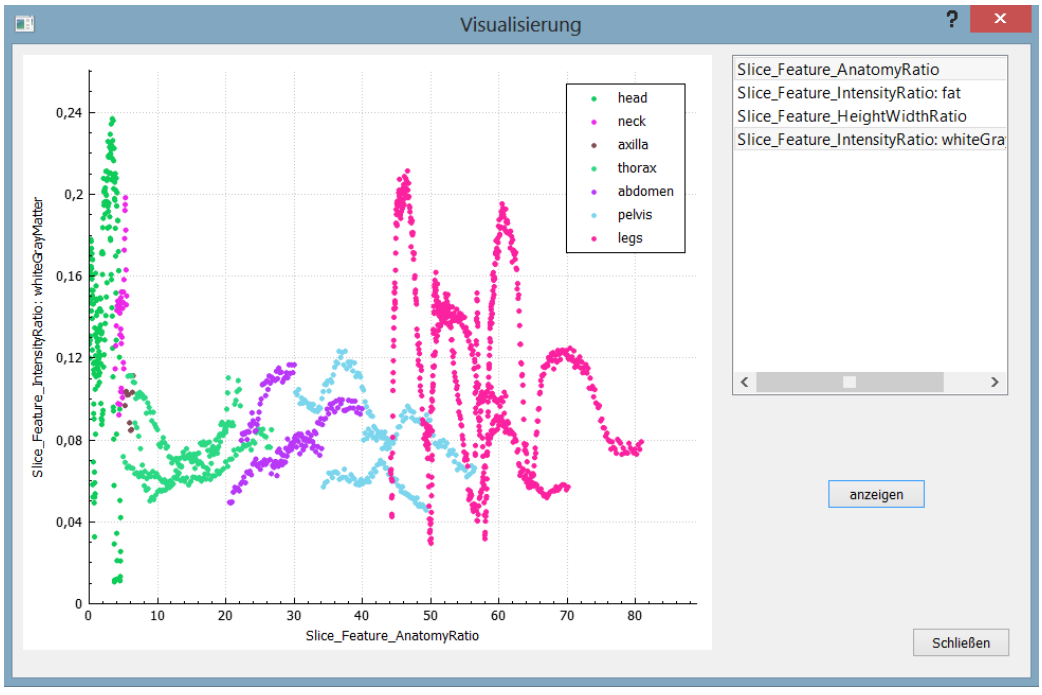


Abbildung 17: Merkmalsvisualisierung

Mit diesem Hilfsmittel kann der Benutzer erkennen, welche Merkmale geeignet sind, die Objekte möglichst gut in Klassen aufzuteilen. Die Auswahl geeigneter Merkmale ist notwendig, um die Klassifikation auf die Merkmale zu beschränken, die die höchste Trennschärfe aufweisen. Damit wird zum einen der Aufwand der Klassifikation reduziert, da weniger Merkmale betrachtet werden müssen. Zum anderen wird die Klassifikation genauer, da das Gesamtergebnis nicht durch unspezifische Teilergebnisse verfälscht wird.

Nach dem Schließen des Visualisierungs-Dialogs kann der Benutzer den Klassifikator auswählen. Zuvor muss die Merkmalsmatrix über den Untermenüpunkt "Merkmalsmatrix speichern" abgespeichert und der Speicherort festgelegt werden (Abbildung 18).

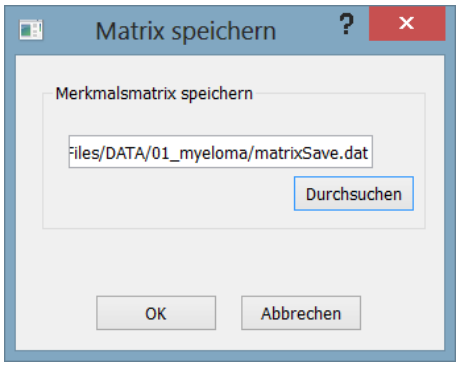


Abbildung 18: "Merkmalsmatrix speichern"-Dialog

Vorausgesetzt die Matrix wurde erfolgreich abgespeichert, lässt sich die Klassifikationsmethode festlegen (Abbildung 19). Im Rahmen dieser Arbeit wurde ein Random Forest-Klassifikator [7] in die Benutzeroberfläche integriert. Weitere Klassifikatoren sind entsprechend nachzuimplementieren.

Abbildung 19: Auswahl der Klassifikationsart

In dem neuen Fenster muss der Benutzer die Parameter für den Klassifikator festlegen. Bei dem Random Forest-Klassifikator sind dies die Anzahl der Bäume („number of trees“) und Anzahl der geprüften Merkmale für jeden Baumknoten („number of splits“). Diese Eingaben dürfen nur Zahlen enthalten. Eine Falscheingabe ist durch Validierung ausgeschlossen. Sollte der Benutzer sich dazu entscheiden keinen Speicherort des Klassifikators festzulegen, wird automatisch der Ordner genommen, in dem die Merkmalsmatrix abgesichert wurde und es wird ein Name mit den vorher festgehaltenen Werten generiert.

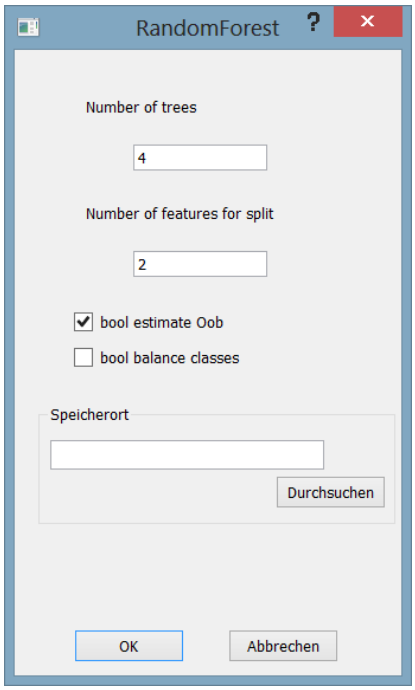


Abbildung 20: Random Forest Klassifikation

Nach erfolgreichem Speichern erscheint eine Erfolgsmeldung und der Benutzer wird zur Startseite weitergeleitet.

3.2.3. Klassifizieren

Über die Startseite wird der zweite Modus „Klassifizieren starten“ ausgewählt. Daraufhin gelangt der Benutzer direkt in das Datenladefenster. Wenn nicht mindestens ein Datensatz geladen wurde, erscheint eine Fehlermeldung, die zum Laden von mindestens einem Datensatz auffordert. Sind die Datensätze erfolgreich geladen, öffnet sich ein Dialog der zum Laden eines Klassifikators veranlasst (Abbildung 21).

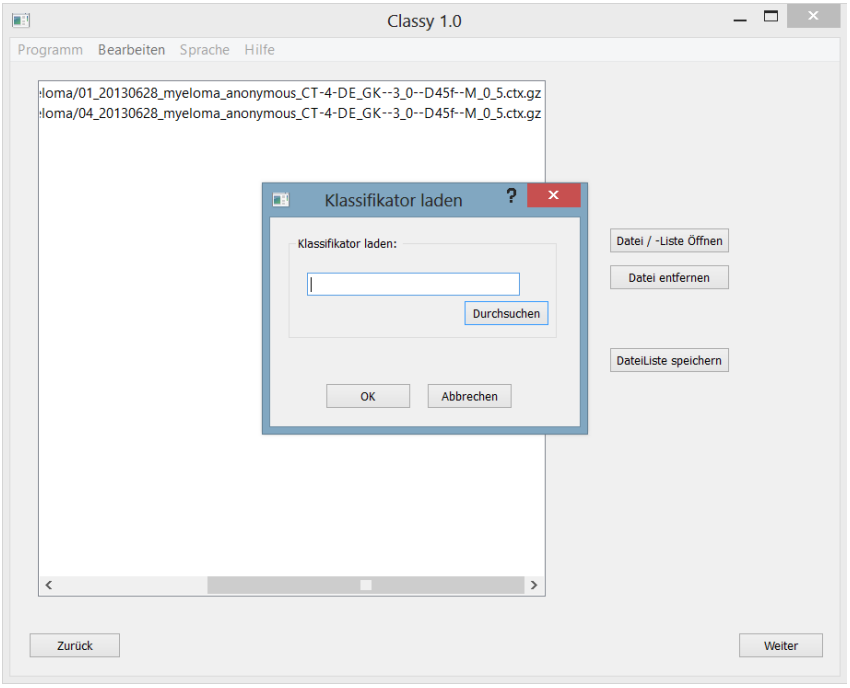


Abbildung 21: Fenster zum Laden eines Klassifikators

Ist dieser erfolgreich geladen, öffnet sich die Merkmalsoberfläche (Abbildung 22). Der Unterschied zum Merkmalsfenster (Abbildung 13) besteht darin, dass hier die Datensätze und die Merkmale angezeigt werden, auf welche der Klassifikator trainiert wurde. Hierbei wurde beim Klassifikator trainieren eine Datei mit den Dateipfaden und den Merkmalen angelegt, welche beim nächsten Schritt automatisch geladen wird. In diesem Modus können keine zusätzlichen Merkmale hinzugefügt oder entfernt werden.

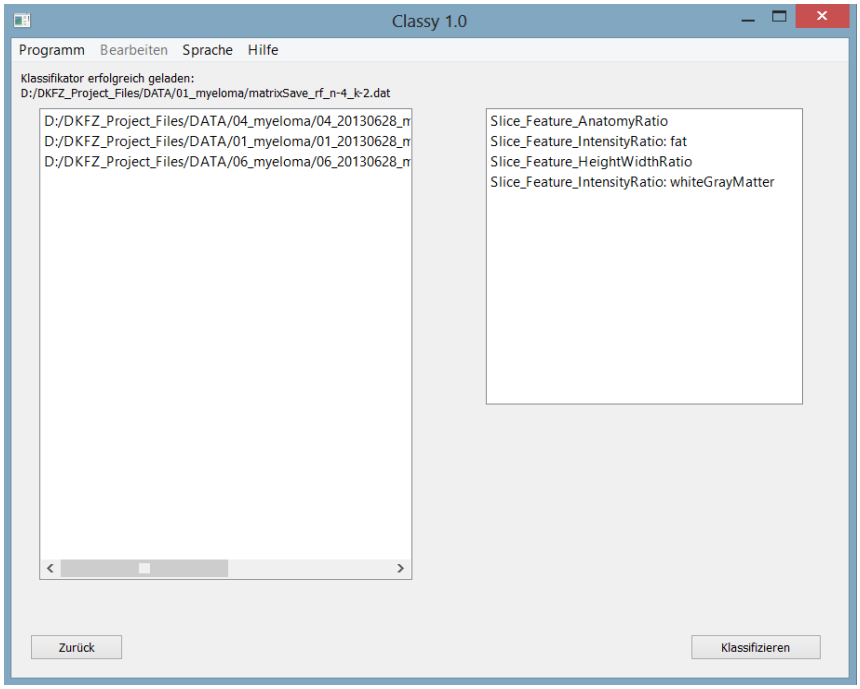


Abbildung 22: Darstellung der Daten und Merkmale auf denen der Klassifikator trainiert wurde

Durch das Betätigen des "Klassifizieren"-Buttons erscheint das Klassifizierungsergebnis in Form einer Tabelle (siehe Abbildung 23). In der ersten Spalte wird der Musterindex und in der zweiten Spalte die vom Klassifikator gewählte Klasse angezeigt.

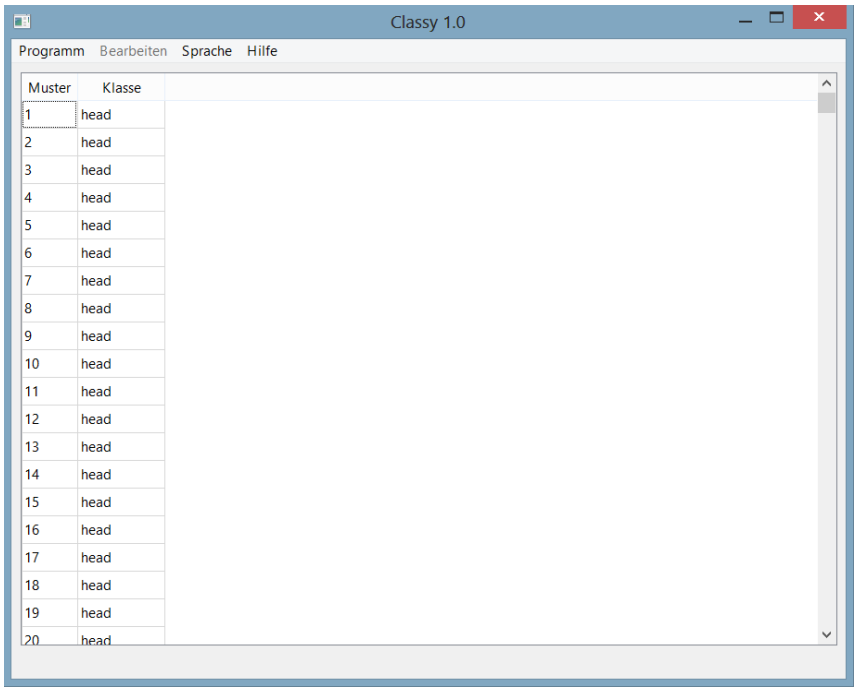


Abbildung 23: Beispiel für ein Klassifikationsergebnis

3.2.4. Leave-One-Out-Test

Nachdem von der Startseite der "Leave-One-Out-Test"-Button ausgewählt wurde, gelangt man, wie im "Klassifikator trainieren"-Modus auch, zuerst zur Wahl der Klassifikation. Wählt der Benutzer die Schichtklassifikation, muss er mindestens zwei Datensätze laden, bevor er fortfahren kann. Anschließend können Merkmale hinzugefügt und entfernt werden. Ist mindestens ein Merkmal ausgewählt und der "Leave-One-Out-Test starten"-Button gedrückt, wird der Anwender dazu aufgefordert einen Titel für das Szenario und einen Speicherort für die Matrizen festzulegen (Abbildung 24).

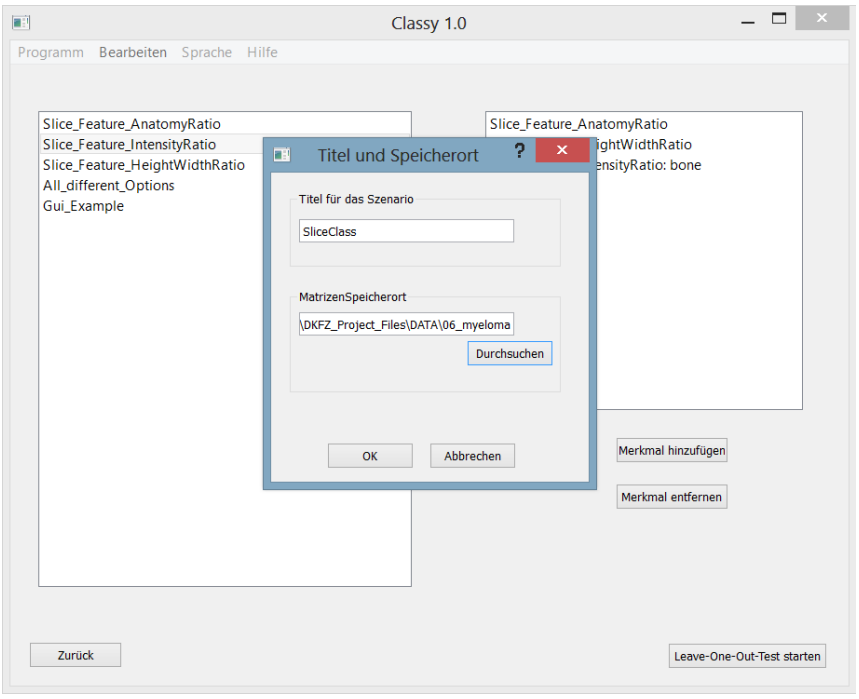
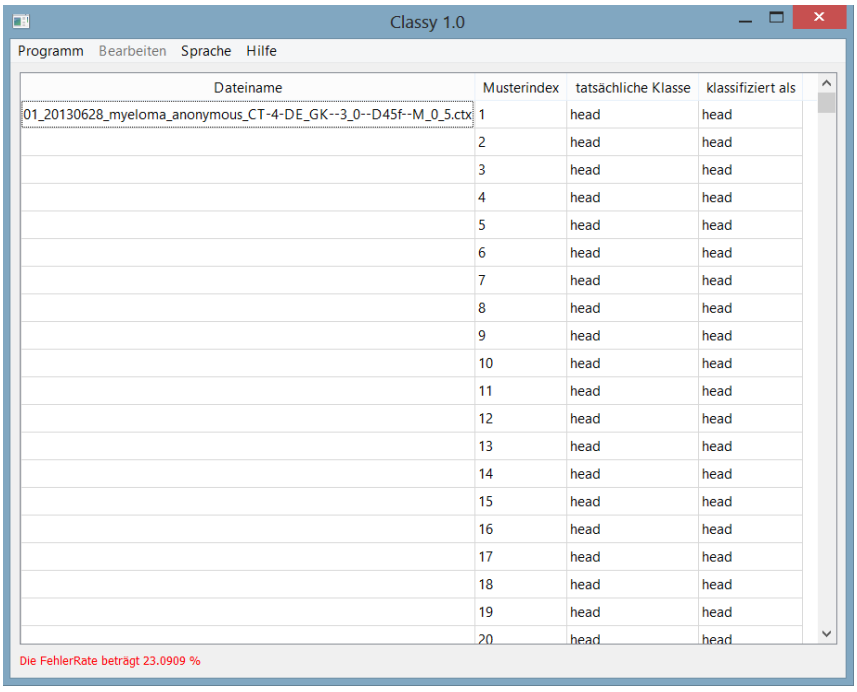


Abbildung 24: Einrichtung des Titel für das Szenario und den Speicherort

Nach Bestätigung der Eingaben muss die Klassifikationsmethode gewählt werden. Anschließend wird das Ergebnis des Bewertungsverfahrens in Tabellenform dargestellt (Abbildung 25).



Dateiname	Musterindex	tatsächliche Klasse	klassifiziert als
01_20130628_myeloma_anonymous_CT-4-DE_GK--3_0--D45f--M_0_5.ctx	1	head	head
	2	head	head
	3	head	head
	4	head	head
	5	head	head
	6	head	head
	7	head	head
	8	head	head
	9	head	head
	10	head	head
	11	head	head
	12	head	head
	13	head	head
	14	head	head
	15	head	head
	16	head	head
	17	head	head
	18	head	head
	19	head	head
	20	head	head

Abbildung 25: Ergebnisdarstellung Leave-One-Out-Test

Wie in Abbildung 25 zu sehen ist, wird in der ersten Spalte jeweils der Dateiname des geladenen Datensatzes, in der zweiten der Musterindex, in der darauffolgenden die tatsächliche Klasse und in der letzten die klassifizierte Klasse angezeigt. In der linken

unteren Ecke gibt es zusätzlich einen Hinweis für die Fehlerrate. Anhand dieser Fehlerrate kann ermittelt werden, wie viele der Muster falsch klassifiziert wurden. Näheres zur Fehlerrate findet sich in Kapitel 2.2.2.2.

3.2.5. Internationalisierung

Zum Wechseln der Sprache findet man im Untermenü des Tabs „Sprache“ die passende Flagge (Abbildung 26). Natürlich lässt sich die Sprache auch jederzeit durch dieselbe Aktion umstellen, auch wenn bereits ein Modus betreten ist. Dies wurde mit dem QT Linguist umgesetzt (2.3.4). Nähere Informationen und die Dokumentation findet man auf der QT Linguist Webseite [10].

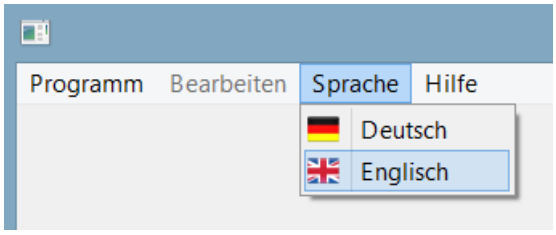


Abbildung 26: Internationalisierung

3.3. Implementierung

In diesem Abschnitt wird die Implementierung und Architektur der Anwendung beschrieben. Zu Beginn werden die Architektur und einige Feinheiten zur Implementierung festgehalten. Darauf folgend wird eine Übersicht über alle Klassen geschaffen. In Kapitel 3.3.3 werden die Hauptausgangsklasse und Klassen ohne GUI erläutert. Anschließend werden die einzelnen Gruppen der Klassen, wie zum Beispiel die "stack"-(3.3.4) oder die "dialog"-Klassen (3.3.5) beschrieben.

3.3.1. Architektur

Die Anwendung in dieser Bachelorarbeit wurde mit Visual Studio 2010 und dem QT4.8-Plugin entwickelt. Dabei wurde auf eine Trennung zwischen den GUIs und dem eigentlichen Inhalt geachtet.

Um zu gewährleisten, dass von jeder grafischen Benutzeroberfläche nur jeweils eine Instanz vorhanden ist, ist ein Singleton Entwurfsmuster implementiert und angewendet worden. Dies bot bei der Verwendung der jeweiligen Klassen einen einfachen Zugriff auf das benötigte, existierende Objekt.

3.3.2. Übersicht der Klassen

Abbildung 27 zeigt eine Übersicht über alle implementierten Klassen. Hierbei gibt es mehrere Einteilungen, in die die unterschiedlichen Klassen strukturiert wurden. In der oberen Hälfte sind die grafischen Oberflächen, die mit dem QT Designer (2.3.3)

entwickelt wurden, aufgelistet. Jedes User Interface(UI)-Element besitzt auch eine passende header- bzw. cpp-Datei, welche im unteren Teil sichtbar sind. In diesem Ordner sind für die bessere Übersichtlichkeit, zwei zusätzliche Unterordner differenziert. Die "stack"- und die "dialog"-Klassen. Klassen die sich im "stack"-Unterordner in Abbildung 27 befinden, sind Oberflächen, die auf dem Hauptfenster angezeigt werden können. Die Klassen die sich im Ordner "dialog" befinden, öffnen ein Dialogfenster. Die beiden Unterordner werden jeweils in einem separaten Kapitel genauer besprochen (3.3.4/3.3.5).

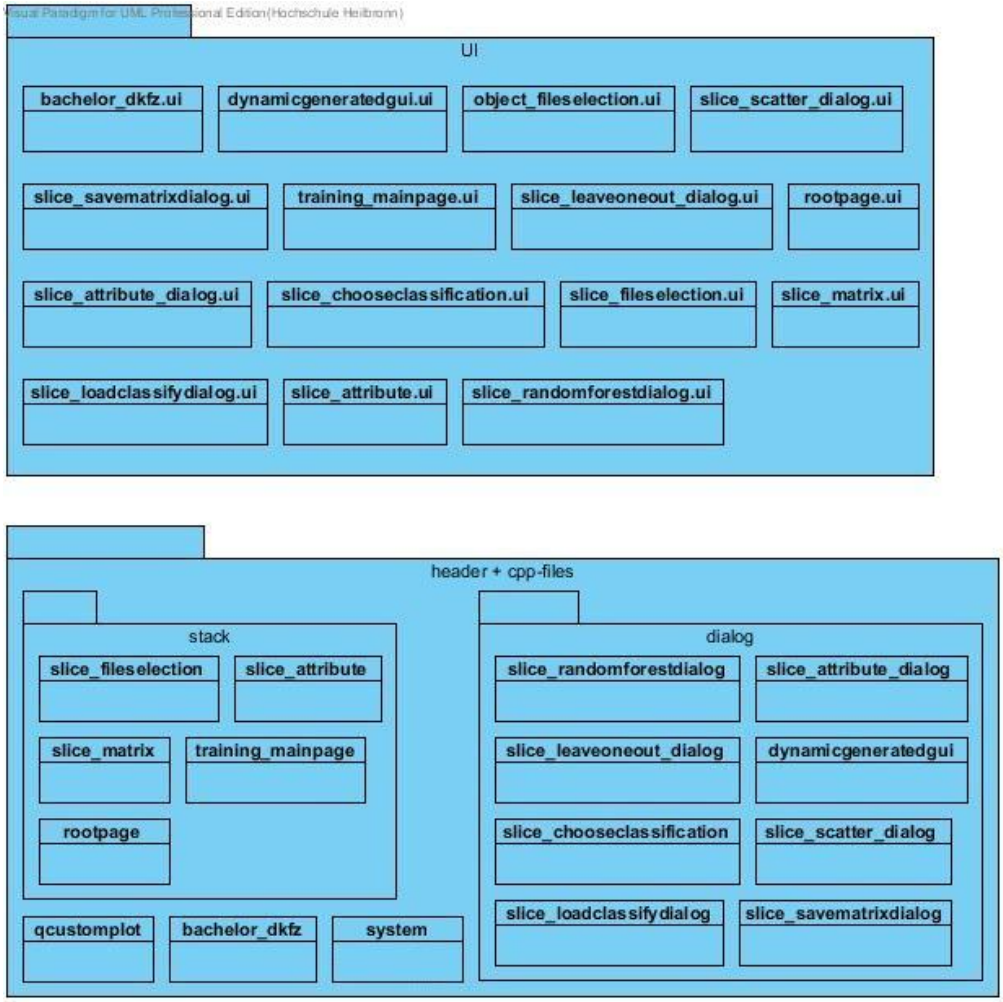


Abbildung 27: UML-Übersicht über alle Klassen

3.3.3. Hauptklasse und Klassen ohne GUI

In diesem Unterkapitel werden die Hauptklasse (Bachelor_DKFZ), die Klasse, die das Anwendungsfenster mit der Menüleiste bildet, und Klassen ohne eine grafische Benutzeroberfläche vorgestellt.

Klasse Bachelor_DKFZ

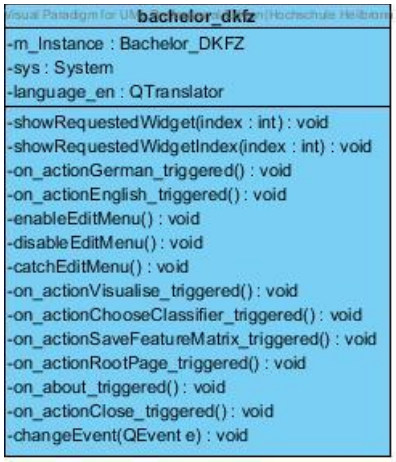


Abbildung 28: UML-Diagramm der Klasse Bachelor_DKFZ

Die Klasse "Bachelor_DKFZ" (Abbildung 28) und das dazugehörige UI stellen das Hauptfenster der Anwendung dar und bilden die Grundebene. Dieses Anwendungsfenster wird beim Initialisieren mit den "stack"-Klassen gefüllt und das erste Element aus dieser Gruppe, die Startseite (3.2.1), wird aufgerufen. Zu dem Fenster gehört außerdem die Menüleiste, die beim Starten der Anwendung ebenfalls erstellt wird. Damit keine mehrfachen Instanzen der "stack"-Klassen vorhanden sind, ist das in 3.3.1 vorgestellte Entwurfsmuster Singleton in dieser Klasse implementiert.

Zusätzlich zu den grafischen Oberflächen und den dazugehörigen header/cpp-Dateien existieren die Dateien QCustomPlot und System. Sie bestehen aus einer header- und einer cpp-Datei.

Klasse QCustomPlot

QCustomPlot ist eine für QT entwickelte Bibliothek zur Visualisierung und zum Plotten von Grafiken und Tabellen. QCustomPlot wurde ausschließlich in der Klasse "Slice_Scatter_Dialog" verwendet, um dort einen Scatterplot zu entwickeln. Eine ausführliche Dokumentation findet man auf der QCustomPlot-Webseite [11].

Klasse System

Die Klasse System besitzt keine grafische Oberfläche. Diese Klasse hat die Aufgabe Werte zu halten und diese dann ein- oder mehrfach weiterzuleiten (Abbildung 29).

Abbildung 29: UML-Diagramm der Klasse System

Diese Werte können zum Beispiel Parameter sein, die in einem Eingabefenster definiert wurden, jedoch in einer anderen Klasse verwendet werden sollen. Hierbei werden die Werte mit Hilfe dieser Klasse und deren Methoden erst gesetzt und dann bei späterer Verwendung aufgerufen.

3.3.4. "stack"-Klassen

Diese Klassen werden auf einem QStackedWidget gelagert. Dies ermöglicht, dass mehrere Oberflächen (Widgets) übereinander liegen, jedoch nur jeweils eine ausgewählte auf dem Hauptfenster (Bachelor_DKFZ) angezeigt wird (Abbildung 30).

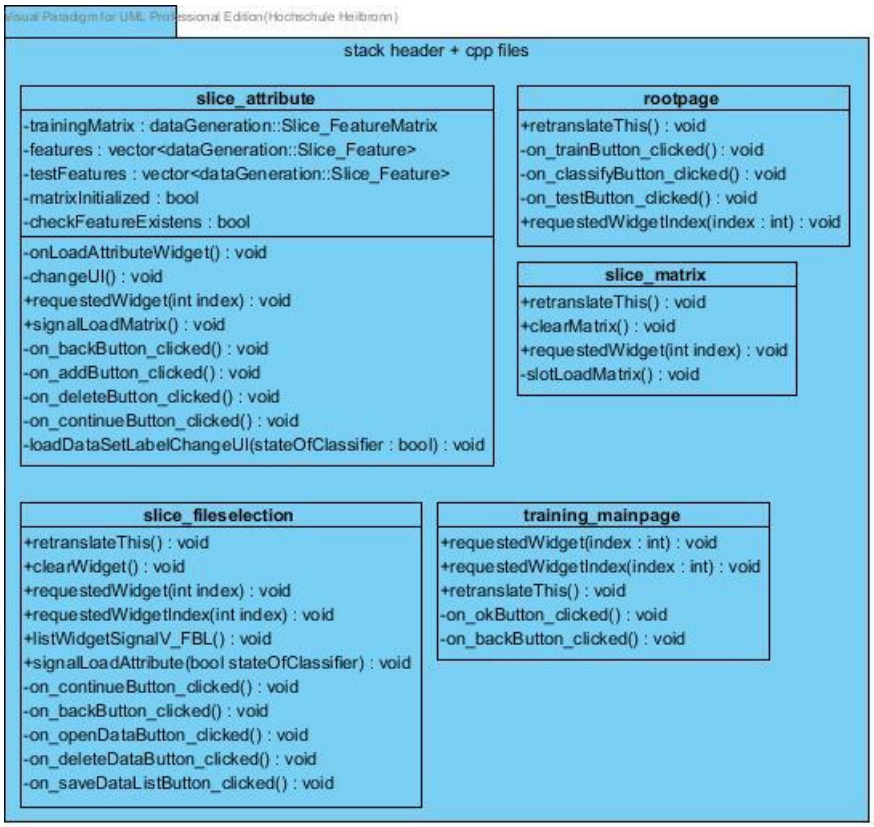


Abbildung 30: UML-Diagramm der "stack"-Klassen

Folgend werden die Klassen kurz vorgestellt, damit die jeweilige Funktion deutlich wird.

Klasse RootPage

Diese Klasse bildet die Startseite der Anwendung. Sie besteht nur aus Buttons, die wiederum auf andere Seiten verweisen.

Klasse Slice_FileSelection [1]

Slice_FileSelection bietet in allen drei Modi die Möglichkeit Datensätze zu laden. Hierbei wird ein Öffnen-Dialog zur Dateiauswahl aufgerufen und die markierten Dateien der Liste hinzugefügt.

[1] Die Namensgebung der Klassen Slice_* wurde gewählt, da sich in dieser Implementierung zuerst der Schichtklassifikation angenommen wurde. Diese Klassen können jedoch auch für andere Klassifikationsprobleme eingesetzt werden.

Klasse Slice_Attribute

Die Klasse Slice_Attribute besteht aus zwei Listen. Eine Liste enthält eine Auswahl an Merkmalen. Die zweite Liste umfasst die Merkmale, die vom Benutzer für die weiteren Arbeitsschritte hinzugefügt wurden.

Klasse Slice_Matrix

Slice_Matrix enthält beim Initialisieren eine leere Tabelle, die im Laufe des Arbeitsablaufs mit Werten gefüllt wird.

Klasse Training_MainPage

Die Klasse Training_MainPage bietet die Auswahl zwischen den drei unterschiedlichen Klassifikationsarten: Schicht-, Voxel- und Objektklassifikation.

3.3.5. "dialog"-Klassen

Die in Abbildung 31 dargestellten Klassen sind Dialog-Fenster, die zu unterschiedlichen Zeitpunkten aufgerufen werden. Jede dieser dialog-Klassen ist realisiert worden, um spezifische Parameter, die für die einzelnen Modi benötigt werden, in einem separaten und übersichtlichen neuen Fenster einzugeben.

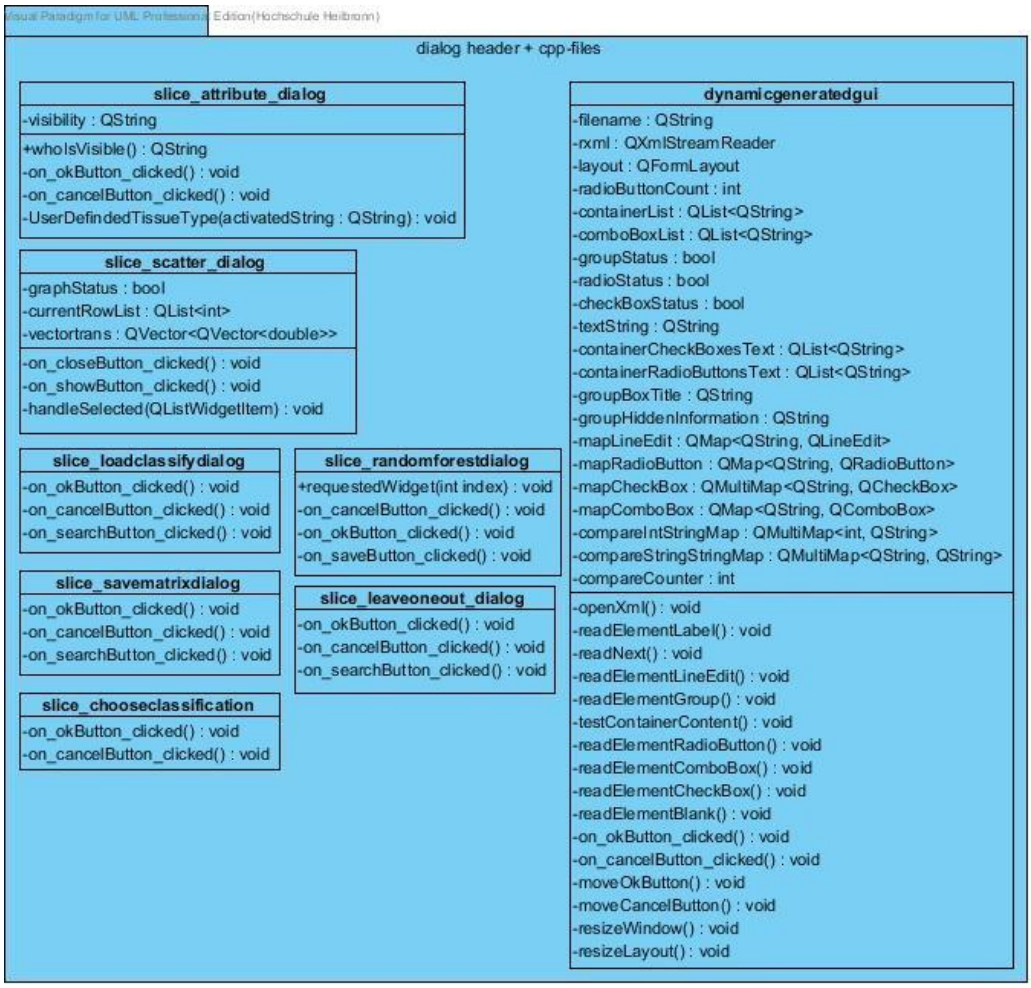


Abbildung 31: UML-Diagramm der "dialog"-Klassen

Zum Zweck des einfacheren Verständnisses, werden die einzelnen Klassen nachstehend knapp dargelegt.

Klasse Slice_ChooseClassification

Slice_ChooseClassification bietet eine Auswahl an Klassifikationsarten an, wie zum Beispiel den Random Forest. Zur Bestimmung der Parameter für die ausgewählte Klassifikationsart wird ein neuer Dialog aufgerufen.

Klasse Slice_LeaveOneOut_Dialog

In diesem Dialogfenster wird ein Titel für das Szenario und den Speicherort der Matrizen gesetzt.

Klasse Slice_LoadClassifyDialog

Mit Hilfe dieses Dialoges wird ein erstellter Klassifikator geladen.

Klasse Slice_SaveMatrixDialog

Diese Klasse erlaubt es dem Anwender, die Werte, welche in der Tabelle der Klasse Slice_Matrix dargestellt sind, in eine vom Benutzer bestimmte Datei abzuspeichern.

Klasse Slice_RandomForestDialog

Slice_RandomForestDialog erreicht man über Slice_ChooseClassification. In diesem Fenster werden Elemente abgefragt, wie zum Beispiel die Anzahl der Bäume (number of trees), die Zahl der Trennungen (number of splits) und der Speicherort.

Klasse Slice_Attribute_Dialog

Mit der Klasse Slice_Attribute_Dialog wird ein für das Merkmal Slice_IntensityRatio spezifiziertes Dialogfenster geladen. In diesem kann der Anwender zwischen vorgegebenen Dropdown-Menüs wählen oder selbst einen Namen und Werte eingeben.

Klasse Slice_Scatter_Dialog

Im Slice_Scatter_Dialog findet die Visualisierung der Werte aus der Tabelle von Slice_Matrix statt. Hierbei werden zwei der gelisteten Merkmale ausgewählt und mit Hilfe der Klasse QCustomPlot visualisiert.

Klasse DynamicGeneratedGUI

Diese Klasse wird für das Auslesen der XML Datei und für das Erstellen der XML-basierten Elemente genutzt. Als Pfad für die Dateien ist der "GUI"-Ordner vorgegeben. Weitere Informationen zu der XML-Erweiterung können im Kapitel 3.4. nachgelesen werden.

3.3.6. Sequenzdiagramme der Modi

Da in Kapitel 3.2 alle Modi mit Hilfe der Oberflächen veranschaulicht wurden, wird in diesem Unterkapitel jeder der Modi mit Hilfe eines Sequenzdiagrammes und einer kurzen Beschreibung näher erläutert.

3.3.6.1. "Klassifikator trainieren"

In Abbildung 32 ist der Arbeitsablauf für das Trainieren eines Klassifikators dargestellt. Der Modus wird von der Startseite aus mit der Funktion "showRequestedWidgetIndex()"aufgerufen. Nach der Auswahl der Klassifikationsart wird mit "showRequestedWidget()" die Oberfläche "Slice_FileSelection" geladen. Nachdem mindestens ein Datensatz hinzugefügt wurde, kann die nächste Oberfläche "Slice_Attribute" mit "loadDataSetLabelChangeUI()" aufgerufen werden. Anschließend können Merkmale hinzugefügt werden und mit "slotLoadMatrix()" wird

"Slice_Matrix" aufgerufen. Durch Auswahl des Menüpunktes "Merkmale visualisieren" wird die Funktion "on_actionVisualise_triggered()" und das Fenster "Slice_Scatter_Dialog" geladen. Von hier aus lassen sich die Merkmale mit "visualiseGraph()" visualisieren oder das Fenster wird mit "close()" geschlossen. Um einen Klassifikator zu erstellen, muss die Ergebnismatrix, welche in "Slice_Matrix" dargestellt ist, abgespeichert werden. Nachdem das Fenster "Slice_SaveMatrixDialog" durch "on_actionSaveFeatureMatrix_triggered()" aufgerufen wurde, kann der Speicherort festgelegt und der Dialog mit "accept()"geschlossen werden. Darauf wird eine Datei erstellt und kann genutzt werden. Nun kann mit "on_actionChooseClassifier_triggered()" eine Klassifikationsmethode ausgewählt werden. Da exemplarisch der Random Forest-Klassifikator implementiert ist, kann durch "showRandomForest()" das Fenster "Slice_RandomForestDialog" geöffnet und die für den Klassifikator benötigten Parameter können festgelegt werden. Im Anschluss wird durch das Bestätigen des Fensters die Funktion "acceptAndRootPage()" aufgerufen, was dazu führt, dass der Klassifikator trainiert und zur Startseite zurückgekehrt wird.

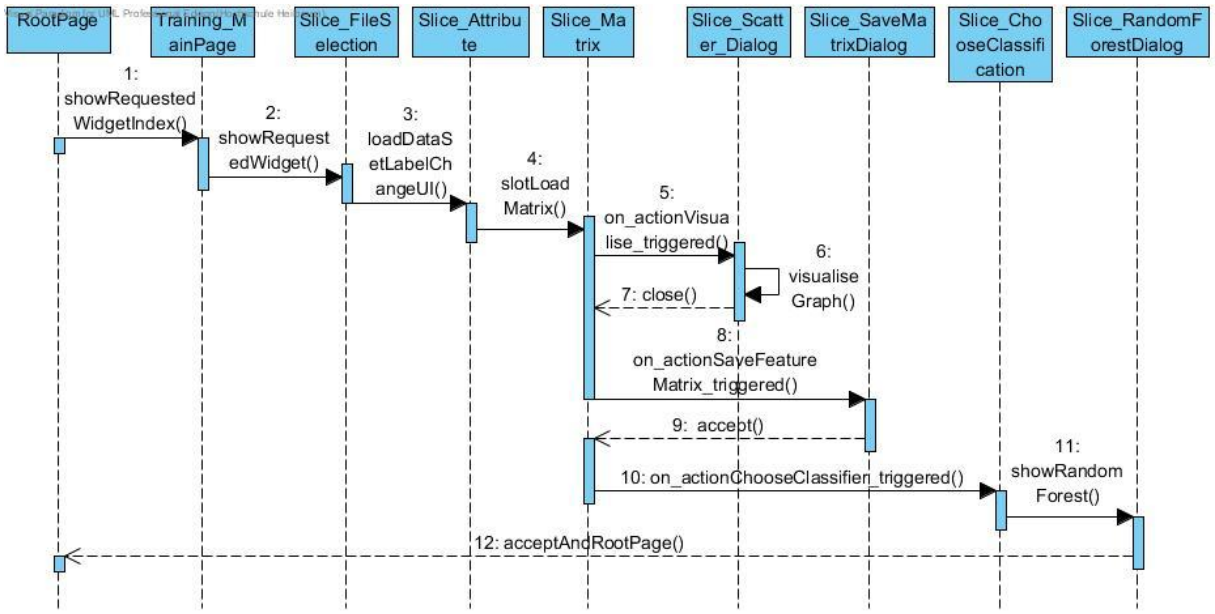


Abbildung 32: Sequenzdiagramm für "Klassifikator trainieren"

3.3.6.2. "Klassifizieren"

Abbildung 33 zeigt ein Sequenzdiagramm für den Modus "Klassifizieren". Der Modus wird von der Startseite aus mit "showRequestedWidgetIndex()" gestartet und führt zur Oberfläche "Slice_FileSelection". Nach dem Laden von mindestens einem Datensatz, erscheint mit "on_continueButton_clicked()" das Fenster "Slice_LoadClassifyDialog". Hier kann ein in Abschnitt 3.2.2 erstellter Klassifikator geladen werden und mit "loadDataSetLabelChangeUI()" wird anschließend die Oberfläche "Slice_Attribute" aufgerufen. In diesem Modus lassen sich keine Merkmale hinzufügen. Stattdessen werden die vom Klassifikator zum Trainieren

verwendeten Datensätze und Merkmale dargestellt. Beim Fortfahren wird mit der Funktion "slotLoadMatrix()" klassifiziert und auf der Oberfläche "Slice_Matrix" das Klassifizierungsergebnis aufgeführt.

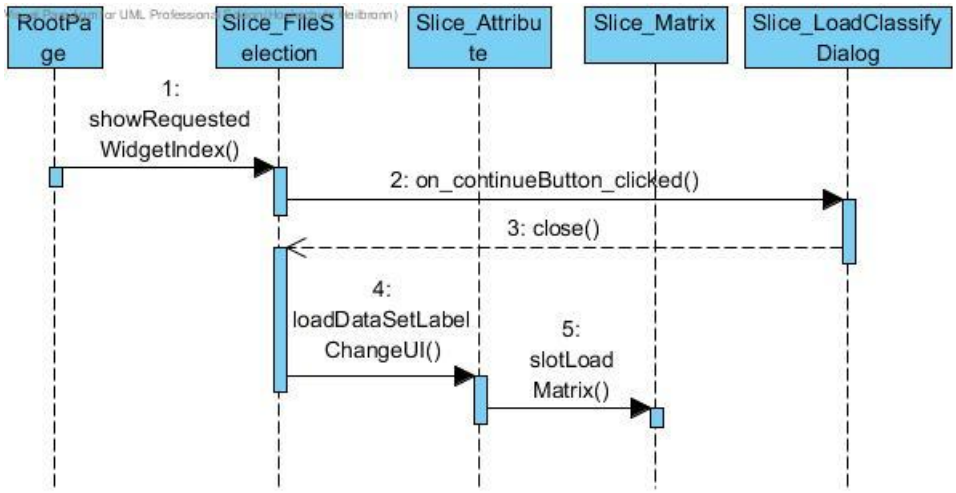


Abbildung 33: Sequenzdiagramm für "Klassifizieren"

3.3.6.3. "Leave-One-Out-Test"

Das Sequenzdiagramm in Abbildung 34 veranschaulicht den Modus "Leave-One-Out-Test". Wie auch die anderen beiden Modi wird der "Leave-One-Out-Test" von der Startseite durch "showRequestedWidgetIndex()" gestartet. Nach Auswahl der Klassifikationsart wird mit "showRequestedWidget()" die Oberfläche "Slice_FileSelection" geladen. Nach dem Laden von mindestens zwei Datensätzen, wird durch den "ok"-Button und die Funktion "loadDataSetLabelChangeUI()" die nächste Oberfläche "Slice_Attribute" geladen. Auf dieser Benutzeroberfläche können Merkmale hinzugefügt werden und wenn mindestens ein Merkmal ausgewählt wurde, kann fortgefahren werden. Durch "on-continueButton_clicked()" wird ein neues Fenster "Slice_LeaveOneOut_Dialog" geöffnet. Der Titel für das Szenario und der Matrizenspeicherort werden in diesem Fenster festgelegt. Nachdem beides ausgefüllt und der Dialog bestätigt wurde, wird durch die Methode "chooseMethodLeOneOut()" ein Fenster zur Wahl der Klassifikationsmethode geöffnet. Da exemplarisch der Random Forest-Klassifikator implementiert ist, wird durch Auswahl die Funktion "showRandomForest()" aufgerufen und ein zusätzliches Fenster wird geladen. In diesem lassen sich die Parameter für den Random Forest-Klassifikator festlegen. Im Anschluss werden die Fenster durch "accept()" geschlossen und das "Leave-One-Out"-Verfahren (2.2.2.2) wird ausgeführt. Nach erfolgreichem Abschluss wird die Funktion "slotLoadMatrix()" aufgerufen und die Ergebnisse des Bewertungsverfahren werden in der Oberfläche "Slice_Matrix" dargestellt.

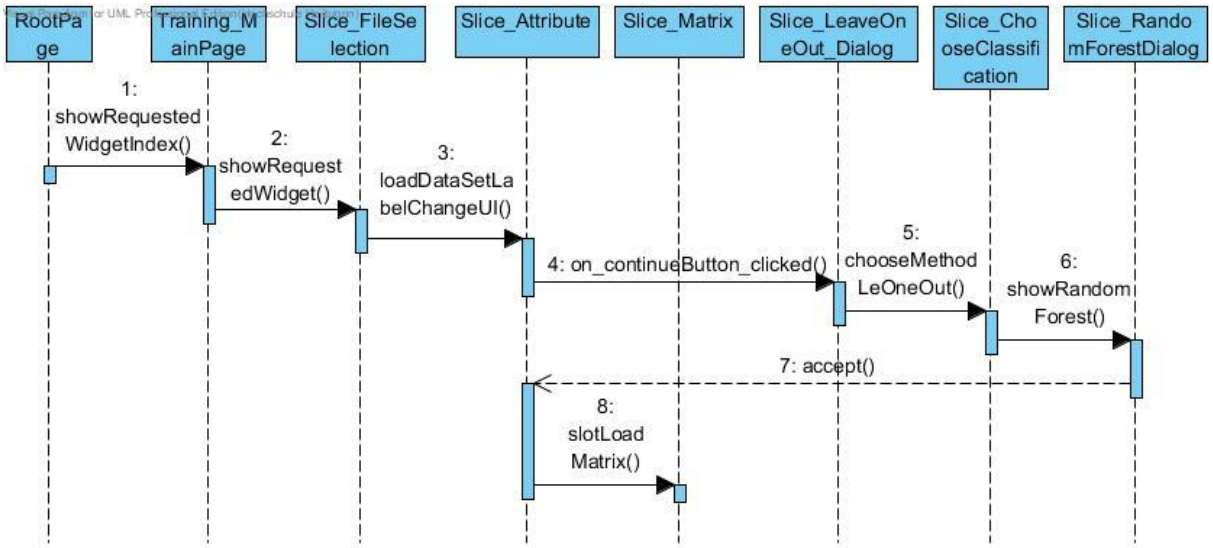


Abbildung 34: Sequenzdiagramm für "Leave-One-Out-Test"

3.4. XML-Erweiterbarkeit

In diesem Kapitel wird eine Erweiterung vorgestellt, die entwickelt wurde, damit Merkmale mit spezifischen Eingaben ohne Zugriff mittels Visual Studio oder QT eingebunden und genutzt werden können. Da man die Merkmale außerhalb einer Entwicklungsumgebung einfügen, ändern und entfernen kann, lässt sich die Anwendung auch ohne Kenntnis der eigentlichen Anwendungslogik und ohne Entwicklungsumgebung erweitern. Diese Erweiterbarkeit ist von grundlegender Bedeutung, damit die entwickelte Software nicht nur für die in dieser Arbeit genutzten Schichtklassifikation genutzt werden kann, sondern auch leicht an andere Klassifikationsaufgaben angepasst werden kann. Um eine strukturierte und einfache Datei einlesen zu können, wurde die XML-Syntax verwendet. Die Wahl der Syntax wurde außerdem durch die Möglichkeit unterstützt, dass mit einem XML-Parser in QT XML-Dateien eingelesen werden können. Genaueres zur XML-Syntax und zur Motivation der Implementierung findet man auf [12] und [13].

3.4.1. Nutzung der XML-Datei

Sollten dem Anwender die passenden Merkmale in der Merkmalsübersicht fehlen, hat er die Möglichkeit Merkmale durch XML hinzuzufügen. Die XML-Dateien besitzen eine festgelegte Syntax und die Dateien müssen in dem "/GUI"-Ordner abgelegt werden, um automatisch eingelesen werden zu können. In der Datei "All_different_Options.xml" sind alle möglichen Optionen festgehalten, eine solche GUI mit Hilfe der XML-Datei zu erstellen. Als Beispiel wurde die XML-Datei "Gui_Example" erstellt. Slice_Feature_IntensityRatio wurde nicht per XML umgesetzt, da diese eine höhere Komplexität aufweist. Nachdem eine solche Merkmalsoberfläche erzeugt wurde, können diese Merkmale angelegt werden. Alle Felder, die entweder ausgefüllt oder bestätigt sind, werden daraufhin als

Unterpunkte eines Merkmales in der Hauptmerkmalsansicht in die rechte Liste hinzugefügt. Diese werden beim Fortfahren ausgelesen und falls eine Übereinstimmung der Merkmale vorhanden ist, genutzt.

3.4.2. Erstellung der XML-Datei

Wie bereits erwähnt, muss die XML-Datei einer festgelegten Syntax folgen. Diese festgelegte Syntax wird im folgenden Abschnitt Schritt für Schritt erklärt.

Die Datei (Abbildung 35/Abbildung 36) muss mit einer UI-Kennzeichnung (I) beginnen und sich am Ende als UI schließen (II). Der Anwender hat mehrere Möglichkeiten die grafische Oberfläche zu füllen. Durch ein "Label" kann er einen einfachen Schriftzug erzeugen, wobei der Inhalt in den Anführungszeichen hinter "text=" stehen muss (1). Um ein Dropdown-Menu zu erzeugen, muss sich an <ComboBox> (2) orientiert werden. Hierbei stehen die Listenunterpunkte hinter "list=". Zur Erstellung eines Eingabefeldes dient <LineEdit> (3). Damit mehrere Optionsfelder erzeugt werden können, müssen diese zuerst mit Hilfe einer "<Group>"-Kennzeichnung eingeschlossen werden. Zu der Kennzeichnung kann ein Titel hinzugefügt werden. Danach kann der Benutzer beliebig viele Felder erzeugen (4). Einen Absatz erzeugt man mit dem Ausdruck <Blank/> (5). Äquivalent zu den Optionsfeldern können Kontrollkästchen erstellt werden (6). Durch das <Group>-Element lässt sich die Oberfläche zusätzlich zeilenweise in zwei Spalten einteilen. Falls dieses Element genutzt wird, wird das erste erstellte Element, z.B. Label links und das zweite Element, z.B. ein Dropdown-Menu rechts angeordnet (7). Wenn keine <Group>-Kennzeichnung vorhanden ist, wird das jeweilige Element auf die komplette Zeile übertragen (10). Standardmäßig besitzt die Oberfläche zwei Knöpfe, den "Ok"- und den "Cancel"-Button. Die Position der Buttons kann durch "xy=" bestimmt werden (9). Um nun auch die Fenster- und die Layoutgröße anpassen zu können, müssen hinter "widthHeight=" zwei Zahlen, durch ein Komma getrennt, eingegeben werden (8). Alle sichtbaren Element außer Blank, Label und den Ok/Abbrechen-Button besitzen das sogenannte "hiddenInformation"-Attribut. Dieses Attribut dient zum Auslesen des Zustandes oder Inhaltes des Elements, um es schließlich in die Hauptmerkmalsansicht zu überführen (Abbildung 37).

```
<?xml version="1.0" encoding="UTF-8"?>  (I)
<UI>
    <Group>
        <Label text="Destination"/> (1)                    (7)
        <ComboBox list="One,Two,Three" hiddenInformation="Destination: "/> (2)
    </Group>
    <Group>
        <Label text="Row to Work"/>
        <LineEdit hiddenInformation="RoWToWork: "/> (3)
    </Group>
    <Blank/>
    <Group title="Choose Classification" hiddenInformation="Chosen Classification: ">
        <RadioButton text="Slice"/>
        <RadioButton text="Object"/>                       (4)
        <RadioButton text="Voxel"/>
    </Group>

    <Blank/> (5)
    <Group title="Where to Save" hiddenInformation="SaveLocation: ">
        <CheckBox text="home"/>                            (6)
        <CheckBox text="extern"/>
    </Group>
    <Label text="ExtraInformation"/>
    <ComboBox list="Yes,No" hiddenInformation="Important: "/>
    <LineEdit hiddenInformation="ExtraInformation: "/>     (10)
    <Layout widthHeight="360,390"/>
    <Window widthHeight="400,460"/>                        (8)
    <OkButton xy="100,415"/>
    <CancelButton xy="200,415"/>                           (9)
</UI> (II)
```

Abbildung 35: XML-Datei mit Elementmarkierungen

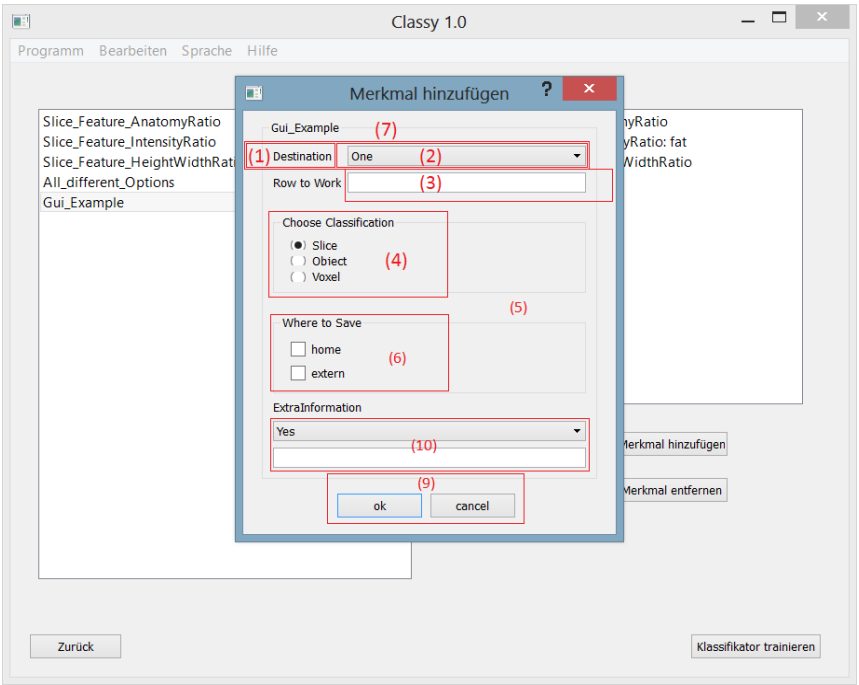


Abbildung 36: XML-erzeugte Oberfläche mit Markierungen

Abbildung 37: Hinzugefügtes XML-generiertes Merkmal

4. Diskussion und Ausblick

Thema dieser Arbeit am DKFZ Heidelberg war das Entwickeln und Implementieren einer Benutzeroberfläche, die auf einem existierenden Programm zur Mustererkennung aus einem Forschungsprojekt basiert. In dem Forschungsprojekt aus der Abteilung Medizinische Physik in der Strahlentherapie wurden Klassifikationsverfahren erstellt, die zum automatischen Erkennen osteolytischer und osteoblastischer Läsionen bei Patienten mit einem multiplen Myelom genutzt werden. Hierzu wurde eine Bibliothek mit unterschiedlichen Algorithmen für die verschiedenen Klassifikationsaufgaben erzeugt. Da eine Nutzung dieser Bibliothek umfangreich und die Anpassung an eine neue Fragestellung sehr aufwändig ist, wurde in dieser Arbeit ein Graphical User Interface geschaffen. Nicht nur werden die Arbeitsprozesse beschleunigt, sondern auch durch intuitiv gestaltete Elemente wie Checkboxen oder Dialogfenster, die zur Eingabe verschiedener Parameter auffordern, wird die Bedienung der Bibliothek und deren Verfahren vereinfacht. Diese Oberflächen ermöglichen Personen mit einfachen Grundkenntnissen, durch intuitive und selbsterklärende Bedienung und durch die Visualisierung von abstrakten Daten, sich mit der Problematik zu befassen. Zielgerichtetes Arbeiten wird somit auch ohne Expertenwissen im Bereich der Software-Entwicklung zugänglich. Ein weiterer Vorteil ergibt sich aus der Erweiterbarkeit durch die XML-Implementierung. Die Erweiterbarkeit ermöglicht das Anpassen und Ergänzen der Benutzeroberfläche bezüglich der Klassifikationskriterien außerhalb einer Entwicklungsumgebung, was zu einem noch profitableren Programm führt. Die Arbeit umfasst zudem ein Verfahren, den Leave-One-Out-Test, welcher zur Überprüfung der Brauchbarkeit von Klassifikatoren und der Klassifizierungen genutzt wird. Darüber hinaus wurde die Internationalisierung in Form von zwei Sprachen (Deutsch und Englisch) umgesetzt, was den nicht deutschsprachigen Nutzern den Zugang erlaubt. Durch alle diese Faktoren kann die Qualität des Forschungsprojektes noch gesteigert werden und resultierend daraus, kann die Abteilung Medizinische Physik der Strahlentherapie am DKFZ Heidelberg davon profitieren.

Die derzeitige Implementierung umfasst eine Klassifikationsart (Schichtklassifikation) und ein Klassifikationsverfahren (Random Forest). Eine denkbare Erweiterung sind die Klassifikationsarten Voxel- und Objektklassifikation. Diese würden äquivalent zu der Schichtklassifikation funktionieren und die meisten Oberflächen könnten erweitert werden, aber erhalten bleiben. Die andere große Erweiterungsmöglichkeit wäre, mehr Klassifikationsverfahren einzubinden, wobei man für diesen Erweiterungsteil den QT-Designer verwenden könnte.

Als Visualisierung ist ein 2D-ScatterPlot implementiert, welcher nur genau zwei Merkmale zur gleichen Zeit gegeneinander abbilden kann. Diesen könnte man mit einem Toolkit, wie VTK (Visualization Toolkit) mit der Möglichkeit von dreidimensionalen Visualisierungen auf drei Merkmale ausbauen.

Im Rahmen dieser Bachelorarbeit wurde der Workflow mit einer grafischen Benutzeroberfläche für das Trainieren und Testen eines Random Forest-Klassifikators für die Klassifikation von Bildschichten als exemplarisches Beispiel vollständig umgesetzt. Weitere Klassifikationsszenarien sind entsprechend zu ergänzen.

5. Literaturverzeichnis

[1]. **Fränzle, Andrea.** Diplomarbeit - Automatische Bildanalyse von MR-Aufnahmen zu Segmentierung von Risikoorganen für die Bestrahlungsplanung mit Random Forests. S. 30-36.

[2]. **Stoll, Armin.** Diplomarbeit - Wissensbasierte Segmentierung von Risikoorganen für die Strahlentherapieplanung. 2006.

[3]. **Jähne, Bernd.** *Digitale Bildverarbeitung.* s.l. : Springer, 2005.

[4]. **Fränzle, Andrea.** Studienarbeit - Entwurf und partielle Implementierung eines Programms zur Vermittlung von Konzepten der Mustererkennung im Rahmen eines Praktikums. 2008.

[5]. **Anil K. Jain, Robert P.W. Duin and Jianchang Mao.** *Statistical Pattern Recognition: A Review. IEEE Transactions on Pattern Analysis and Machine Intelligence.* 2000.

[6]. **E.J. Ciaccio, S.M. Dunn, M. Akay.** *Biosignal Pattern Recognition and Interpretation Systems, 1. Fundamental Concepts.* S. 89-92.

[7]. **Breiman, Leo.** Random Forests. s.l. : Springer, 2001. S. 5-32.

[8]. **Corporation, Nokia.** QT Signal & Slot. [Online] http://harmattan-dev.nokia.com/docs/library/html/qt4/signalsandslots.html.

[9]. **Digia, QT.** QT Designer Manual. [Online] http://qt-project.org/doc/qt-4.8/designer-manual.html.

[10]. —. QT Linguist Manual. [Online] http://qt-project.org/doc/qt-4.8/linguist-translators.html.

[11]. **Eichhammer, Emanuel.** QCustomPlot 1.0.1 Documentation. [Online] http://www.qcustomplot.com/documentation/index.html.

[12]. **Digia, QT.** QXmlStream Bookmarks Example. [Online] http://qt-project.org/doc/qt-4.8/xml-streambookmarks.html.

[13]. **Ferreira, Flavio.** How to Read and Write in XML Files with QT/C++. [Online] https://sites.google.com/a/embeddedlab.org/community/technical-articles/qt/qt-posts/howtoreadandwriteinxmlfileswithqtc.

6. DVD

Die DVD enthält alle Dateien der Anwendung, die Bachelorarbeit als PDF, die Referenzen abzüglich der Bücher und einige Datensätze zum Testen.

7. Danksagung

Ich danke Herrn Prof. Dr. Rolf Bendl für das Anbieten des Themas, das mir entgegengebrachte Vertrauen und die Unterstützung und Durchsicht meiner Arbeit.

Ich möchte Frau Andrea Fränzle danken für ihre hervorragende Betreuung, ihre Anregungen und Ratschläge und für ihre tatkräftige Unterstützung.

Dank geht auch an alle Mitarbeiter und Mitarbeiterinnen der Arbeitsgruppe Therapieplanung der Abteilung Medizinische Physik in der Strahlentherapie am DKFZ Heidelberg, die mich während der Zeit am DKFZ sehr herzlichst aufgenommen haben.