

HOCHSCHULE FÜR TECHNIK RAPPERSWIL

BACHELORARBEIT  
ABTEILUNG INFORMATIK

---

# Selbstlernende Software zur Analyse von Texten

---

Frühlingssemester 2013

*Autoren:*  
Fabian Senn  
Cyrill Lam

*Betreuer:*  
Rolf Schärer

*Projektpartner:*  
Luware AG, ZH

*Experte:*  
Michael Jakob

*Gegenleser:*  
Prof. Dr. Markus Stolze



# Inhaltsverzeichnis

<b>1 Abstract</b>	<b>1</b>
<b>2 Management Summary</b>	<b>2</b>
<b>3 Problemstellung</b>	<b>5</b>
<b>4 Anforderungsspezifikation</b>	<b>7</b>
4.1 Allgemeine Beschreibung . . . . .	7
4.1.1 Produkt Perspektive . . . . .	7
4.2 Funktionale Anforderungen / Use Cases . . . . .	7
4.2.1 Funktionale Anforderungen . . . . .	7
4.2.2 Aktoren und Stakeholder . . . . .	8
4.2.3 Use Case Diagramm . . . . .	9
4.3 Nichtfunktionale Anforderungen . . . . .	17
4.3.1 Mengen . . . . .	17
4.3.2 Qualitätsmerkmale . . . . .	18
4.3.3 Randbedingungen . . . . .	20
4.4 Weitere Anforderungen . . . . .	20
4.4.1 Schnittstellen . . . . .	20
4.4.2 Trainingsdaten . . . . .	21
<b>5 Analyse</b>	<b>22</b>
5.1 Analyse von Texten . . . . .	22
5.1.1 Wortextrahierung / Tokenization . . . . .	22
5.1.2 Stopp-Wörter entfernen . . . . .	22
5.1.3 Word Stemming . . . . .	23
5.1.4 Synonym Erkennung . . . . .	23
5.1.5 Feature Selection . . . . .	23
5.2 Klassifizierung . . . . .	24
5.2.1 Lebenszyklus eines Klassifizierers . . . . .	24
5.2.2 Typen von Klassifikationsalgorithmen . . . . .	25
5.2.3 Umgang mit wenig Trainingsdaten . . . . .	31
5.3 Spracherkennung . . . . .	32
5.3.1 Common Words and Unique Letter Combinations . . . . .	32
5.3.2 Markov Modelle . . . . .	33
5.3.3 N-Gramm . . . . .	33
5.3.4 Compression based approach mittels PPM . . . . .	33
<b>6 Erkenntnisse</b>	<b>34</b>
6.1 Algorithmenwahl . . . . .	34
6.1.1 Sprach Identifizierung . . . . .	34
6.1.2 Word Stemming . . . . .	34
6.1.3 Text Klassifizierung . . . . .	34

6.2	Erste Architekturentwürfe . . . . .	37
6.2.1	Domain Model . . . . .	37
6.2.2	Mail-Kategorie Beziehung . . . . .	39
<b>7</b>	<b>Lösung</b>	<b>40</b>
7.1	Trainingsdaten . . . . .	40
7.2	Frameworks und Libraries . . . . .	40
7.2.1	NTextCat für Sprachidentifikation . . . . .	41
7.2.2	Lucene.Net . . . . .	41
7.2.3	Accord.NET Framework . . . . .	41
7.2.4	DotNetZip . . . . .	41
7.2.5	log4net . . . . .	42
7.2.6	OpenPop.NET . . . . .	42
7.2.7	Moq . . . . .	42
7.2.8	Graph# . . . . .	42
7.3	Trainingsvorgang . . . . .	43
7.3.1	Ablauf . . . . .	45
7.3.2	Neuimplementierung des Optimierungsvorgang . . . . .	46
7.3.3	Domain Modell . . . . .	47
7.4	Kategorisierungsvorgang . . . . .	56
7.4.1	Ablauf . . . . .	57
7.4.2	Categorization Handler Chaining . . . . .	58
7.4.3	Verwendung des Klassifizierers . . . . .	61
7.4.4	Anbindung an ein Mailsystem . . . . .	62
7.5	Architektur . . . . .	63
7.5.1	Layer Architektur . . . . .	63
7.5.2	Projektstruktur . . . . .	66
7.5.3	Externe Abhängigkeiten . . . . .	68
7.5.4	Datenmodell . . . . .	72
7.5.5	Interfaces: Core.Interfaces . . . . .	87
7.5.6	DTOs: Core.Interfaces/Dto . . . . .	93
7.5.7	Interfaces: MachineLearning.Interfaces . . . . .	95
7.5.8	Delegates: MachineLearning.Interfaces . . . . .	100
7.5.9	DTOs: MachineLearning.Interfaces/Dto . . . . .	101
7.6	Konfiguration . . . . .	105
7.6.1	Trainingsparameter - configuration.xml . . . . .	105
7.6.2	Language Identifier . . . . .	117
7.7	Logging . . . . .	118
7.8	Visualisierung des Resultats . . . . .	119
7.9	Testdokumentation . . . . .	120
7.9.1	Testumgebung . . . . .	120
7.9.2	Test Ende C1 . . . . .	122
7.9.3	Test Ende C2 . . . . .	123
7.9.4	Test Parameteroptimierung . . . . .	125
7.9.5	Test Neuimplementierung der Parameteroptimierung . . . . .	126
7.9.6	Test Neuimplementierung der Parameteroptimierung 2 . . . . .	128
7.9.7	Systemtests . . . . .	129
7.9.8	Language Identifier Performancetest . . . . .	137
7.9.9	Performance Test . . . . .	138
7.9.10	Unit Tests . . . . .	139

<b>8 Weiterführende Arbeiten</b>	<b>140</b>
8.1 Regelbasierter Klassifizierer . . . . .	140
8.2 Anbindung an ein Mailsystem . . . . .	140
8.3 Genauere Informationen zum trainierten Klassifizierer . . . . .	140
8.4 Neutrainieren eines Klassifizierers . . . . .	141
8.5 User Interface . . . . .	141
8.6 Konfigurationsfenster für das configuration.xml . . . . .	141
8.7 Zusätzliche Trainingsparameter . . . . .	141
8.8 Generalisierung des adaptiven Klassifizierers . . . . .	141
8.9 Logging und Unit Testing . . . . .	142
<b>9 Projektmanagement</b>	<b>143</b>
9.1 Zweck und Ziel des Projektes . . . . .	143
9.2 Lieferumfang . . . . .	143
9.3 Projektorganisation . . . . .	143
9.3.1 Organisationsstruktur . . . . .	144
9.3.2 Externe Schnittstellen . . . . .	144
9.4 Planung . . . . .	144
9.4.1 Zeitbudget . . . . .	144
9.4.2 Zeitliche Planung . . . . .	144
9.4.3 Besprechungen . . . . .	149
9.5 Risikomanagement . . . . .	150
9.6 Infrastruktur . . . . .	151
9.6.1 Arbeitswerkzeuge . . . . .	151
9.6.2 Zentraler Server . . . . .	151
9.6.3 Entwicklungsumgebung . . . . .	151
9.6.4 Betriebssystem . . . . .	151
9.6.5 Dokumentationswerkzeug . . . . .	151
9.7 Qualitätsmassnahmen . . . . .	152
9.7.1 Dokumentation . . . . .	152
9.7.2 Projektmanagement . . . . .	152
9.7.3 Entwicklung . . . . .	152
9.7.4 Testen . . . . .	153
9.8 Projektauswertung . . . . .	154
9.8.1 Zeit . . . . .	154
9.8.2 Code Statistiken . . . . .	154
<b>10 User Manual</b>	<b>156</b>
10.1 Systemanforderungen . . . . .	156
10.2 Installation . . . . .	156
10.3 User Interface . . . . .	156
10.3.1 Languageidentifier Configuration . . . . .	157
10.3.2 Category . . . . .	158
10.3.3 Training . . . . .	159
10.3.4 Categorization Handler . . . . .	160
10.3.5 Categorization Handler Chaining . . . . .	161
10.3.6 Mailbox . . . . .	162
10.3.7 Mail Emulation . . . . .	163
<b>11 Eigenständigkeitserklärung</b>	<b>165</b>
<b>12 Vom Betreuer unterschriebene Aufgabenstellung</b>	<b>166</b>

<b>13 Glossar</b>	<b>167</b>
<b>Abbildungsverzeichnis</b>	<b>168</b>
<b>Tabellenverzeichnis</b>	<b>170</b>
<b>Literaturverzeichnis</b>	<b>171</b>

# 1 Abstract

In der heutigen Zeit ist gerade die Informationsflut ein grosses Thema. Es ist eine Struktur gefordert, um Informationen effizient verarbeiten zu können. Mit der Zunahme an Daten, wird es jedoch immer schwieriger den Überblick zu wahren. Automatisierung ist also das Stichwort.

Illustriert wird das Problem durch das Beispiel einer Schweizer Firma, die eine öffentliche E-Mail Adresse für Kundenservice zur Verfügung stellt. Um die vielen Kundenanfragen bewältigen zu können, fordert sie eine automatisierte E-Mail Kategorisierung, damit die Nachrichten an die dafür zuständigen Stellen weitergeleitet werden können. Auf diese Weise wird erhofft, die Antwortzeiten zu verkürzen und gleichzeitig Kosten zu reduzieren. Da der Standort in der Schweiz ist, kann der geschäftliche E-Mail Verkehr in Deutsch, Englisch, Italienisch wie auch in Französisch stattfinden. Deshalb soll der Klassifizierer mit diesen Sprachen umgehen können.

In dieser Bachelor Arbeit wurde eine Software entwickelt, die die obenstehenden Anforderungen in Angriff nimmt. Für den Klassifizierungsprozess wird eine Multiclass Support Vector Machine verwendet. Sie gehört zu den intelligenten Systemen, die Modelle eigenständig lernen können. Einzige Voraussetzung für den Lernprozess ist das Vorhandensein von vorkategorisierten E-Mail Daten.

Im Lernprozess müssen die optimalen Parameter der Maschine mittels einem Validationsverfahren eruiert werden. Anschliessend kann mit diesen Einstellungen ein finales Training gestartet werden, welches am Schluss eine voll funktionsfähige Maschine zur Verfügung stellt. Das Ausführen sowie die Spezifikation des Trainingsvorgangs soll mittels einem GUI konfigurierbar gemacht werden.

Ist eine Maschine trainiert worden, kann sie Eingabedaten selbstständig den ihr bekannten Kategorien zuordnen. Die Maschine verhält sich hierbei passiv, da sie nur probabilistische Aussagen treffen kann.

Als Erweiterung können auch mehrere Klassifizierer hintereinander geschaltet werden. Dies ist zum Beispiel nützlich, wenn eine feingranulare Aufteilung nötig ist. So könnte beispielsweise in erster Instanz ein Spamfilter trainiert werden, der nur die beiden Kategorien "Spam" und "nicht-Spam" kennt. Erst wenn klar ist, dass es sich bei der Nachricht nicht um Spam handelt, beginnt die eigentliche Zuteilung. Der Entscheidungsweg kann grafisch dargestellt werden, damit gut nachvollzogen werden kann wie man auf das Resultat gekommen ist.

## 2 Management Summary

### Ausgangslage

Wir leben heute im digitalen Zeitalter. Die Menge an Informationen, die täglich bewältigt werden müssen, wächst von Tag zu Tag. Deshalb wird es immer wichtiger, die Bearbeitung mittels automatisierten Prozessen zu unterstützen. Eine solche Anwendung wäre die selbstständige Kategorisierung von E-Mail Daten.

Was für einen Mehrwert ist zu gewinnen, wenn E-Mails automatisch einer Kategorie zugeordnet werden können? Besitzt eine Firma zum Beispiel eine öffentliche E-Mail Adresse, so werden Nachrichten für verschiedene Zuständigkeitsbereichen an einem zentralen Ort eingehen. Damit das E-Mail die richtige Stelle findet, muss es weitergeleitet werden. Dies kann aber ab einem gewissen Umfang nicht mehr durch eine Person bewerkstelligt werden. Deshalb ist eine intelligente Analyse und automatisierte Weiterleitung gefordert. Auf diese Weise werden die Antwortzeiten verkürzt und es können Kosten eingespart werden.

Da in der Schweiz der geschäftliche E-Mail Verkehr in Deutsch, Englisch, Italienisch oder Französisch stattfindet, muss zusätzlich die Sprache identifiziert werden.

Weiterhin handelt es sich bei E-Mails um vertrauliche Daten. Somit dürfen keine Online Services angesteuert werden. Stattdessen sollen alle Berechnungen im lokalen Netz stattfinden.

### Vorgehen

Bei diesem Projekt wurde nach RUP gearbeitet. Zuerst wurden vier Phasen definiert und anstehende Arbeiten auf diejenigen verteilt.

Die erste Phase diente primär dazu, einen Überblick über die verschiedenen Klassifikationsalgorithmen zu verschaffen. Dazu gehörte die Analyse derer Funktionsweisen und die Überprüfung ihrer Tauglichkeit.

In einer zweiten Phase wurden die verschiedenen Algorithmen miteinander verglichen und deren Vor- und Nachteile gegeneinander abgewogen. Zum Schluss wurde die Auswahl auf eine Möglichkeit beschränkt und so festgelegt, welcher Algorithmus für den weiteren Verlauf des Projektes verwendet werden soll. Kleinere Prototypen folgten, um erste Erkenntnisse im Umgang zu sammeln.

In der dritten Phase fand der Grossteil der Entwicklung statt. Gegen Ende der dritten Phase und in Teilen der vierten Phase, wurde die Zeit in das Refactoring und Strukturierung des Codes investiert.

Die vierte und letzten Phase diente zur Validierung der Software. Dazu wurden viele Systemtests durchgeführt und womögliche Fehler behoben. Des Weiteren wurde der Dokumentation der letzte Schliff verpasst.

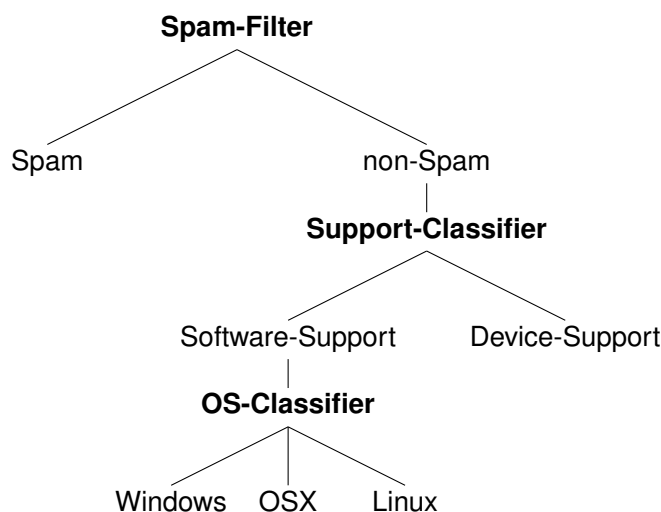
## Ergebnisse / Technologien

Das Resultat dieser Arbeit ist eine Software, welche Texte in Form von E-Mails klassifizieren und Kategorien zuordnen kann. Dazu wurde ein selbstlernendes System mittels Support Vektor Maschine entworfen und umgesetzt. Für den Einsatz einer solchen Maschine muss in erster Instanz ein Lernprozess durchgeführt werden. So können die Parameter des Klassifikationsalgorithmus festgelegt werden. Einzige Voraussetzung für das selbständige Training ist das Vorhandensein von vorkategorisierten E-Mails; den sogenannten Trainingsdaten. Diese können über eine einfache grafische Oberfläche den verschiedenen Kategorien hinzugefügt werden, worauf sie beim nächsten Training miteinbezogen werden.

Als Zusatzfeature bietet die Software auch die Möglichkeit, einen ganzen Baum von Klassifizierer zu erstellen. So dass immer genauere Aufteilungen möglich sind. Dazu kann für einen bestehenden Klassifizierer angegeben werden, welcher Nachfolger aufgerufen werden soll, wenn die Nachricht einer bestimmten Kategorie zugeteilt wurde.

So ist es zum Beispiel möglich ein vorgängiges Spam-Filtering zu betreiben, um anschliessend mit der eigentlichen Klassifizierung fortzufahren. Dazu wird eine Support Vektor Maschine mit zwei Kategorien "Spam" "nicht-Spam" erstellt. In einem weiteren Schritt werden die nicht-Spam-Mails in die gewünschten Kategorien weitere Kategorien unterteilt. Dabei wird wiederum eine neue Maschine trainiert.

Als Illustration soll ein Klassifizierer-Baum einer fiktiven Firma erstellt werden. Sie unterhält zwei Support Services "Device-" und "Software-Support". Weiterhin wird beim Software-Support zwischen den Betriebssystemen "OSX", "Linux" und "Windows" unterschieden. Mit diesen Anforderungen sieht der daraus resultierende Baum folgendermassen aus:



Die fett geschriebenen Knotenpunkte stellen jeweils die Klassifizierer dar. Die übrigen Punkte sind die Kategorien.

Die komplette Software wurde mit Microsoft Technologien realisiert. Als Programmiersprache kam C# zur Anwendung. Für das Konfigurations-UI wurde WPF verwendet und um Resultate persistent zu speichern ein MS SQL Server eingesetzt.



## Ausblick

Ein automatische Klassifizierung hat immer den Nachteil, dass an dessen Entscheidungsfindung nichts beigesteuert werden kann. Wenn ein Klassifizierer somit bestimmte E-Mails ständig falsch klassifiziert, bleibt als einzige Möglichkeit ein neuer Trainingsvorgang übrig. Da dazu jedoch zuerst manuell Trainingsdaten angelegt werden müssen, die den Fehler des Klassifizierers ausbügeln sollten, wäre es vorübergehend einfacher, ein regelbasiertes System miteinzubeziehen. Ein solcher regelbasierter Klassifizierer zu entwickeln, wäre eine weiterführende Arbeit.

Weiterhin setzt man die Software ein, um Daten zu klassifizieren, die im Laufe der Zeit Veränderungen durchlaufen können. So ist es beispielsweise möglich, dass das Geschäftsfeld einer Firma wechselt. Das bedeutet auch, dass sich die Inhalte der zukünftigen E-Mails verändern werden. Ein klassischer Klassifizieralgorithmus, wie er in dieser Arbeit umgesetzt wurde, kann auf einen solchen Wandel nicht reagieren. Einzig der Administrator der Software kann neue Trainingsdaten beilegen, die die neuen Gegebenheiten abdecken. Es ist also ein dynamisches System gefordert, das auf Änderungen in den Daten reagieren kann. Dazu wäre zum Beispiel die Hinzunahme von externen Suchmaschinen oder grösseren internen Dictionaries eine Möglichkeit, dem Klassifizierer Wissen mitzuteilen, das über die Trainingsdaten hinaus geht.

Im Moment ist die Entwickelte Software eine eigenständige Applikation ohne Kommunikation mit anderen Systemen. Es wäre aber möglich, sie als eine Extension zu bestehenden Mailsystemen, zum Beispiel MS Exchange, zu verwenden. Eingehende E-Mails können so automatisch klassifiziert und in vorkonfigurierte Ordner verschoben werden.

## 3 Problemstellung

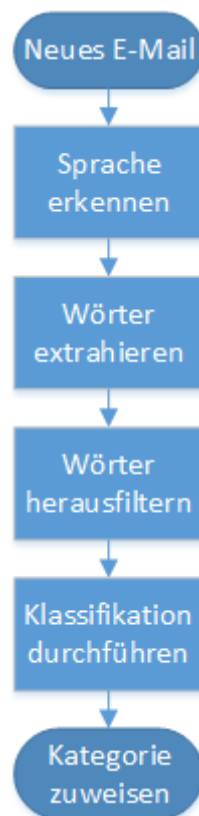
Es soll ein Textklassifizierer entwickelt werden, der E-Mails automatisiert in verschiedene Kategorien einteilen kann. Er soll die Fähigkeit besitzen, die Sprache von Texten zu erkennen und muss in Bezug auf die ihm bekannten Kategorien konfigurierbar sein.

Falls der Klassifizierer bei einem Kunden zum Einsatz kommt, so muss derjenige vorkategorisierte E-Mails zur Verfügung stellen. Diese Daten werden benötigt, um den Klassifizierer zu trainieren.

Sprachen, welche unterstützt werden, sind Deutsch und Englisch. In einem zweiten Schritt Französisch und Italienisch.

Die Architektur soll so aufgebaut sein, dass es nicht viel Aufwand erfordert einen Algorithmus für eine bestimmte Aufgabe durch einen anderen zu ersetzen.

Aus der Problemstellung leitet sich folgender Ablauf ab:



**Abbildung 3.1:** Übersicht gesamter Klassifikationsvorgang

1. **Sprache erkennen:** Es wird die Sprache des Textes identifiziert und gekennzeichnet
2. **Wörter extrahieren:** Der Text wird in Subelemente (Wörter) aufgeteilt. Die Trennung erfolgt bei Leer- und Satzzeichen

3. **Wörter herausfiltern:** Unwichtige, wie auch hochfrequente Wörter werden herausgefiltert. Diejenigen Worte, die herausgefiltert wurden, tragen nichts zur verbesserten Klassifizierung bei und können ignoriert werden
4. **Klassifikation durchführen:** Anhand der Wörter, die aus dem E-Mail extrahiert wurden, wird eine Klassifikation mittels eines Maschinen Lernalgorithmus durchgeführt
5. **Kategorie zuweisen:** Anhand der Ausgabe des Maschinen Lernalgorithmus wird die Kategorie festgelegt und dem E-Mail zugewiesen

In den nachfolgenden Kapiteln wird genauer auf die einzelnen Schritte eingegangen. Algorithmen und Methoden für das Lösen der einzelnen Probleme werden genauer erläutert.

# 4 Anforderungsspezifikation

## 4.1 Allgemeine Beschreibung

Der Text Klassifizierer soll Firmen helfen E-Mails, effizienter zu bearbeiten. Elektronische Nachrichten, die auf eine allgemeine öffentliche E-Mailadresse eingehen, sollen einer bestimmten Kategorie wie zum Beispiel "Buchhaltung" oder "Informatik" zugeordnet werden können. Dazu müssen aber zu Beginn bereits vorklassifizierte E-Mails (Trainingsdaten) zur Verfügung gestellt werden. Im Betrieb soll es möglich sein, dem Klassifizierer weitere Trainingsdaten zuzuspielen, die er für weitere Trainingssessionen verwenden kann.

### 4.1.1 Produkt Perspektive

Im Rahmen der Bachelor Arbeit werden die Grundfunktionalitäten des E-Mail Text Klassifizierers umgesetzt. Wenn zufriedenstellende Ergebnisse erzielt wurden, soll der Klassifizierer in das Produktportfolio der Luware aufgenommen werden. Da bei der Entwicklung des E-Mail Text Klassifizierer auf einen modularen Aufbau geachtet wird, ist es auch denkbar, dass in Zukunft auch eingehende Facebook Nachrichten klassifiziert werden können.

## 4.2 Funktionale Anforderungen / Use Cases

### 4.2.1 Funktionale Anforderungen

- Das System kann aufgrund von vordefinierten Trainingsdaten automatisiert E-Mails kategorisieren.
- Das System trainiert selbstständig durch manuellen Anstoss verschiedene Klassifiziererinstanzen.
- Ist die Trainingsphase abgeschlossen, werden die Instanzen validiert.
- Das System meldet alle seine Trainingsergebnisse mit deren Konfigurationen und gibt an, mit welcher Konfiguration das beste Ergebnis erzielt wurde.
- Ein Benutzer kann jederzeit die bestehende Instanz mit einer neuen Konfiguration updaten.

## 4.2.2 Aktoren und Stakeholder

### 4.2.2.1 Administrator

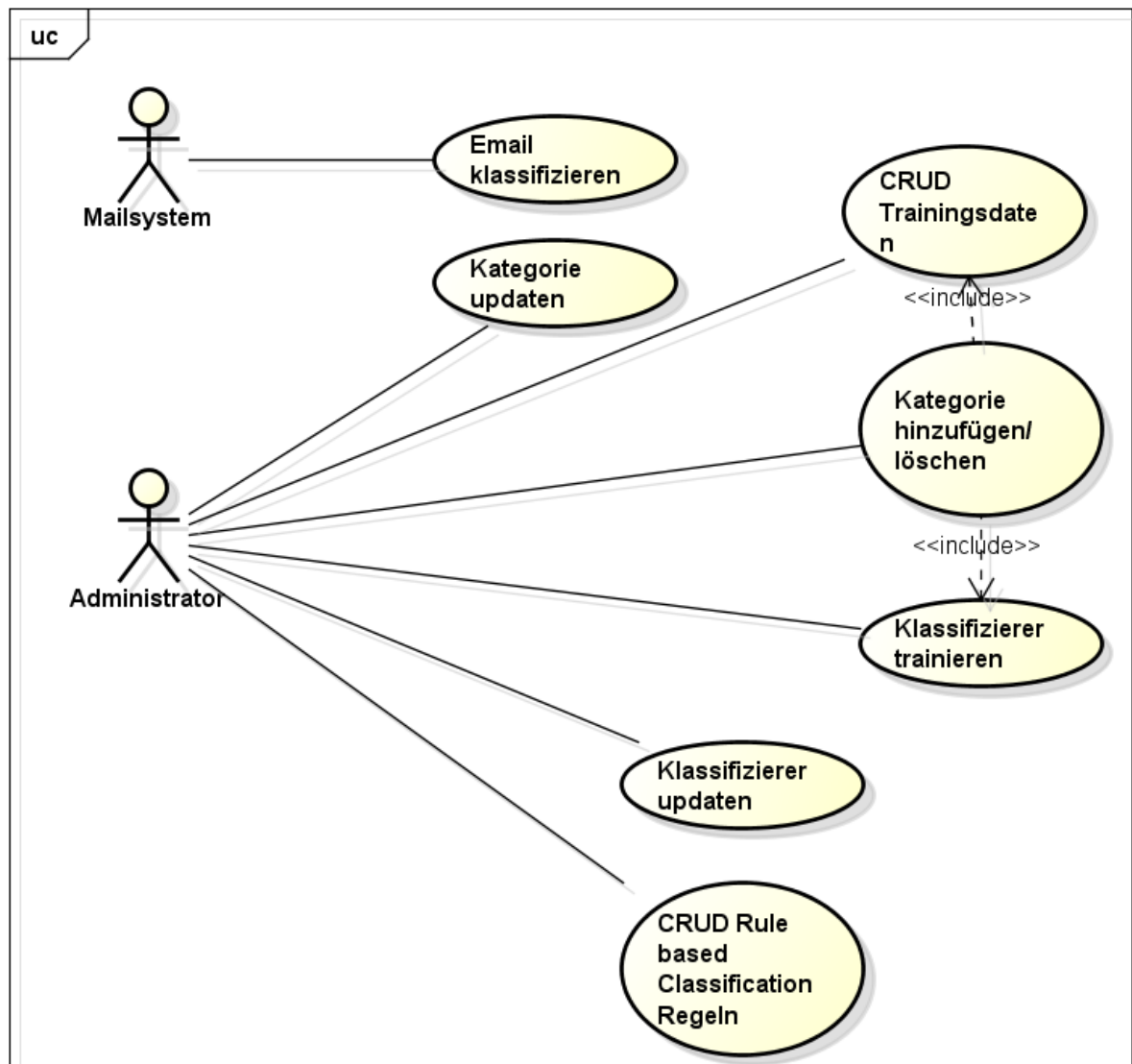
Der Administrator ist verantwortlich für die Konfiguration des E-Mail Text Klassifizierers. Er verwaltet die Kategorie und pflegt die dazugehörigen Trainingsdaten. Weiterhin konfiguriert und aktualisiert er den Klassifizierer mit neuen Parametern, wenn eine Verbesserung erzielt werden kann.

Administratoren können in einer Firma mehrere verschiedene Personen sein. Für das System spielt es aber keine Rolle, wer genau mit dem System interagiert.

### 4.2.2.2 Mailsystem

Das Mailsystem übergibt dem E-Mail Klassifizierer un kategorisierte E-Mails zur Klassifizierung.

### 4.2.3 Use Case Diagramm



powered by Astah

Abbildung 4.1: Use Case Diagramm

#### 4.2.3.1 Use Cases Brief

##### 4.2.3.1.1 E-Mail klassifizieren

Mails die vom Klassifizierer empfangen werden über zwei Klassifizierungsstufen einer bestimmten Kategorie zugeordnet. Die Stufen haben folgende Verantwortlichkeiten.

1. Adaptiver Klassifizierer: Hier werden die Mails anhand eines aufwändigen Maschinen-Lernalgorithmus klassifiziert. Als Grundvoraussetzung müssen ihm Trainingsdaten zur Verfügung gestellt werden, damit er mit diesen trainiert kann. Seine Stärken liegen darin, dass er die Klassifizierung automatisch bewerkstelligen kann und die Konfiguration nicht manuell durchgeführt werden muss.

2. Regelbasierter Klassifizierer: Ist die Klassifikation durch den vorhergehenden Klassifikator abgeschlossen, wird über einen regelbasierten Klassifizierer die Zuweisung verfeinert. Anschliessend wird die Nachricht an den Verteiler weitergegeben und die Klassifizierung ist abgeschlossen. Die Idee hinter dem regelbasierten Klassifizierer ist es, dass wiederauftretende Falschklassifizierungen mit einer Regel behoben werden können, bis der adaptive Klassifikator mit besseren Trainingsdaten neu trainiert wurde.

#### 4.2.3.1.2 Kategorie updaten

Namen der Kategorien kann geändert werden.

#### 4.2.3.1.3 Kategorien löschen/hinzufügen

Es können verschiedene Kategorien hinzugefügt oder gelöscht werden. Diesen Kategorien nach werden die E-Mails zukünftig eingeteilt. Diese Änderung tritt jedoch erst in Kraft, wenn der adaptive Klassifizierer neu gelernt hat und die neuen Parameter übernommen wurden.

#### 4.2.3.1.4 CRUD Trainingsdaten

Pro Kategorie kann auf eine Sammlung von Trainingsdaten zu dieser Kategorie hingewiesen werden. Diese Trainingsdaten werden dann für das Training und Auswertung des Klassifizierers verwendet.

#### 4.2.3.1.5 Klassifizierer updaten

Ist die Validation des Trainings abgeschlossen, werden die Resultate an den Administrator mitgeteilt. Anhand der Ergebnisse kann nun der Administrator entscheiden, ob der bestehende Klassifizierer ersetzt werden soll. Der Administrator übergibt nun die erhaltenen Parameter an den Klassifizierer, woraufhin ein Update stattfindet und der neue Klassifikator im Einsatz ist.

#### 4.2.3.1.6 Klassifizierer trainieren

Der Klassifizierer nimmt anhand der zur Verfügung stehenden Trainingsdaten ein neues Training vor. Nach dem Training wird der Administrator über die Verbesserung oder Verschlechterung gegenüber dem bestehenden Klassifizierers informiert.

#### 4.2.3.1.7 CRUD Rule-Based Classification Regeln

Post-Klassifikation tritt dann in Kraft, wenn die Kategorie des Mails bestimmt wurde. Hier können weitere Regeln angefügt werden, die das Klassifizieren verfeinern. Ein paar Beispiele sind folgend aufgelistet.

- Die Nachricht hat einen bestimmten Absender
- Ein Schlüsselwort wird im Betreff oder im Body erwähnt
- Klassifizieren nach der Sprache, in der es geschrieben wurde
- Mail hat ein Attachment
- Ist das Mail nicht eindeutig einer Klasse zuzuordnen, kann definiert werden, an welchen Verteiler Mails dieser Art weitergeleitet werden
- Mails die zur gleichen Klasse gehören, die aber in unterschiedlichen Sprachen sind können an jeweils andere Empfänger übermittelt werden

### 4.2.3.2 Use Cases (Fully Dressed)

#### 4.2.3.2.1 E-Mail klassifizieren

Primary Actor	Mailsystem
Stakeholders	Die Firma möchte eine automatische Klassifizierung umsetzen, um die Abarbeitungsgeschwindigkeit von eingehenden Mails zu beschleunigen.
Preconditions	<ul style="list-style-type: none"> <li>• Der Klassifizierer ist eingerichtet und wurde trainiert</li> <li>• Es ist ein Mail empfangen worden, das die Klassifizierung auslöst</li> </ul>
Postconditions	<ul style="list-style-type: none"> <li>• Das Mail ist klassifiziert und ist an den Verteiler mit der Information zur Kategorie übermittelt worden</li> </ul>
Main Success Scenario	<ol style="list-style-type: none"> <li>1. Es wird ein E-Mail empfangen, welches am Klassifizierer übergeben wird.</li> <li>2. Die E-Mail Elemente werden entfernt, damit nur die verschiedenen Felder mit den Inhalten zur Verfügung stehen</li> <li>3. Der zur Verfügung stehende Text wird von einem Sprachidentifikator analysiert. Ist eine zutreffende Sprache gefunden worden, wird das Mail damit gekennzeichnet.</li> <li>4. Das Mail wird an den adaptiven Klassifizierer übergeben, der das Mail einer Kategorie zuordnet</li> <li>5. Bevor das Mail an den Endverteiler weitergereicht wird, wird es vom Post-Klassifizierer auf eigens definierte Regeln überprüft</li> </ol>
Extensions	*a Jede Entscheidung, die getroffen wird, wird in einem Log gespeichert

#### 4.2.3.2.2 CRUD Trainingsdaten

Primary Actor	Administrator
Stakeholders	Im Verlaufe der Zeit verändert sich der Fokus einer Firma. So werden beispielsweise neue Produkte angefertigt oder die Kundengruppe ändert sich. Um diesem Problem gewachsen zu sein, muss der Klassifizierer diese Veränderungen lernen und dazu sind neue Trainingsdaten von Nöten.
Preconditions	<ul style="list-style-type: none"> <li>• Die Kategorien sowie deren Zielordner mit den Trainingsdaten wurden angegeben</li> </ul>



Postconditions	<ul style="list-style-type: none"><li>• Trainingsdaten sind den Kategorien hinzugefügt worden</li></ul>
----------------	---

Main Success Scenario	<ol style="list-style-type: none"> <li>1. Der Administrator öffnet Trainingsdaten Menu</li> <li>2. Im Trainingsdaten Menu befindet sich eine Übersicht über die Kategorien, die existieren       <ol style="list-style-type: none"> <li>a) Es sollen neue Trainingsdaten hinzugefügt werden           <ol style="list-style-type: none"> <li>i. Im Menu wählt der Administrator die Kategorie an, in der das Mail hineinplatziert werden soll</li> <li>ii. Über den Menüpunkt „hinzufügen“ öffnet sich ein Fileexplorer</li> <li>iii. Im File-Explorer wählt der Administrator die gewünschten Mails an und klickt „Ok“, um sie zu laden</li> <li>iv. Nach dem Ladevorgang sind die Trainingsdaten der Kategorie zugeordnet</li> </ol> </li> <li>b) Es soll ein Trainingsmail gelöscht werden           <ol style="list-style-type: none"> <li>i. Im Menu wählt der Administrator die Kategorie an, aus der das Mail gelöscht werden soll</li> <li>ii. In einer Liste der Trainingsmails dieser Kategorie wählt der Administrator jene Mails aus, welche er löschen möchte</li> <li>iii. Durch Druck auf den Button „löschen“ wird der Löschvorgang eingeleitet</li> <li>iv. Nach dem Vorgang sind die Trainingsdaten nicht mehr der Kategorie zugeordnet</li> </ol> </li> <li>c) Es soll ein Trainingsmail in eine andere Kategorie übertragen werden           <ol style="list-style-type: none"> <li>i. Im Menu wählt der Administrator den Menüpunkt, um Trainingsdaten in eine neue Kategorie zu transferieren</li> <li>ii. Der Administrator gibt an, zwischen welchen Kategorien Trainingsdaten ausgetauscht werden sollen</li> <li>iii. Es erscheint ein Menu mit zwei Listen, wobei jede mit den Trainingsdaten der jeweiligen Kategorien gefüllt sind</li> <li>iv. In einer der jeweiligen Liste wählt der Administrator die Trainingsdaten aus, die in die andere Kategorie übertragen werden sollen</li> <li>v. Durch einen Druck auf „transferieren“ wird der Transfer eingeleitet</li> <li>vi. Nach dem Vorgang sind die Trainingsdaten der neuen jeweilig anderen Kategorie zugeteilt</li> </ol> </li> </ol> </li> <li>3. Sollen weitere Trainingsdaten bearbeitet werden, wird beim Punkt 2 weitergefahren</li> <li>4. Der Nutzer speichert die Einstellungen</li> </ol>
Extensions	-

#### 4.2.3.2.3 Kategorie hinzufügen/löschen

Primary Actor	Administrator
Stakeholders	Eine Firma möchte Mails, die über eine gemeinsame Adresse empfangen werden, auseinander halten. Aus diesem Grund definiert sie Kategorien, zu denen die Mails zugeordnet werden sollen.
Preconditions	-
Postconditions	<ul style="list-style-type: none"> <li>Die Kategorien sind angelegt worden und können für ein neues Training des Klassifikators verwendet werden</li> </ul>
Main Success Scenario	<ol style="list-style-type: none"> <li>Der Administrator öffnet die Übersicht über die Kategorien</li> <li>In der Übersicht sind die bisher definierten Kategorien aufgelistet <ol style="list-style-type: none"> <li>Es soll eine neue Kategorie hinzugefügt werden <ol style="list-style-type: none"> <li>Der Administrator wählt den Menüpunkt "hinzufügen"</li> <li>Im nun ersichtlichen Menu gib der Administrator der Name der neuen Kategorie an</li> <li>Weiterhin wird die Zieladresse für die Weiterleitung festgelegt</li> <li>Nach einem Klick auf den "speichern" Button wird die Einstellung gespeichert</li> <li>Der Vorgang ist abgeschlossen. In der Kategorienübersicht wird die neue Kategorie aufgelistet</li> </ol> </li> <li>Es soll eine Kategorie gelöscht werden <ol style="list-style-type: none"> <li>Der Administrator wählt in der Übersicht die Kategorie, die gelöscht werden soll</li> <li>Der Administrator klickt auf den "löschen" Button, woraufhin die Kategorie gelöscht wird</li> <li>Der Vorgang ist abgeschlossen. In der Kategorienübersicht wird die gelöschte Kategorie nicht mehr aufgelistet</li> </ol> </li> </ol> </li> </ol>
Extensions	

#### 4.2.3.2.4 Klassifizierer trainieren

Primary Actor	Administrator
Stakeholders	Die Firma hat Interesse daran, dass sich die automatisierte Klassifizierung über die Zeit verbessert und auf Domänenveränderungen reagieren kann.
Preconditions	-
Postconditions	<ul style="list-style-type: none"> <li>Der Klassifizierer arbeitet mit den neuen Parametern</li> </ul>

Main Success Scenario	<ol style="list-style-type: none"> <li>1. Der Administrator öffnet das Menu des Klassifizierers</li> <li>2. Der Administrator klickt auf den Button "Klassifizierer trainieren"</li> <li>3. Es wird das Trainingsmenu angezeigt</li> <li>4. Der Administrator wählt die Kategorien, die gelernt werden sollen</li> <li>5. Der Administrator drückt auf "Training starten"</li> <li>6. Der Trainingsvorgang wird gestartet</li> <li>7. Wenn der Trainingsvorgang abgeschlossen ist, wird das beste Resultat des Trainingsvorgangs angezeigt und mit der bisherigen Installation verglichen</li> </ol>
Extensions	

#### 4.2.3.2.5 Klassifizierer updaten

Primary Actor	Administrator
Stakeholders	Die Firma hat Interesse daran, dass sich die automatisierte Klassifizierung über die Zeit verbessert und auf Domänenveränderungen reagieren kann.
Preconditions	-
Postconditions	<ul style="list-style-type: none"> <li>• Parameter für die verschiedenen Resultate stehen zur Verfügung</li> <li>• Die Resultate wurden dem Administrator mitgeteilt</li> </ul>
Main Success Scenario	<ol style="list-style-type: none"> <li>1. Der Administrator öffnet das Menu des Klassifizierers</li> <li>2. Im Menu befindet sich eine Übersicht über das zuletzt durchgeführte Training. Dazu ist ersichtlich, wie hoch die Fehlerrate war, welche Kategorien trainiert worden sind und wie viele Trainingsdaten zu den jeweiligen Kategorien zur Verfügung standen.</li> <li>3. Der Administrator wählt das Trainingsresultat aus</li> <li>4. Es wird der Button „berechneter Klassifizierer anwenden“gedrückt</li> <li>5. Die aktuelle Instanz des Klassifizierers wird mit der ausgewählten ausgetauscht</li> <li>6. Die Klassifizierung wird mit der neuen Instanz fortgeführt</li> </ol>

Extensions	<p>2a Es wird kein Trainingsresultat angezeigt. Dies bedeutet, dass seit der Generierung des verwendeten Klassifikators kein Training durchgeführt wurde.</p> <p>3a Der Administrator entscheidet sich, das Trainingsresultat nicht anzuwenden</p>
------------	--

#### 4.2.3.2.6 CRUD Rule-Based Classification Regeln

Primary Actor	Administrator
Stakeholders	Kommt es häufig vor, dass Mails falsch kategorisiert werden, ist es ein Anliegen, dass möglichst schnell ein Fix dafür herausgebracht wird. Da für einen neuen Trainingsvorgang zuerst dementsprechende Trainingsdaten angelegt werden müssen, ist es vorteilhaft, wenn kurzzeitig manuell die Änderung vorgenommen werden kann.
Preconditions	-
Postconditions	<ul style="list-style-type: none"> <li>• Die definierten oder angepassten Regeln werden verwendet. Regeln, die gelöscht wurden, sind nicht mehr aktiv</li> </ul>

Main Success Scenario	<ol style="list-style-type: none"> <li>1. Der Administrator öffnet das Regelmenu des regelbasierten Klassifizierers</li> <li>2. Im Regelmenu befindet sich eine Übersicht über die bisher definierten Regeln       <ol style="list-style-type: none"> <li>a) Es soll eine neue Regel erfasst werden           <ol style="list-style-type: none"> <li>i. Es wird über der Menüpunkt „hinzufügen“angewählt</li> <li>ii. Der Nutzer formiert die Regel über einen Regex oder Keywörter, die über boolsche Regeln verknüpft werden können.</li> <li>iii. Dann gibt er an, auf welche E-Mail Elemente die Regel angewendet werden soll</li> <li>iv. Zum Schluss wird festgelegt, welcher Kategorie das Mail, das dieser Beschreibung betrifft, hinzugefügt werden soll</li> </ol> </li> <li>b) Es soll eine Regel gelöscht werden           <ol style="list-style-type: none"> <li>i. Der Benutzer wählt eine Regel aus</li> <li>ii. Über den Menüpunkt „löschen“wird die Regel gelöscht</li> </ol> </li> <li>c) Es soll eine Regel bearbeitet werden           <ol style="list-style-type: none"> <li>i. Der Benutzer wählt eine Regel aus</li> <li>ii. Über den Menüpunkt „bearbeiten“öffnet sich ein Menu</li> <li>iii. Im Menu kann die bisherige Regel angepasst werden</li> </ol> </li> </ol> </li> <li>3. Sollen weitere Regeln definiert werden, wird beim Punkt 2 weitergefahren</li> <li>4. Der Nutzer speichert die Einstellungen, die nun vom Klassifikator angewendet werden.</li> </ol>
Extensions	

### 4.3 Nichtfunktionale Anforderungen

Die Nichtfunktionalen Anforderungen sind gestützt auf ISO 25000.

Offizielle Seite: [www.iso.org/iso/iso\\_catalogue/catalogue\\_tc/catalogue\\_detail.htm?csnumber=35683](http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=35683)

Wikipedia: [de.wikipedia.org/wiki/ISO/IEC\\_25000](http://de.wikipedia.org/wiki/ISO/IEC_25000)

#### 4.3.1 Mengen

- Es ist immer nur eine Instanz des gleichen Klassifizierers aktiv. Verschiedene Klassifizierer können gleichzeitig agieren.

## 4.3.2 Qualitätsmerkmale

### 4.3.2.1 Funktionalität

#### Interoperabilität

- Das Programm wird mittels C# entwickelt und als ausführbares Programm ausgeliefert. Aus diesem Grund ist es nur auf Servern oder Computern lauffähig, welche Windows und das .NET Framework 4.5 installiert haben.

#### Richtigkeit

- Beim Klassifizieren wird das E-Mail unverändert einer Kategorie zugeordnet. Alle Informationen, die vor der Klassifizierung vorhanden sind, sind auch nach der Klassifizierung noch anwesend.

#### Sicherheit

- Klassifizierte E-Mails dürfen nicht zwischengespeichert werden.
- Trainingsdaten die zur Verfügung gestellt werden, müssen gesichert abgelegt sein.
- Es werden keine Online Schnittstellen angesprochen um keine sensible Daten weiterzugeben.
- Für die Datensicherheit der persistenten Daten wie das Klassifizierermodeill ist die Firma selber verantwortlich.

### 4.3.2.2 Zuverlässigkeit

#### Zuverlässigkeit

- Fehler werden in jedem Fall abgefangen. Der Fehler wird immer versucht intern behandelt zu werden, so dass die Applikation weiter laufen kann. Ist dies nicht möglich, wird eine benutzerfreundliche Fehlermeldung ausgegeben, um über die Fehlerursache zu informieren.

### 4.3.2.3 Benutzbarkeit

#### Verständlichkeit

- Die Software soll möglichst einfach konfigurierbar sein, ohne dass Fachwissen über die verwendeten Algorithmen benötigt werden.

#### Erlernbarkeit

- Das Administrations UI soll möglichst simpel gehalten und intuitiv designed sein, so dass man sehr schnell den Klassifizierer konfigurieren kann.

## Bedienbarkeit

- Die einfachen Interaktionsmöglichkeit mit dem Klassifizierer findet über das Konfiguration UI statt. Komplexere Konfigurationen werden über ein Konfigurationsfile verändert.

### 4.3.2.4 Effizienz

#### Zeitverhalten

- Die E-Mails werden möglichst schnell kategorisiert, so dass keine Verzögerung beim Abarbeiten entsteht.
- Der Dauer des Trainingsvorgangs hängt stark von den eingestellten Parametern und Kategorien ab. Damit die Geschwindigkeit von Mehrkernsystemen genutzt werden kann, wird das Training stark parallel ausgelegt.

#### Verbrauchsverhalten

- Es sollen zu jedem Zeitpunkt nur so viele Ressourcen angefordert werden, wie effektiv benötigt nötig sind.
- E-Mails sollen in keinem Fall zwischengespeichert werden.
- Während dem Training eines Klassifizierers kann es wegen der Parallelität des Vorgangs vorkommen, dass sehr viel Speicher benötigt wird. Dies hängt vor allem von der Anzahl Trainingsdaten und Kategorien ab.
- Über einen Parameter kann eingestellt werden, wie viel Speicher während dem Trainingsvorgang genutzt werden soll.

### 4.3.2.5 Änderbarkeit

#### Analysierbarkeit

- Da eine modulare Architektur angestrebt wird, in der alle Komponenten getrennt realisiert werden, können Fehler auch einfach lokalisiert werden.
- Die Einarbeitung wird dadurch erleichtert.

#### Modifizierbarkeit

- Das Austauschen von einzelnen Algorithmen soll durch klare Abgrenzung innerhalb der Architektur möglichst vereinfacht werden.
- Die verwendeten Programmbibliotheken werden weitestgehend gekapselt, damit Abhängigkeiten minimiert werden.



## Stabilität

- Da der Klassifizierer 7/24 in Betrieb ist, dürfen keine Memory Leaks entstehen.

## Testbarkeit

- Es soll für alle Kernkomponenten während der Entwicklung Unit-Tests geschrieben werden, um Fehlverhalten möglichst früh zu erkennen.
- Weiterhin wird die Software so ausgelegt, dass sie ein hoher Grad an Abstraktheit besitzt. Dadurch wird das Simulieren von Programmkomponenten vereinfacht, was die Testbarkeit verbessert.

### 4.3.2.6 Übertragbarkeit

## Installierbarkeit

- Das Programm wird als Executional File für Windows Plattformen ausgeliefert, weshalb keine Installation nötig ist.
- Jegliche Konfigurationsfiles werden ausgeliefert und so definiert, dass sie für die allgemeine Nutzung nicht angepasst werden müssen.

### 4.3.3 Randbedingungen

- Es wird .NET Framework 4.5 benötigt.
- Es werden für jede zu klassifizierende Kategorie Trainingsdaten benötigt. Die Menge muss für jede Kategorie ausgeglichen sein.

## 4.4 Weitere Anforderungen

### 4.4.1 Schnittstellen

#### 4.4.1.1 Benutzerschnittstellen

- Die Einstellung des Klassifizierers findet über ein GUI statt. Dazu gehört die Festlegung der Kategorien sowie deren Trainingsdaten
- Einstellungen für das Training von Klassifizierern wird mittels einem XML definiert. Dies wird bei der Auslieferung der Software mitgegeben und ist so konfiguriert, dass im Allgemeinen keine Anpassungen nötig sind

#### 4.4.1.2 Netzwerkschnittstellen

- Der Klassifizierer verwendet TCP/IP, um mit dem MS SQL Server zu kommunizieren. Damit soll gute Skalierbarkeit sichergestellt werden.

## 4.4.2 Trainingsdaten

Um den Klassifizierer trainieren zu können, werden Trainingsdaten benötigt. Trainingsdaten sind in dieser Arbeit nichts anderes als E-Mails deren Kategoriezugehörigkeit schon bekannt ist.

Um die Zugehörigkeit festzulegen, steht eine grafische Oberfläche zur Verfügung. Mit ihr ist es möglich über einen File-Explorer E-Mails, die sich auf einem Datenträger befinden, einer Kategorie hinzuzufügen.

Nachfolgend wird beschrieben wie die Trainingsdaten vorliegen müssen.

### 4.4.2.1 Format

Für E-Mails ist MIME das Standardformat. Deshalb erwartet die zu entwickelnde Software, dass die Trainingsdaten im MIME Format zur Verfügung gestellt wird.

### 4.4.2.2 Aufbau

```
From: "Doug Sauder" <doug@example.com>
To: "Jürgen Schmürgen" <schmuergen@example.com>
Subject: Lorem ipsum dolor
Date: Wed, 17 May 2000 19:08:29 -0400
Message-ID: <17575401.1075855758266.doug@example.com>
MIME-Version: 1.0
Content-Type: text/plain; charset="iso-8859-1"
Content-Transfer-Encoding: 8bit
X-Priority: 3 (Normal)
X-MSMail-Priority: Normal
X-Mailer: Microsoft Outlook IM0, Build 9.0.2416 (9.0.2910.0)
Importance: Normal
X-MimeOLE: Produced By Microsoft MimeOLE V5.00.2314.1300
```

Lorem ipsum

Lorem ipsum dolor sit amet, consetetur sadipscing elitr...

Für die Wortextrahierung werden folgende Felder betrachtet:

- **To** - Liste von E-Mailadressen, an die die Nachricht gerichtet ist
- **From** - E-Mailadresse des Absenders der Nachricht
- **Subject** - Kurzbeschreibung, um was es in der Nachricht geht
- **Body** - Enthält die eigentliche Nachricht

# 5 Analyse

## 5.1 Analyse von Texten

Dokumente bieten als Merkmale die einzelnen Wörter. In der Annahme, dass bestimmte Wörter in einer Klasse oder Kategorie öfters verwendet werden als in anderen, kann für ein neues Dokument entschieden werden, zu welcher Klasse es gehört. Diese Vereinfachung wird von allen Klassifizierungsalgorithmen angewendet. Die Schwierigkeit dabei ist aber, diejenigen Worte zu finden, welche am meisten Aussagekraft bei der Bestimmung der Klassen besitzen. Diese Wörter werden als "Features" bezeichnet, weil sie als differenzierbares Merkmal von Dokumenten betrachtet werden.

Feature Selection ist der Vorgang, aus einem Dokument, diejenigen Worte zu gewinnen. Es hat das Ziel, die Anzahl Merkmale von Texten zu minimieren und so die Laufzeit der Lernalgorithmen zu verringern.

### 5.1.1 Wortextrahierung / Tokenization

Die Textdaten stehen normalerweise als einzelner Eingabestrom zur Verfügung. Aus diesem Strom müssen die Wörter extrahiert und in einen Tokenstream gefasst werden. Dieser Vorgang wird Tokenization genannt. Bei den lateinischen Sprachen werden dazu Leerzeichen und Satzzeichen als Separatoren verwendet.

Einfachheitshalber ist es ausserdem nötig, dass die Tokens ausschliesslich in Lower- beziehungsweise Upper-Case zur Verfügung stehen.

### 5.1.2 Stopp-Wörter entfernen

In einem nächsten Schritt werden diejenigen Wörter entfernt, die keinen Beitrag zur Bestimmung der Klassenzugehörigkeit leisten und nur aus grammatikalischen beziehungsweise syntaktischen Gründen verwendet werden. Zu diesen Wörter gehören beispielsweise die Pronomen, Verbindungswörter usw. Sie werden auch als Stopp-Wörter bezeichnet.

Stopp-Wörter können aber auch domänenspezifisch sein. So ist es zum Beispiel nicht erwünscht, dass "Bildschirm" ein Feature-Wort sein soll, wenn es sich bei den Trainingsdaten nur um Texte über OLED- und LCD-Displays handelt.

Stopp-Wörter sind von Sprache zu Sprache unterschiedlich. Deshalb muss in einem vorhergehenden Schritt die Sprache des Textes identifiziert werden.

### 5.1.3 Word Stemming

Wörter können in Texten aufgrund von Konjugation oder Fallunterscheidungen unterschiedlichste Formen annehmen. Ohne Betrachtung dieser Eigenschaft, werden sie trotz der gemeinsamen Grundform als separate statt als gleiche Wörter gewertet. Um diesem Problem entgegenzukommen, ist es also nötig, sie in eine gemeinsame Grundform zu bringen. Dazu gibt es zwei verschiedene Ansätze. Der einfachere der beiden arbeitet mit einer Tabelle von Wörtern und deren Grundformen. Die andere Variante versucht mittels Algorithmen die Umwandlung zu automatisieren. Einer dieser Algorithmen ist der Porter Stemming Algorithmus. Dieser steht für mehrere Sprachen zur Verfügung. Logischerweise ist die zweite Variante vorzuziehen, da das manuelle Erstellen von Tabellen eine zeitintensive Aufgabe ist. Bei der Verwendung des Algorithmus muss aber in Kauf genommen werden, dass Verallgemeinerungen gemacht werden, die manchmal zu fehlerhaften Grundformen führen. Dieser Nachteil kann aber vernachlässigt werden.

Der Porter Stemming Algorithmus verwendet eine Vielzahl von Regeln bis das zu bearbeitende Wort eine Minimalanzahl von Vokal-Konsonant-Sequenzen aufweist. Danach werden Verkürzungsregeln angewendet, um Suffixe zu minimieren oder zu entfernen. Zum Schluss bleibt nur noch der Wortstamm übrig, der nicht immer linguistisch korrekt ist.

Ursprünglich wurde der Algorithmus für die englische Sprache veröffentlicht und ist dafür auch ein defacto Standard. Mittlerweile wurde er auch für andere Sprachen portiert.

### 5.1.4 Synonym Erkennung

Um Wortwiederholungen zu vermeiden, werden in Texten oft Synonyme verwendet. Diese können bei der Feature Bildung auch betrachtet werden. Eine Möglichkeit wäre hier, Listen mit Wortgruppen, die gleiche Bedeutung aufweisen, zu führen. Zu jeder Wortgruppe existiert ein Grundwort, das die Synonyme im Text ersetzen soll. Da die Synonym Erkennung sowohl vor, wie nach dem Porter Stemming stattfinden kann, muss im zweiten Fall darauf geachtet werden, dass die Wörter der Liste auch in die Stammform gebracht werden müssen.

### 5.1.5 Feature Selection

Zum Schluss werden die Wörter aufgrund deren Frequenzen gewichtet. Wörter die hierbei kleinere Gewichte erhalten, können entfernt werden. Ein weit verbreiteter Ansatz ist die TF-IDF Methode.

**Term Frequency** Term Frequency ist ein Mass für die Bedeutung eines Wortes im betrachteten Text. Je häufiger ein Wort im Text vorkommt, desto mehr Gewicht soll es erhalten.

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}$$

$n_{i,j}$  ist hierbei die Anzahl, in der ein Wort  $i$  in einem Dokument  $j$  vorkommt.  $\sum_k n_{k,j}$  ist die Anzahl Wörter im Dokument  $j$

**Inverse Document Frequency** Die inverse Dokumenthäufigkeit ist ein Mass für die Bedeutung des Worts über die Gesamtheit der Dokumente. Es berechnet sich durch das umgekehrte Verhältnis von Anzahl Dokumenten  $n_i$ , die das Wort enthalten, und die Gesamtzahl an Dokumenten im Korpus  $N$ .

$$idf_i = \log \frac{N}{n_i}$$

Ein Wort, das in vielen Dokumenten auftritt, erhält somit eine tiefere Gewichtung.

Werden diese beiden Werte miteinander multipliziert, erhält man die Gewichtung des Wortes:

$$w_{i,j} = tf_{i,j} \cdot idf_i$$

Die Gewichtung ist somit hoch, wenn ein Wort  $i$  häufig in einem Dokument  $j$  vorkommt, aber in anderen selten ist.

Für die Gewichtung können auch weitere Regelungen eingebaut werden. So ist es beispielsweise gewünscht, dass die Wörter des Betreffs von E-Mails stärker gewichtet werden, da dort bereits ungefähr erwähnt wird, um was es bei dieser Nachricht geht. Weiterhin kann nach Anwendungsfall auch ein Wert verstärkt oder abgeschwächt werden.

## 5.2 Klassifizierung

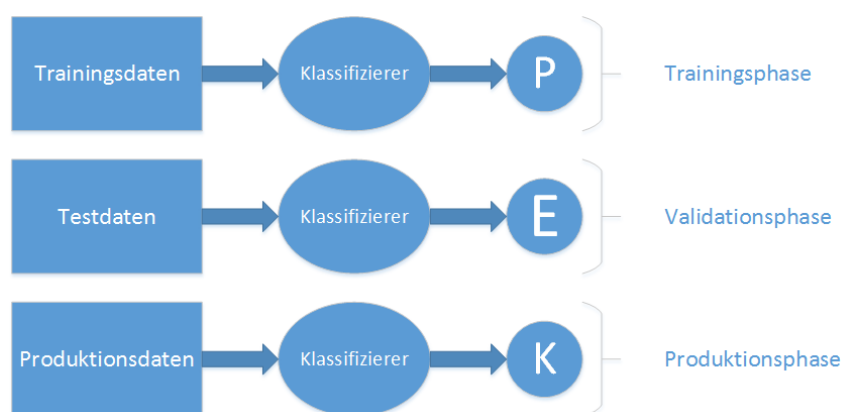
Bei der Klassifikation von Daten ist es die Aufgabe, Daten vordefinierten Kategorien zuzuweisen. Dazu muss ein Klassifizierer Gemeinsamkeiten und Unterschiede zwischen den Daten der verschiedenen Klassen finden. Mit diesem Wissen werden anschliessend Regeln festgelegt, die die Gemeinsamkeiten und Unterschiede beschreiben. Zukünftige Daten werden dann anhand dieser Regeln geprüft und einer Kategorie zugeteilt.

Klassifikatoren gehören zu den Typen der maschinellen Lernalgorithmen. Diese gehören zum Thema der künstlichen Intelligenz und besitzen die Fähigkeit, Modelle anhand von Trainingsdaten zu erstellen. Auf diese Weise sind sie in der Lage, selbst für unbekannte Daten Aussagen treffen zu können.

In den folgenden Kapiteln werden die verschiedenen Ansätze präsentiert, die für eine automatische Klassifizierung verwendet werden können.

### 5.2.1 Lebenszyklus eines Klassifizierers

Im Buch "Algorithm of the intelligent Web" [1] wird der Lebenszyklus eines Klassifizierers beschrieben.



**Abbildung 5.1:** Lebenszyklus eines Klassifizierers

Es ist egal welche Art von Klassifizierer eingesetzt wird. Der Lebenszyklus lässt sich immer in drei Phasen unterteilen.

### 5.2.1.1 Trainingsphase

Jeder Klassifizierer besitzt eine gewisse Anzahl an Parameter (P), die konfiguriert werden müssen. In der Trainingsphase wird versucht, für diejenigen die optimalen Einstellungen zu ermitteln. Dazu werden Trainingsdaten benötigt. Trainingsdaten sind Daten, die schon manuell kategorisiert wurden. Normalerweise werden sie in einem gewissen Verhältnis in effektive Trainingsdaten und in Testdaten aufgeteilt. Erstere werden dazu verwendet, den Lernalgorithmus zu trainieren. Letztere werden für die Validierung des Algorithmus benötigt.

### 5.2.1.2 Validationsphase

Nun werden diejenigen Trainingsdaten verwendet, die nicht für das Training eingesetzt wurden; die Testdaten. Diese werden dem Klassifizierer zum Klassifizieren übergeben. Der Klassifizierer ermittelt nun die Klassenzugehörigkeit, welche anschliessend mit der effektiven Kategorie der Testdatei verglichen wird. Sind alle Testdaten kategorisiert worden, kann die Erfolgsrate (E) des Klassifikators berechnet werden. Die Erfolgsrate ist ein Indikator für die Qualität der Klassifikation.

### 5.2.1.3 Produktionsphase

Bevor mit der Produktionsphase begonnen werden kann, muss zuerst überprüft werden, ob der Klassifizierer eine akzeptable Erfolgsrate liefert. Ist dies nicht gegeben, so sollte die Trainings- und Validationsphase wiederholt werden.

In der Produktionsphase wird der Klassifizierer produktiv eingesetzt, um Daten zu klassifizieren. Dazu werden ihm die Produktionsdaten übermittelt, woraufhin der Klassifizierer die Kategorie (K) bestimmt.

Typischerweise werden die Parameter des Klassifizierers in dieser Phase nicht mehr verändert. Optimal ist jedoch, wenn von Zeit zu Zeit mehr Trainingsdaten zur Verfügung gestellt werden. Dies kann bewerkstelligt werden, indem die klassifizierten Daten manuell überprüft werden und anschliessend mit der korrekten Kategorie gelabelt zu den Trainingsdaten gelegt werden. Da es für den Klassifizierer nicht möglich ist, die neuen Trainingsdaten automatisch zu lernen, muss der gesamte Trainingsvorgang erneut durchgeführt werden. Wenn man keine manuelle Überprüfung durchführen kann, wäre es auch möglich, dem Klassifizierer die Verantwortung zu geben, die Trainingsdaten anzureichern. Dies birgt aber die Gefahr, dass inkorrekte Zuweisungen gelernt werden und sich so Fehler fortpflanzen können.

## 5.2 Typen von Klassifikationsalgorithmen

### 5.2.2.1 Regelbasierter Klassifizierer

Der Regelbasierte Klassifizierer ist vom Konzept her die einfachste Variante. Hierbei werden von Hand Regeln definiert, die festlegen welche Merkmale Texte einer Klasse aufweisen sollen. Als Beispiel könnte die Regel "Wenn das Wort 'Auto' im Text vorkommt, gehört der Text zur Kategorie 'Automobilindustrie'" definiert werden. Über die Hinzunahme von weiteren Regeln, kann so ganz einfach ein Klassifizierer erstellt werden. Der Nachteil dieser Methode ist, dass die Regeln einen viel zu eingeschränkten Problembereich abdecken und so nur für individuelle Lösungen gebraucht werden können. Auch ist der Aufwand relativ gross um genügend Regeln zu definieren.

Die Regelbasierten Klassifikatoren haben einen grossen Nutzen, wenn sie in Verbindung mit lernbasierten Systemen eingesetzt werden. Tritt beispielsweise der Fall ein, dass ein Datensatz ständig falsch zugeordnet wird, kann zur kurzzeitigen Behebung des Problems eine Regel definiert werden. Gleichzeitig kann auf einem anderen System mit Hinzunahme der Änderungen ein neues lernbasiertes System aufgezo-gen werden. Dieses System kann schlussendlich das bestehende ersetzen. Ohne den regelbasierten Klassifizierer müsste man so lange mit dem Problem leben, bis das zusätzlich System die Änderung gelernt hat.

### 5.2.2.2 Naiver Bayes

Naiver Bayes ist eine viel gebrauchte Methode, um Texte zu klassifizieren. Sie funktioniert so, dass sie über den gesamten Textkorpus der Trainingsdaten die Häufigkeiten Wörter bestimmt und anschliessend die relative Wahrscheinlichkeit eines Wortes in Bezug auf die Kategorien berechnet. Das bedeutet, dass ein Wort, das in den Texten einer Klasse besonders viel vorkommt, eine höhere Auftrittswahrscheinlichkeit in jener Klasse besitzt als in den anderen. Sind auf diese Weise die Wahrscheinlichkeiten aller Wörter der Trainingsmenge berechnet worden, können nun Texte kategorisiert werden. Dazu wird für jede Kategorie berechnet, wie hoch die Wahrscheinlichkeit ist, dass der Text zu derjenigen Kategorie gehört. Der Text wird anschliessend der Kategorie mit der höchsten Wahrscheinlichkeit zugeordnet.

Die Methode wird als naiv bezeichnet, da es die Sprache vereinfacht darstellt und die bedingte Wahrscheinlichkeit zwischen den einzelnen Wörter nicht beachtet. Das bedeutet, dass sie in einem Text, in dem über England geschrieben wurde, nicht erkennt, dass das Stichwort "London" stark mit England zusammenhängt.

Trotz diesem Nachteil funktioniert die Methode relativ gut. Des Weiteren benötigt sie nicht viele Trainingsdaten, um ein akzeptables Resultat zu erreichen.

### 5.2.2.3 Fisher Methode

Die Fisher Methode ist ein Klassifizierungsverfahren, das wie der naive Bayes mittels statistischen Eigenschaften der Trainingsdaten arbeitet. Sie berechnet für jedes Wort im Text die Wahrscheinlichkeiten, dass es in den verschiedenen Klassen vorkommt. Dann werden die Wahrscheinlichkeiten kombiniert und geprüft, ob das Set der Wahrscheinlichkeiten einem anderen Set ähnelt. Es wird also nicht nur die Auftrittswahrscheinlichkeit der einzelnen Wörter berücksichtigt, sondern auch die Kombination in der die Wörter auftreten. Die Fisher Methode liefert damit sehr genaue Ergebnisse zurück.

### 5.2.2.4 Support Vector Machine

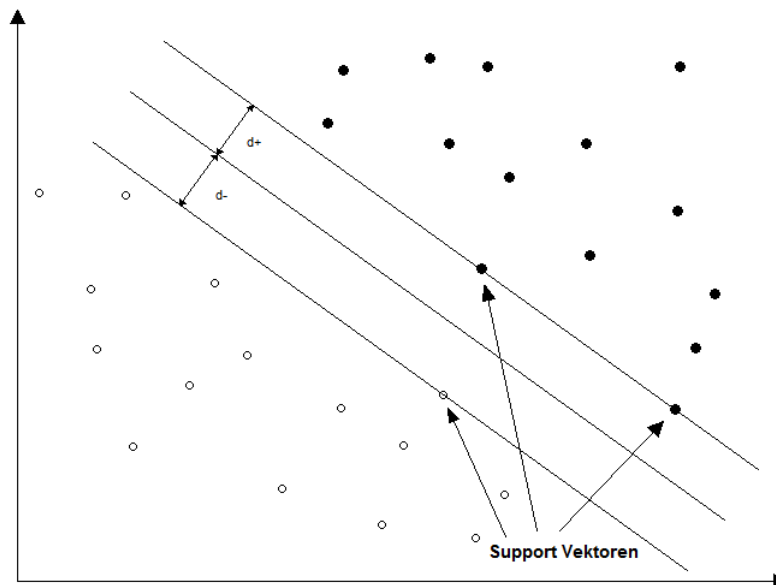


Abbildung 5.2: Support Vector

Die Support Vector Machine (SVM) ist eine weitere beliebte Methode, um Klassifizierungsprobleme zu lösen. Dabei werden die Texte in Vektoren gewandelt. Die Dimensionen der Vektoren repräsentieren dabei Features des Textkorpus. Die Werte der Elemente bedeuten die Gewichtung dieser Features des zu klassifizierenden Textes.

Um nun in Erfahrung zu bringen, zu welcher Kategorie ein Dokument gehört, legt die SVM eine Hyperebene zwischen die Daten. Diese Hyperebene dient als Trennlinie. Die SVM versucht die Ebene möglichst so zu platzieren, dass der Abstand zwischen den Elementen und der Trennlinie maximal ist. Aus diese Weise können Fehl kategorisierungen minimiert werden. Eine grosse Rolle spielen dabei die Datenelemente, die den geringsten Abstand zur Ebene aufweisen. Sie werden als Stützvektoren (support vectors) bezeichnet, weil es mit ihnen möglich ist, die gesamte Hyperebene zu rekonstruieren.

Eine SVM kann nur binäre Probleme lösen. Mit der Hinzunahme von weiteren Maschinen ist es jedoch möglich mehrere Klassen zu unterstützen.

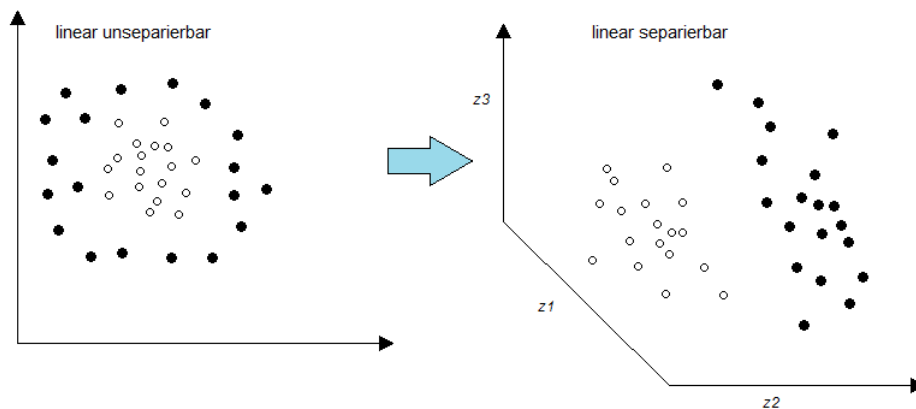
#### 5.2.2.4.1 Training

Für das Training der SVM werden alle Trainingselemente als Vektoren in einen Vektorraum platziert. Über einen langen Prozess wird nun versucht eine Linie zu finden, die die Datenelemente möglichst gut voneinander trennen kann. Da die lineare Trennung nicht immer möglich ist, muss der Kernel Trick angewendet werden.

#### 5.2.2.4.2 Kernel Trick

Eine Hyperebene kann nicht "verbogen" werden. Somit ist eine saubere Trennung von Trainingsdaten nur dann möglich, wenn die Objekte linear trennbar sind. Dies ist in realen Anwendungen selten der Fall, da entweder Messfehler auftreten können oder die definierten Klassen zu ähnlich zueinander sind. Um dennoch solche Fehler zu erlauben, verwendet die SVM den sogenannten Kernel-Trick.





**Abbildung 5.3:** Separierbarkeit

Der Grundgedanke hinter dem Kernel-Trick ist es, nicht linear trennbare Daten in einen Vektorraum höherer Stufe überzuführen. Denn jede Vektormenge wird in einer genügend hohen Dimension separierbar. Nun ist aber die Performance, die für solche Operationen benötigt werden, nicht effizient, da im schlimmsten Fall so viele Dimensionen entstehen können, wie es Features gibt. Die Kernel Methode erlaubt es jedoch, Berechnungen in einer niedrigeren Dimension durchzuführen, ohne die Daten vorher konvertieren zu müssen.

#### 5.2.2.4.3 Complexity-Parameter

Ein wichtiger Parameter für das Trainieren der SVM ist der Complexity- oder Cost-Parameter  $c$ . Er gibt an, wie tolerant die Maschine mit Ausreißern aus der Datenmenge umgehen soll. Wird dafür ein hoher Wert verwendet, wird versucht, eine Hyperebene zu finden, die selbst die Ausreißer richtig voneinander trennen kann; sogenannte strikte Trennung. Ein kleinerer Wert hingegen lässt Falschklassifikationen zu und konzentriert sich verstärkt auf die Ansammlung von Punkten, die viele Gemeinsamkeiten haben. Der Vorteil einer tieferen Komplexität ist, dass das System besser auf die Allgemeinheit abgestimmt ist. Dies ist vor allem bei der Textklassifizierung erwünscht, da die Daten in einer Klasse in unterschiedlichsten Formen auftreten können.

#### 5.2.2.4.4 Relatives Gewicht von positiven zu negativen Klassen

Ein weiterer Parameter  $\gamma$  steht für die Verteilung der Complexity auf die beiden Klassen des binären Problems. Dazu wird von den beiden Klassen eine als positive und die andere als negative Klasse bezeichnet. Ein hoher Wert bedeutet, dass die SVM versucht, eine hohe Complexity auf die positive Klasse anzuwenden. Das heisst, dass es eine Trennlinie sucht, um möglichst viele Elemente dieser Klasse positiv zuzuordnen (tiefe false-positive Rate). Dies ist vor allem erwünscht, wenn die zu betrachtende Kategorie nur wenig Trainingsdaten zur Verfügung stellt. Gegenteilig kann ein tiefer Wert gesetzt werden, wenn eine Klasse mit vielen Elementen betrachtet wird, da zum einen die Wahrscheinlichkeit höher ist, dass die Klasse falsche Trainingsdaten enthält, und zum anderen genügend Informationen zur Verfügung stehen, um Gemeinsamkeiten zwischen den Elementen zu finden.

#### 5.2.2.4.5 Kernel Funktion

Die Kernel Funktion ist das Herzstück jeder SVM. Sie beschreibt den mathematischen Aufbau der Hyperebene und definiert somit die Funktion der binären Trennung. Der standardmässige Kernel wird Linear Kernel genannt. Er legt, wie in den Beispielen zuvor, eine Gerade zwischen die Datenelemente. Andere Kernel, wie der polynomiale Kernel, versucht dasselbe mithilfe einer polynomialen Funktion

höherer Ordnung. Dazu wendet er die polynomiale Funktion auf die Datenelemente an, um sie in einen neuen Raum (feature space) zu transferieren. In diesem Raum wird nun versucht, wieder mittels einer Geraden die Daten zu trennen.

Bekannte Kernel Methoden sind:

- Radiale Basisfunktion (gaussischer) Kernel
- Linearer Kernel
- Polynomialer Kernel

Die Wahl der optimalen Kernel Funktion hängt stark mit der Beschaffenheit der Trainingsdaten ab. In der Praxis wird bei Textklassifizierer oftmals der lineare Kernel angewendet.

#### 5.2.2.4.6 Klassifizierungsvorgang

Nachdem das Training abgeschlossen ist, kann die Maschine für die Klassifizierung von Datenelementen eingesetzt werden. Auf gleiche Weise wie beim Training wird der Input in ein Feature-Vektor gewandelt und in den Vektorraum platziert. Über die Position des Datenelements in Bezug auf die Hyperebene kann nun entschieden werden, zu welcher Klasse das Datenelement gehört.

#### 5.2.2.4.7 Multiclass Support Vector Machine

Support Vector Machines lassen sich nur für binäre Probleme anwenden. Das heisst, es kann nur zwischen zwei verschiedenen Klassen entschieden werden.

Wenn dem Eingabewert mehr als nur zwei mögliche Klassen zugewiesen werden sollte, handelt es sich um ein Mehrklassenproblem. Zu diesem Zweck teilt man das Mehrklassenproblem in mehrere binäre Probleme auf und löst diejenige unabhängig voneinander.

Ein  $n$  Klassen Problem kann immer auf  $n(n - 1)/2$  binäre Probleme abgebildet werden, bei denen jede Klasse gegen jede andere ein Mal entgegentreten muss. Beim Trainieren heisst das, dass für jedes binäre Problem eine Maschine erstellt werden muss. Jede Maschine kennt dabei nur die Trainingselemente derjenigen Klassen und versucht zwischen ihnen eine optimale Trennlinie zu finden.

Bei der Klassifizierung muss der Eingabewert jeder Maschine übermittelt werden. Auf allen Maschinen wird nun die Klassifizierung durchgeführt. Dabei gibt jede Maschine eine Stimme an die gewonnene Klasse. Nachdem alle Maschinen ihre Stimme abgegeben haben, wird der Input derjenigen Klasse zugeteilt, die am meisten Stimmen erhalten hat. Diese Methode nennt sich All-vs-All.

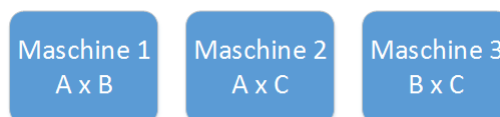
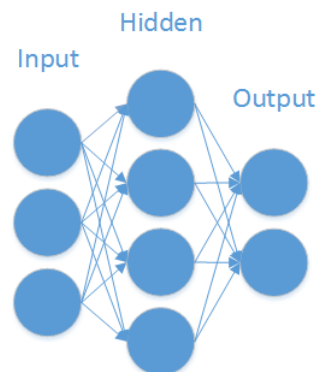


Abbildung 5.4: Binäre Klassifikationsprobleme

In dem Beispiel oben hat die Klasse  $A$  in der Maschine 1 gewonnen, Klasse  $C$  in der Maschine 2 und 3. Somit hat die Klasse  $C$  die Abstimmung mit zwei Stimmen gewonnen und die Multiclass SVM klassifiziert den Input als  $C$ .

## 5.2.2.5 Künstliches neuronales Netz

### 5.2.2.5.1 Allgemein



**Abbildung 5.5:** Künstliches neuronales Netzwerk

Künstliche neuronale Netze, meistens auch nur neuronale Netze (NN) genannt, haben ihren Ursprung in der Biologie. Inspiration für den Aufbau der neuronalen Netze kam bei der Untersuchung des zentralen Nervensystems. Ein neuronales Netz besteht aus Knoten; sogenannte Neuronen; und deren Verbindungen zueinander; auch Synapsen genannt. Aufgrund dieses Aufbaus können neuronale Netze gut als Graphen dargestellt werden.

Neuronale Netze werden in der Regel in hintereinander liegenden Schichten angeordnet. Die erste Schicht wird Input Layer genannt, die hinterste Schicht Output Layer. Alle Schichten die sich dazwischen befinden nennt man Hidden Layer. Beim Bau eines neuronalen Netzes ist nicht festgelegt, wie viele Neuronen in welcher Schicht verwendet werden sollen, wie die Synapsen verlaufen oder wie aus wie vielen Schichten der Hidden Layer bestehen soll. Diese Variablen sind abhängig von der Problemdomäne und müssen vor jedem Einsatz evaluiert werden.

Es kann jedoch zwischen verschiedenen Topologien unterschieden werden:

- **Einschichtiges Feedforward Netze** sind die einfachsten neuronalen Netze. Der Input Layer ist direkt mit dem Output Layer verknüpft, einen Hidden Layer gibt es nicht. Aufgrund ihrer Einfachheit sind einschichtige Feedforward Netze für reale Anwendungen nicht zu gebrauchen. Die Feedforward Eigenschaft sagt aus, dass die Ausgaben der Neuronen nur in Richtung Output Layer weitergeleitet werden kann.
- **Mehrschichtiges Feedforward Netze** besitzen mindestens einen Hidden Layer. Auch hier wird die Ausgabe nur in Richtung Output Layer weitergeleitet.
- **Rekurrentes Netze** besitzen auch rückgerichtete Verbindungen und erhalten somit eine Rückkopplung. Diese Rückkopplungen werden mit einer Zeitverzögerung versehen. Das heisst, dass Resultate früherer Berechnungen einen Einfluss auf neue Berechnungen haben.
- **Vollständig verbundene Netze** erlauben Verbindungen zwischen allen Neuronen ausser direkten Rückkopplungen. Ausserdem müssen die Verbindungen symmetrisch aufgebaut sein.

### 5.2.2.5.2 NN für Klassifizierung

Neuronale Netze sind eine weitere beliebte Methode zur Klassifizierung. Normalerweise wird ein mehrschichtiges Feedforward Netz mit Backpropagation verwendet. Neuronale Netze bieten den Vorteil, dass sie für viele verschiedene Klassifizierungsprobleme eingesetzt werden können. Ihr Nachteil ist aber, dass sie eine grosse Blackbox sind. Denn im Vergleich zu anderen Methoden ist es viel schwieriger nachzuvollziehen, warum das Resultat entsprechend ausfällt.

Als Eingabewert werden wie bei der SVM auch Vektoren verwendet. Die Länge der Input Vektoren und somit auch die Anzahl der Input Neuronen entsprechen der Anzahl an Features. Wie viele Neuronen und Schichten im Hidden Layer verwendet werden müssen, ist nicht festgelegt und kann je nach Problemdomäne und Qualität der Trainingsdaten stark variieren. Die Anzahl Neuronen im Output Layer ist abhängig davon wie das NN für die Klassifizierung eingesetzt wird. Es gibt zwei Varianten: In der ersten Variante existiert pro Klasse ein Neuron im Output Layer. In der anderen Variante wird pro Klasse ein NN erstellt und trainiert. Jedes NN besitzt nur ein Neuron im Output Layer. Das Output Neuron macht dann die Aussage, ob der Eingabewert zu der Kategorie gehört oder nicht. Ist der Ausgabewert 1 oder in der Nähe von 1 so gehört der Input mit grösster Wahrscheinlichkeit zu dieser Kategorie, bei 0 wiederum nicht.

Eine weitere Variable im System ist die Lernrate. Sie bestimmt, wie schnell gelernt werden soll. Die Gefahr ist aber, dass, wenn sie zu klein gewählt wird, deshalb viele Trainingsschritte vollführt werden müssen, bis ein Minimum erreicht wird. Das andere Extrem ist, wenn die Lernrate zu gross ist. Dann kann es vorkommen, dass das Annäherungsverfahren zu grosse Sprünge macht und oberhalb eines Minimums ständig hin und her springt.

Der Vorteil gegenüber einer SVM ist, dass die Methode beim einmaligen durchlaufen eines Testbeispiels eine Klassenzugehörigkeit herausgibt. Dies steht im Gegensatz zur SVM, welche für jede Klasse eine Klassenzugehörigkeit berechnet werden muss und am Schluss der beste Score für die Bestimmung verwendet wird.

## 5.2.3 Umgang mit wenig Trainingsdaten

Das Anlegen von Trainingsdaten kann sehr aufwändig und zeitintensiv sein, da es manuell durchgeführt werden muss. Dennoch ist es unausweichlich, dass für einen Klassifizierer mit geringer Fehlerrate viele Datenmengen für den Lernvorgang angelegt werden müssen. Dieser Umstand versucht das semi-überwachte Lernen zu umgehen, indem es neben klassifizierten Trainingsdaten auch unklassifizierte Daten als Lernsatz verwendet. Die Idee dahinter ist es, dass unter den unklassifizierten Daten Elemente gibt, die mit Bestimmtheit kategorisiert werden können. Diese Elemente können deshalb auch als Lerndaten dienen.

Für das semi-überwachte Lernen gibt es verschiedene Ansätze

### 5.2.3.1 Low-Density Method

Unter dieser Kategorie gehört die Transductive Support Vector Machine (TSVM). Während das Ziel der unterstützt lernenden SVM ist, dass der Abstand zwischen den Trainingsdaten und der Hyperebene maximal ist, versucht die TSVM, eine Linie zu finden, die zu allen Daten, ungelabelte und gelabelte, maximalen Abstand hat.

### 5.2.3.2 Self-Training

Es wird zuerst mit den klassifizierten Daten gelernt. Anschliessend bestimmt man die Klassen der unsortierten Daten. Diejenigen Eingaben, die am zuversichtlichsten einer Klasse zugeordnet werden konnten, werden zu den Trainingsdaten hinzugefügt, damit sie für ein weiteres Training zur Verfügung stehen.

### 5.2.3.3 Co-Training

Die Idee ist es, dass die Daten in verschiedene Teilbereiche aufteilt werden. Beispiel wäre es bei E-Mail Daten möglich, den Betreff-Header und der Inhalt des Mails als zwei separate Teilbereiche anzuschauen. Für jeden Teilbereich, wird nun ein Klassifizierer  $C_1, C_2, \dots, C_n$  erstellt und trainiert. Anschliessend wird auf einer dieser Maschinen die Zuteilung der nicht klassifizierten Daten vorgenommen und diejenigen Daten gewählt, welche mit grosser Bestimmtheit richtig kategorisiert werden konnten. Diese Datenelemente werden dann allen anderen Klassifizierer als Trainingsdaten übergeben. Nun wird mit den anderen Maschinen fortgefahren, bis keine unklassifizierten Daten mehr vorhanden sind.

## 5.3 Spracherkennung

Um den Text weiterverarbeiten und kategorisieren zu können, ist es zwingend notwendig zuerst die Sprache des Eingabetext zu erkennen. Andernfalls lassen sich Aufgaben wie das Word Stemming und die Stopp-Wörter Entfernung nicht umsetzen. Dies führt dazu, dass die Anzahl der Schlüsselwörter erhöht und somit der gesamte Lernprozess verlangsamt wird.

In den letzten Jahrzehnten wurden mehrere Ansätze entwickelt, wie dies bewerkstelligt werden kann. So ist es heute möglich, dass in ca. 99% der Fälle die Sprache richtig erkannt wird. Auch dann, wenn nur ein kleiner Textausschnitt zur Verfügung steht.

Gängige Methoden zur Spracherkennung sind:

- Common Words and Unique Letter Combinations
- Markov Modelle
- N-Gramm
- Compression based approach with PPM

### 5.3.1 Common Words and Unique Letter Combinations

Die Idee dahinter ist, dass für jede Sprache diejenigen Wörter gespeichert werden, die häufig auftreten. Der auszuwertende Text wird dann mit allen erfassten Wörtern verglichen und aufgezeichnet, aus welcher Sprache wie viele Wörter übereinstimmen. Zum Schluss erhält somit jede Sprache eine Wertung, welche gegeneinander verglichen werden kann.

Dies ist ein guter Ansatz, falls der auszuwertende Text eine gewisse Länge hat. Bei kurzen Texten ist dieser Ansatz nicht brauchbar, weil die Wahrscheinlichkeit, dass ein Wort aus einer der Listen erscheint relativ klein ist. Besteht die Eingabe nur aus langen Texten so bietet diese Methode den Vorteil, dass sie einfach zu implementieren ist und wenig Rechenleistung braucht.

Ein verwandter Ansatz ist die Unique Letter Combinations Methode. Dabei wird davon ausgegangen, dass jede Sprache für sich typische Buchstabenkombinationen besitzt. Die Auswertung findet dabei gleich wie bei der Common Words Methode statt. Auch diese Methode hat die Schwäche, dass der Text eher lang sein muss, damit sie genügend gute Aussagen treffen kann.

### 5.3.2 Markov Modelle

Dazu werden Markov Modelle zu jeder Sprache basierend auf dem entsprechenden Alphabet erstellt und dann mit Trainingsdaten gefüttert. Je mehr Trainingsdaten vorhanden sind, desto genauer werden die Modelle. Danach wird für jedes Modell berechnet wie hoch die Wahrscheinlichkeit ist, dass es den zu analysierenden Text erzeugen kann. Die höchste Wahrscheinlichkeit sagt aus, welchem Modell und somit auch welcher Sprache der Text angehört.

### 5.3.3 N-Gramm

Texte die in gleicher Sprache geschrieben wurden, besitzen immer ähnliche Häufigkeiten der Buchstaben. So ist beispielsweise der Buchstabe "E" mit 17.4% (Wikipedia) der häufigste Buchstabe in der deutschen Sprache.

Für jede Sprache lässt sich so ein Profil erstellen. Diese Sprachprofile lassen sich für die Identifizierung der Sprache gebrauchen, indem das gleiche Verfahren auf die zu analysierenden Texte angewendet wird. Anschliessend werden sie mit dem Sprachprofilen verglichen.

Einzelne Buchstaben sind sogenannte Monogramme. Also N-Gramme der Länge 1. Durch Hinzunahme von weiteren Buchstaben lassen sich höhere N-Gramme bilden, deren relative Häufigkeiten auch das Profil einer Sprache beschreiben. Auf diese Weise sind die Sprachen noch klarer voneinander unterscheidbar.

Im Vergleich zu den ersten beiden Methoden ist die N-Gramm Methode robuster, da allfällige Rechtschreibfehler das Resultat nicht gross beeinflussen. Auch existieren zu allen gängigen Sprachen bereits N-Gramm Statistiken, welche verwendet werden können.

### 5.3.4 Compression based approach mittels PPM

Diese Idee geht davon aus, dass komprimierte Texte aus der gleichen Sprache ähnliche Eigenschaften haben. PPM steht für "Prediction by partial matching". Beispieltex te von verschiedenen Sprachen werden unter der Verwendung des PPM Algorithmus komprimiert. Wenn ein Text analysiert werden soll wird er komprimiert. Anschliessend wird die komprimierte Datei mittels Markov-Analyse der einzelnen Bits mit bestehenden Sprachmodellen verglichen.

## 6 Erkenntnisse

### 6.1 Algorithmenwahl

#### 6.1.1 Sprach Identifizierung

Im Rahmen der Bachelor Arbeit wurde entschieden, dass zur Sprachidentifizierung N-Gramm als Algorithmus verwendet wird. Der Algorithmus ist robust gegenüber Rechtschreibfehlern und funktioniert auch gut unter der Verwendung von kurzen Texten, wie E-Mails. Ein weiterer Vorteil in der Verwendung dieses Algorithmus ist, dass er einfach zu implementieren ist. Auch braucht die N-Gramm Methode weniger Performance als die *Common Words and Unique Letter Combinations*-Methode.

#### 6.1.2 Word Stemming

Für das Word Stemming wird der Porter Stemming Algorithmus eingesetzt, da er ein quasi Standard in diesem Bereich ist. Zudem unterstützt er neben den gewünschten Sprachen (Deutsch, Englisch, Französisch, Italienisch) auch noch einige weitere. Dies lässt die Möglichkeit für eine spätere Erweiterung offen.

#### 6.1.3 Text Klassifizierung

Keiner der betrachteten Algorithmen scheint sich hinsichtlich der Genauigkeit von den anderen abzuheben. Diese Erkenntnis ist auch in vielen Büchern und Arbeiten ersichtlich. Schlussendlich spielt es immer eine Rolle, in welcher Domäne der Klassifikator eingesetzt wird. Das heisst, wie viele Trainingsdaten vorhanden sind und wie gut beziehungsweise wie ausbalanciert diejenigen sind.

Nachfolgend sind nochmals die Stärken und Schwächen der aller Ansätze aufgelistet:

Fisher/Bayes	SVM	Neuronale Netze
<ul style="list-style-type: none"> <li>+ Transparent, da die Schlüsselwörter und deren Auftretswahrscheinlichkeiten verständlich sind</li> <li>+ Trainingsvorgang ist sehr schnell</li> <li>+ Anpassung im Betrieb möglich. Wichtig wenn manuelle Zuweisungen während der Einsatzzeit vorgenommen werden</li> <li>+ Gute Ergebnisse auch bei wenig Trainingsdaten</li> <li>+ Lernt Zusammenhänge zwischen den Wörtern</li> </ul>	<ul style="list-style-type: none"> <li>+ Sehr mächtig, da viele Optimierungsmöglichkeiten</li> <li>+ Viel dokumentiert</li> <li>+ Variantenreich in Bezug auf Problemlösung</li> <li>+ Mathematisch erklärbar</li> <li>+ Können komplexe wie auch einfache Systeme lernen</li> <li>+ Findet globales Minima</li> </ul>	<ul style="list-style-type: none"> <li>+ Sehr mächtig, da viele Optimierungsmöglichkeiten</li> <li>+ Kann für komplexe Systeme genutzt werden</li> <li>+ Echte Implementierung von Mehrklassenproblemen möglich</li> <li>+ Lernt Zusammenhänge zwischen den Wörtern</li> </ul>



<ul style="list-style-type: none"> <li>- Kein Optimierungspotential</li> <li>- Können komplexere Systeme nicht lernen</li> <li>- Qualität nimmt proportional zu den Trainingsdaten ab</li> </ul>	<ul style="list-style-type: none"> <li>- Blackbox: Entscheidungen schwierig erklärbar</li> <li>- Keine wirkliche Mehrklassen Implementation möglich (Ansatz mit mehreren binären Problemen)</li> <li>- Trainingsvorgang sehr langsam</li> <li>- Sehr viele Einstellungsmöglichkeiten</li> <li>- Es werden keine Zusammenhänge zwischen den Wörtern gelernt</li> <li>- Kaum Kontrolle über die Richtigkeit des Trainingsvorgang</li> <li>- Die Anzahl an SVMs bei Mehrklassenproblemen steht in einem quadratischen Verhältnis zu den Anzahl Klassen. Dies verlängert den gesamten Trainingsprozess</li> </ul>	<ul style="list-style-type: none"> <li>- Blackbox: Entscheidungen schwierig erklärbar</li> <li>- Ideale Architektur kann sich je nach Eingabedaten variieren</li> <li>- Trainingsvorgang sehr langsam</li> <li>- Sehr viele Einstellungsmöglichkeiten</li> <li>- Es werden keine Zusammenhänge zwischen den Wörtern gelernt</li> <li>- Kaum Kontrolle über die Richtigkeit des Trainingsvorgang</li> <li>- Da zuerst zufällige Gewichtungen der Neuronen Inputs gewählt werden, werden auch bei gleichen Trainingsdaten immer andere Genauigkeiten erreicht</li> <li>- Gefahr, dass lokale Minimas gefunden werden</li> <li>- Theoretisch besteht die Gefahr des Overfittings</li> </ul>
--	---	--

**Tabelle 6.1:** Vergleich Klassifikationsalgorithmen

### 6.1.3.1 Schlussfolgerung

Die Fisher und die naive Bayes Methode werden voraussichtlich nicht betrachtet. Sie sind zwar sehr einfach zu implementieren, scheinen aber für komplexere Probleme keine guten Ergebnisse zu erzielen. Dies lässt für einen Entscheid nur noch die Neuronale Netze und die SVM übrig.

Nach Gegenüberstellung wurde entschieden, dass für dieses Projekt ein Ansatz mit SVM gebraucht wird. Gründe dafür sind die, dass die SVM gut mit hochdimensionalen Eingabedaten funktioniert. Da Texteingaben sehr schnell viele Features aufweisen können, scheint dies der richtige Ansatz zu sein. Auch der Aspekt, dass weniger Trainingsdaten als bei neuronale Netze benötigt werden, spricht für den Einsatz einer Support Vektor Maschinen. Der letzte Ausschlaggebende Punkt war, dass es für neuronale Netze keine vorgegeben ideale Architektur gibt. Es müsste experimentell bestimmt

werden, welche Architektur gute Resultate erzielt. Als Folgerung kann es sein, dass, falls andere Trainingsdaten verwendet werden, das Model schlechte Resultate liefert.

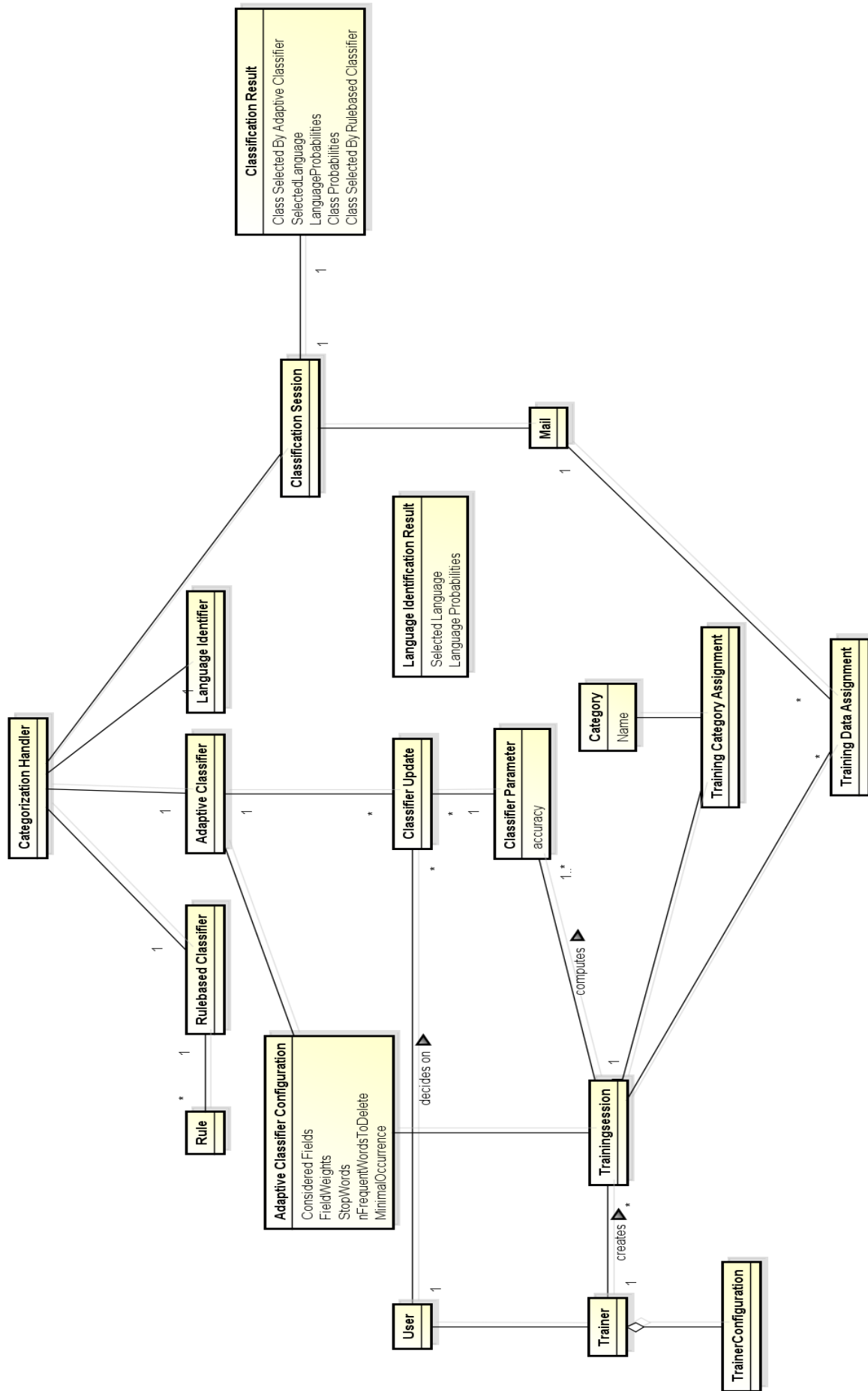
## 6.2 Erste Architektorentwürfe

### 6.2.1 Domain Model

Ein Trainer initialisiert den Trainingsvorgang. Im Trainingsvorgang werden die Parameter des adaptiven Klassifizierers ermittelt. Wenn das Training abgeschlossen ist, kann der Benutzer entscheiden, die Parameter auf den bereits bestehenden Klassifizierer anzuwenden.

Die Klassifikation der Mails wird vom CategorizationHandler durchgeführt. Dafür wird ein Klassifikationsvorgang gestartet, der durch die Klasse Classification Session repräsentiert wird. Bei diesem Vorgang kommen mehrere Elemente zum Zug, die Zwischenresultate berechnen.

Klasse	Beschreibung
Language Identifier	Der Language Identifier produziert die Resultate der Sprachidentifizierung. Dazu gehört die Zugehörigkeitswahrscheinlichkeit und die endgültige Sprachzuweisung.
Adaptive Classifier	Der adaptive Klassifizierer ist verantwortlich für die eigentliche automatische Klassifikation von Eingangsdaten. Dazu ermittelt er für jede ihm bekannte Kategorie eine Wahrscheinlichkeit. Er wird über einen Trainingsvorgang initialisiert.
Rulebased Classifier	Der Rulebased Classifier ist verantwortlich für die definitive Zuweisung eines Mails zu einer Kategorie. Hierfür besitzt er mehrere Regeln, die von einem Benutzer definiert wurden. Die Eingangsdaten, sowie die Resultate der vorhergehenden Schritte werden ihm übergeben. Anschliessend werden die Regeln angewandt. Greift eine der Regeln zu, so kann sie dem eingehenden Mail eine neue Kategorie zuordnen. Falls keine Regel zur Anwendung kommt, gilt die Entscheidung des adaptiven Klassifizierers.

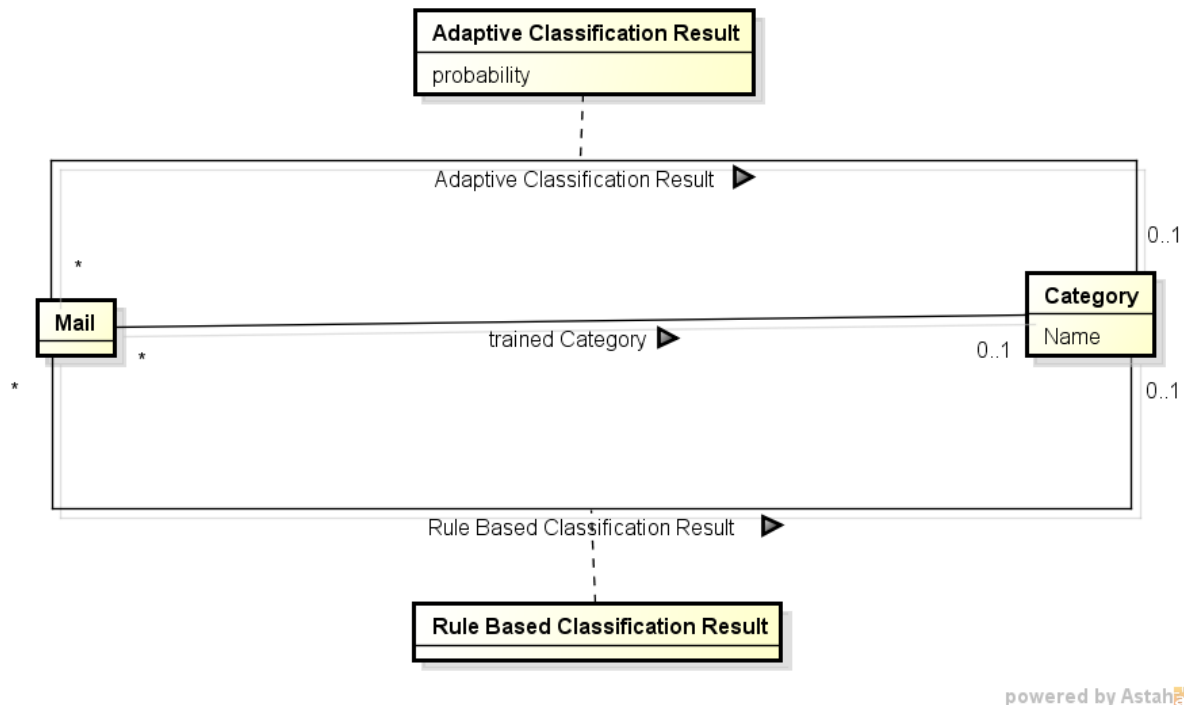


powered by Astah

Abbildung 6.1: Domainmodell  
Selbstlernende Software zur Analyse von Texten

## 6.2.2 Mail-Kategorie Beziehung

Ein Mail kann verschiedene Beziehungen zu einer Kategorie haben. Alle diese Beziehungen sind optional, da ein E-Mail zu Trainingszwecken oder oder durch automatisierte Klassifikation einer Kategorie hinzugefügt werden kann. Folgende Grafik soll das illustrieren



powered by Astah

Abbildung 6.2: Domainmodell

Beziehung	Beschreibung
Adaptive Kategorizuweisung	Diese Beziehung wird vom adaptiven Klassifizierer erstellt. Dazu berechnet er eine Zugehörigkeitswahrscheinlichkeit, da er keine endgültige Aussage über die Zuweisung machen kann. Diese wird mittels dem probability Feld angegeben.
Trainierte Kategorie	Da die E-Mails auch als Trainingsdaten verwendet werden können, muss eine Beziehung bestehen, die das E-Mail als Trainingsdatenelement identifiziert.
Regelbasierte Kategorizuweisung	Diese Beziehung wird zusammen mit der adaptiven Zuweisung im Klassifikationsvorgang erstellt. Sie repräsentiert das Resultat des regelbasierten Klassifizierers.

# 7 Lösung

## 7.1 Trainingsdaten

Für die Umsetzung werden als Trainingsdaten E-Mails der Firma Enron verwendet. Der Grosskonzern Enron hatte seinen Firmensitz in Houston, Texas. Im Jahre 2001 kam die Firma gross in die Schlagzeilen wegen Bilanzfälschung im noch nie dagewesenem Ausmass. In Folge der Untersuchung wurden E-Mails von 150 Benutzern offen gelegt. Die offengelegten Daten umfassen ca. eine halbe Million E-Mails.

In der Arbeit wird ein reduziertes Datenset verwendet. Es werden nur die E-Mails von den sieben grössten Mailbenutzern verwendet. Dieses wird von Ron Bekkerman, einem Spezialisten im Gebiet der Maschinen Lernalgorithmen, auf seiner Webseite zur Verfügung gestellt.

Webseite: <http://people.cs.umass.edu/~ronb/>

Übersicht des Datensets:

Benutzer	Anzahl Ordner	Anzahl Nachrichten	Nach-	Mindestanzahl pro Ordner	Höchstanzahl pro Ordner
beck-s	101	1971		3	166
farmer-d	25	3672		5	1192
kaminski-v	41	4477		3	547
kitchen-l	47	4015		5	715
lokay-m	11	2489		6	1159
sanders-r	30	1188		4	420
williams-w3	18	2769		3	1398

**Tabelle 7.1:** Trainingsdaten Übersicht

Nachteilig an den Daten ist, dass die Mails nur in Englisch geschrieben sind. Da die Software auch mit deutschen, französischen und italienischen Mails umgehen muss, wären mehrsprachige Daten wünschenswert. Jedoch gibt es keine öffentlichen mehrsprachige Mails im Internet, die dafür verwendet werden könnten.

## 7.2 Frameworks und Libraries

In diesem Kapitel werden Third Party Frameworks beschrieben, welche für Teile des Projektes verwendet werden. Das Kriterium aller beschriebener Frameworks ist es, dass sie für .Net verfügbar sind. Ausserdem sollen sie für kommerzielle Projekte einsetzbar sein und sollen die geforderten Aufgaben effizient erledigen.

### 7.2.1 NTextCat für Sprachidentifikation

NTextCat ist ein Framework zur Sprachidentifizierung welches unter der MIT Lizenz vertrieben wird. Inspiriert wurde das Projekt durch das bekannte Perl Tool TextCat. Standardmässig werden 280+ Sprachen unterstützt. Die empfohlene Mindestlänge für Texte beträgt 50 Wörter. Falls die Anzahl der zu erkennenden Sprachen beschränkt wird, kann bereits nach 10 Wörtern gute Ergebnisse erzielt werden.

Es werden zu jeder Sprache mehrere verschiedene Wörterbücher mitgeliefert. Je nach Wahl des Wörterbuchs kann dies die Erfolgsrate der Spracherkennung verbessern.

Verwendete Version: 0.2.0a

Webseite: <http://ntextcat.codeplex.com/>

### 7.2.2 Lucene.Net

Lucene ist ein Projekt der Apache Software Foundation. Ursprünglich wurde es in Java geschrieben, mittlerweile aber auch auf .Net portiert. Hauptziel des Frameworks ist die Volltextsuche. Die Suchfunktion von Wikipedia ist beispielsweise mit Lucene realisiert. Das Framework implementiert einige komplexe Algorithmen wie TF-IDF oder K-Nearest-Neighbor, die verwendet werden könnten.

Lucene.Net wird in der Arbeit für das Wordstemming und Tokenizing verwendet.

Verwendete Version: 3.0.3

Webseite: <http://lucenenet.apache.org/>

### 7.2.3 Accord.NET Framework

Das Accord .Net Framework basiert auf dem AForge.Net Framework. Das AForge.Net Framework ist ein Framework für Computer Vision und Künstliche Intelligenz. Es bietet unter anderem eine API für Maschinen Lernalgorithmen an. Zu diesen zählen die neuronalen Netze, Support Vektor Maschinen, Entscheidungsbäume, K-Means, naiver Bayes und vieles mehr.

Das Framework ist unter der LGPL v3 Lizenz erhältlich.

In dieser Arbeit wird Accord.Net für die Support Vector Machine Implementierung verwendet. Es ist ein Open Source Framework und wird immer noch aktiv weiterentwickelt.

Verwendete Version: 2.8.2.0

Webseite: <https://code.google.com/p/accord/>

### 7.2.4 DotNetZip

DotNetZip ist eine Open Source Zip Library. Im Projekt wird sie verwendet um die Enron Trainingsdaten zu extrahieren.

Verwendete Version: 1.9.1.8

Webseite: <http://dotnetzip.codeplex.com/>

### 7.2.5 log4net

log4net ist ein Framework zum Loggen von Statusmeldungen aus der Applikation. Das Framework ist Teil der Apache Software Foundation und ist praktisch defacto Standard, wenn es ums Logging geht.

Verwendete Version: 1.2.11

Webseite: <http://logging.apache.org/log4net/>

### 7.2.6 OpenPop.NET

OpenPop.NET ist eine Open Source Implementation eines POP3 Client. In der Arbeit wird der MIME Parser der Library verwendet, da die Enron Trainingsdaten im MIME Format vorliegen.

Verwendete Version: 2.0.4.369

Webseite: <http://hpop.sourceforge.net/>

### 7.2.7 Moq

Moq ist ein Mocking Framework. Es kommt im Projekt beim Unit Testing zum Einsatz.

Verwendete Version: 4.0.10827

Webseite: <https://code.google.com/p/moq/>

### 7.2.8 Graph#

Graph # ist ein Graph Framework. Es bietet Datenstrukturen zur visuellen Darstellung von Graphen in WPF-Applikationen.

Im Projekt wird es verwendet um das Resultat einer Categorization Handler Chain zu visualisieren.

Verwendete Version: 1.0

Webseite: <http://graphsharp.codeplex.com/>

## 7.3 Trainingsvorgang

Die SVM gehört zu den Machine Learning Algorithmen. Deren Eigenschaften ist es, dass sie Systeme selbstständig lernen können. Damit dies bewerkstelligt werden kann, benötigen die Maschinen vorkategorisierte Daten; die Trainingsdaten. Der gesamte Lernprozess wird auch Training genannt.

### Erfolgsrate

Mithilfe eines Validierungsvorgangs kann für eine trainierte Maschine eine Erfolgsrate berechnet werden. Sie ist ein Mass für die Qualität der Maschine.

### Nicht Konvergierbare Optimierung

Beim Trainieren einer Maschine kann es vorkommen, dass der Vorgang nicht konvergiert. Das bedeutet, dass das Optimierungsproblem, eine optimale Trennlinie zu finden, nicht in absehbarer Zeit gelöst werden konnte. In diesem Falle wird eine `ConvergenceException` geworfen und der Trainingsvorgang der Maschine wird abgebrochen. Ein weiterer Grund für die nicht Konvergierung kann auch sein, dass eine hohe Complexity festgelegt wurde und somit eine zu strikte Trennung verlangt wird.

### Laufzeit des Trainingsvorgangs

Die Findung der optimalen Trennlinie zwischen zwei Kategorien ist ein quadratisches Optimierungsproblem. Die Dimension des Vektorraums spielt bei der Berechnung keine grosse Rolle. Stattdessen nimmt die Komplexität proportional zu den Anzahl Trainingsdaten zu. Die Laufzeit variiert zwischen  $\mathcal{O}(n_{features} * n_{samples}^2)$  und  $\mathcal{O}(n_{features} * n_{samples}^3)$ , je nachdem wie gut der Cache genutzt werden kann.

Bei einer Multiclass SVM werden  $n_{classes} * (n_{classes} - 1) / 2$  Maschinen benötigt. Die Laufzeit nimmt dann folgenden Wert an:

$$\mathcal{O}(n_{classes}^2 * n_{features} * n_{samples}^3)$$

### Parameteroptimierung

Klassifikationsalgorithmen haben den Nachteil, dass sie wenig bis gar nicht kontrollierbar sind. Die vielen Einstellungsmöglichkeiten, die sie bieten, hängen stark von den Eigenschaften der Trainingsdaten ab. So ist es zum Beispiel nicht unüblich, dass zwei Text-Klassifizierer mit ähnlichen Trainingsdaten völlig verschiedene Parametereinstellungen haben können. Verstärkend kommt hinzu, dass die Vielzahl von benötigten Trainingsdaten die Kontrolle weiter minimiert.

Um diesem Problem entgegenzukommen, ist es nötig, bei jedem Trainingsvorgang die Parameter neu festzulegen. Dazu werden mehrere Klassifizier-Modelle erstellt, die gegeneinander verglichen werden. Ein Modell ist dabei nichts anderes als ein Setup, das die Parameter des Klassifizierers festhält.

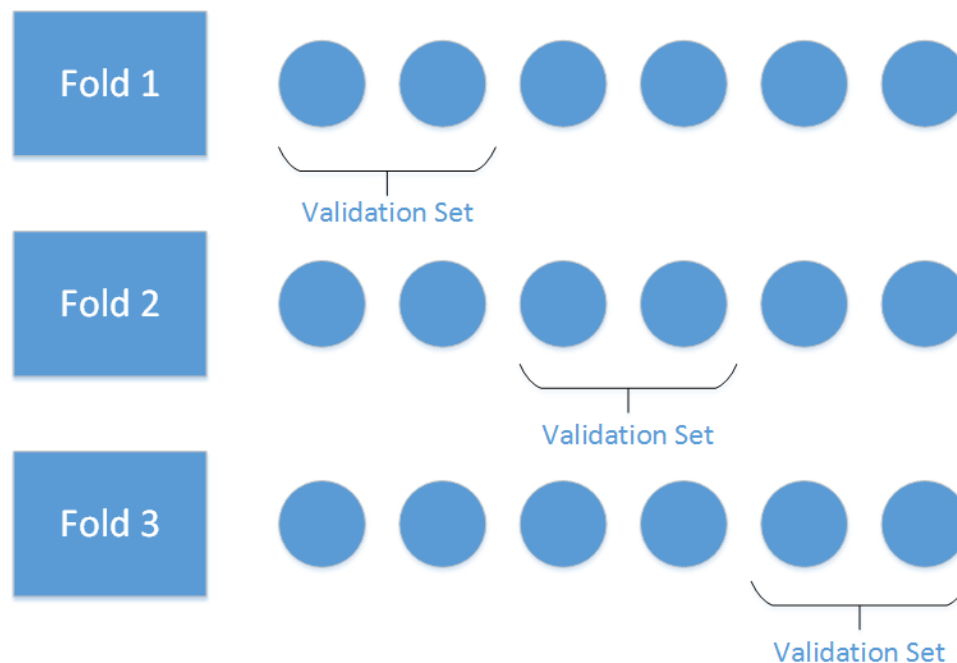
Für den Vergleich der Modelle wird das Konzept der K-fold Cross Validation verwendet.



## K-fold Cross Validation

K-fold Cross Validation ist ein Verfahren aus der Statistik, um die Genauigkeit von Modellen zu bestimmen. Es geht von der Idee aus, dass Trainingsdaten gleichzeitig als Testdaten und umgekehrt verwendet werden können.

Beim K-fold wird eine Menge von Trainingsdaten in  $K$  Subsets (folds) unterteilt. Anschliessend wird jedes Subset ein einziges Mal für die Validierung verwendet, während das Modell mit den Daten der restlichen Sets trainiert wird. Die somit berechneten Erfolgsraten werden gemittelt.



**Abbildung 7.1:** 3-Fold Cross Validation

Üblicherweise wird ein 10-fold Cross Validation angewendet.  $K$  kann aber beliebig variiert werden, so dass selbst jedes Trainingsdaten-Element einzeln als Subset verwendet werden kann. Diese Variante wird als leave-one-out Cross Validation bezeichnet.

Sind die verschiedenen Modelle miteinander verglichen worden, wird das Modell mit der besten Erfolgsrate ausgewählt und für ein finales Training angewendet.

## GridSearch für die Modellauswahl

Bei der Modell Generierung wird der GridSearch Algorithmus angewendet. Er ist eine simple Variante des Brute Force Algorithmus und ist gerade bei Parameteroptimierungen von Klassifizierern sehr beliebt.

Ihm werden mehrere Listen von Werten für verschiedene Parameter übergeben. Der Algorithmus generiert anschliessend alle Parameter Kombinationen und erstellt für jede einzelne ein Klassifizierermodell.

### 7.3.1 Ablauf

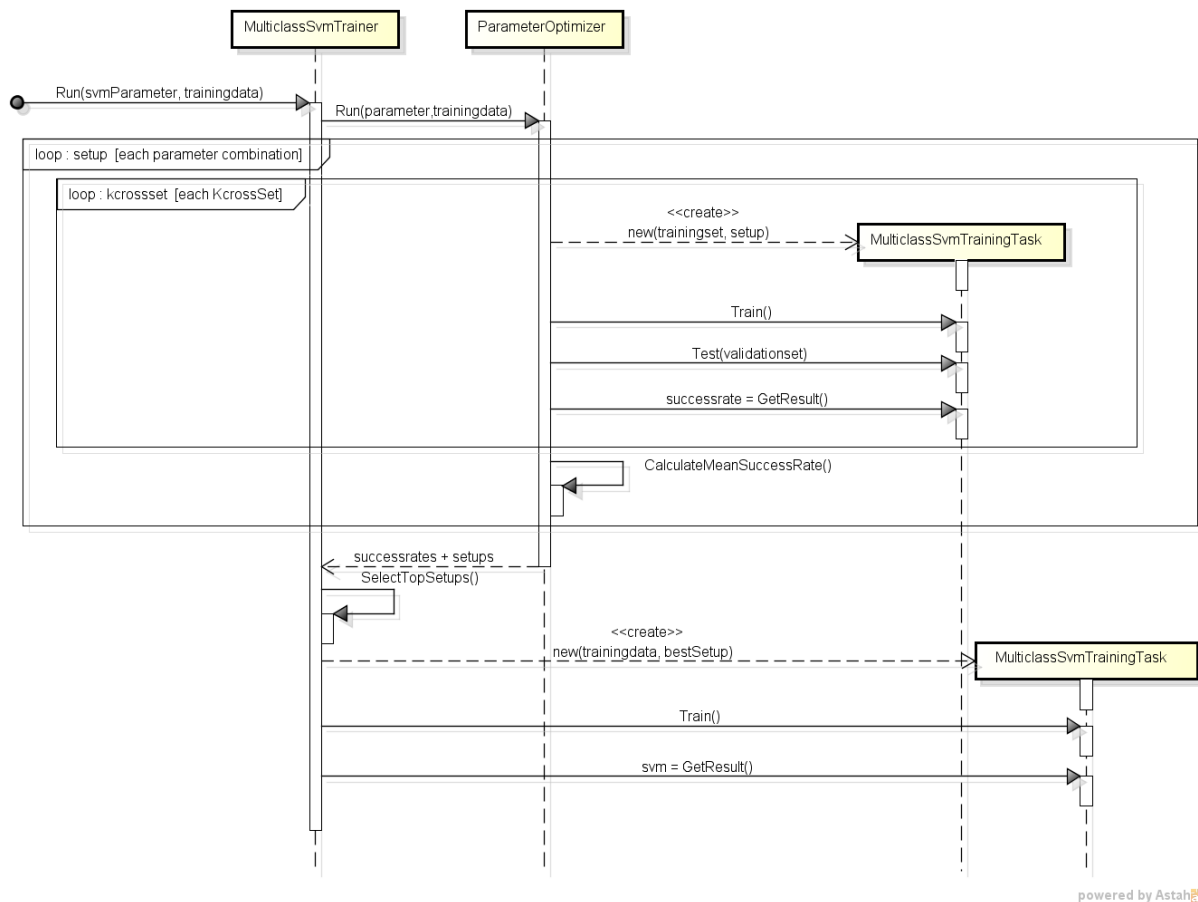


Abbildung 7.2: Ablauf: Trainingsvorgang

### Ablaufbeschreibung

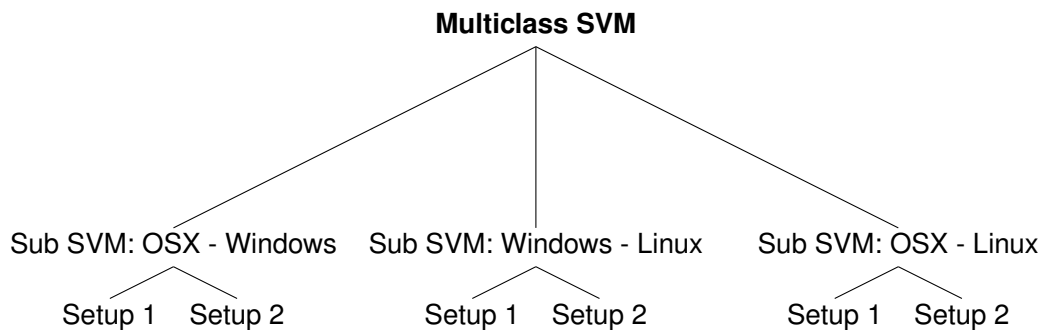
1. Ein Trainingsvorgang wird gestartet. Mitgegeben werden die Trainingsdaten und die Parameter-einstellungen
  - a) Aus den Parametern wird eine Kombination erstellt, mit der getestet werden soll.
    - i. Aus den Trainingsdaten wird ein K-Fold Cross Validation Set erstellt, das die Daten in einen Teil für das effektive Training und einen Teil für die Testphase unterteilt.
    - ii. Der Klassifizierer wird trainiert und mit den Testdaten validiert. Als Resultat steht die Erfolgsrate der Validation zur Verfügung.
    - iii. Wenn noch k-Cross Sets möglich sind, wird mit Punkt 1(a)i fortgefahren
  - b) Anhand der Anzahl Durchgänge und den Testresultaten wird die mittlere Erfolgsrate berechnet
  - c) Wenn noch Parameter Kombinationen möglich sind, wird mit Punkt 1a fortgefahren
2. Sind die Trainings abgeschlossen werden die verschiedenen Setups, die getestet wurden, sowie deren mittlere Erfolgsraten zurückgegeben.

3. Anhand der erhaltenen Erfolgsraten wird nun jene mit dem höchsten Wert ausgewählt und für ein weiteres Training verwendet. Dieses Training beinhaltet der gesamte Umfang an Trainingsdaten.
4. Die somit ermittelte SVM wird zurückgegeben

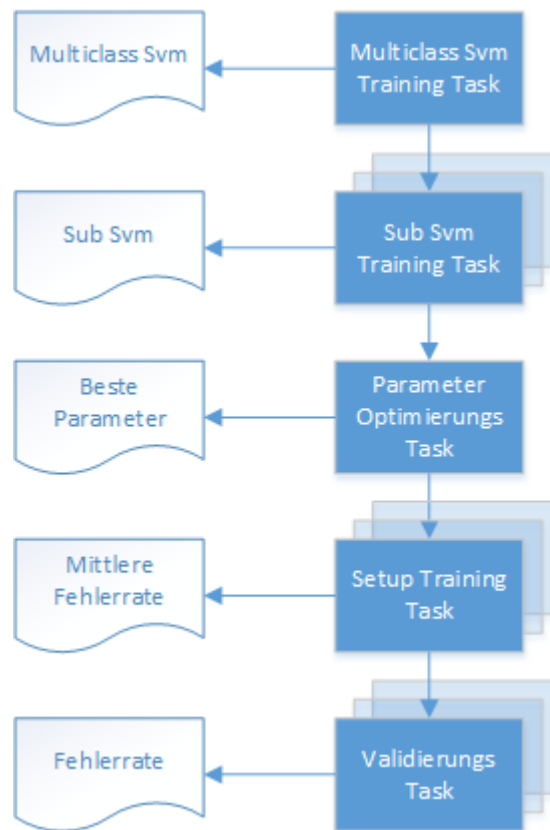
### 7.3.2 Neuimplementation des Optimierungsvorgang

Die vorherige Parameteroptimierung ist nicht optimal umgesetzt. Denn statt jedes einzelne Subproblem zu optimieren, wird stattdessen die Optimierung an der gesamten Multiclass SVM vorgenommen. Das heisst, es wird versucht, das ermittelte Setup auf alle Maschinen anzuwenden. Besser wäre es, wenn für jede Maschine ein individuelles Setup gefunden werden würde.

Der folgende Baum zeigt ein Mehrklassenproblem zwischen den Klassen "Windows", "OSX" und "Linux". Für jede Maschine werden zwei Setups trainiert und anschliessend das beste der beiden für die Sub SVM verwendet.



Die folgende Grafik illustriert der ungefähre Ablauf, die Entscheidungsträger sowie deren Resultate:



**Abbildung 7.3:** Ablauf: Trainingsvorgang

Der Grund, warum der vorherige Ansatz falsch war, ist daran ersichtlich, dass ein Mehrklassenproblem im Grunde kein eigenständiges Problem ist. Stattdessen ist es eine Ansammlung von binären Problemen, die zusammen die Klassifizierung vornehmen können. Deshalb ist es folglich zwingend, auf die Optimierung der Submaschinen einzugehen. Da ausserdem jedes Subproblem individuelle Eigenschaften hat, läuft man bei einem gesamtheitlichen Setup der Gefahr auf, dass die Parameterwerte zu nicht Konvergierung der Lösung führen können. Dies könnte das Training der gesamten Maschine zum Fehlschlagen bringen.

### 7.3.3 Domain Modell

Im folgenden Domain Modell ist das eigentliche Training als Klassendiagramm zusammengefasst.

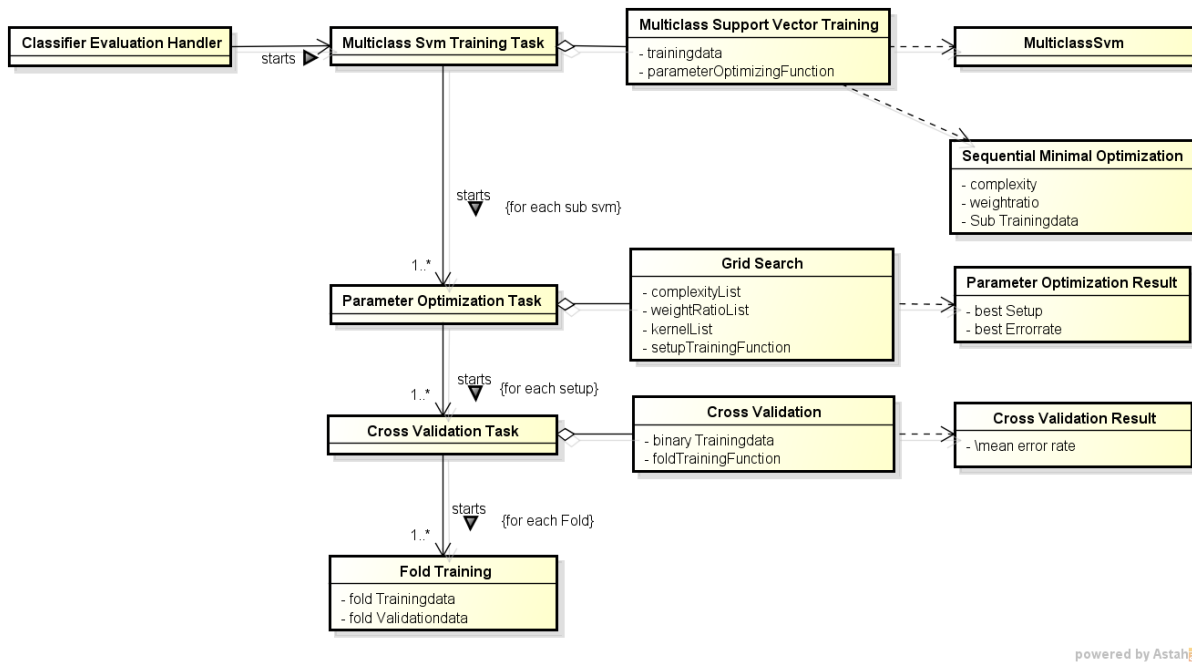


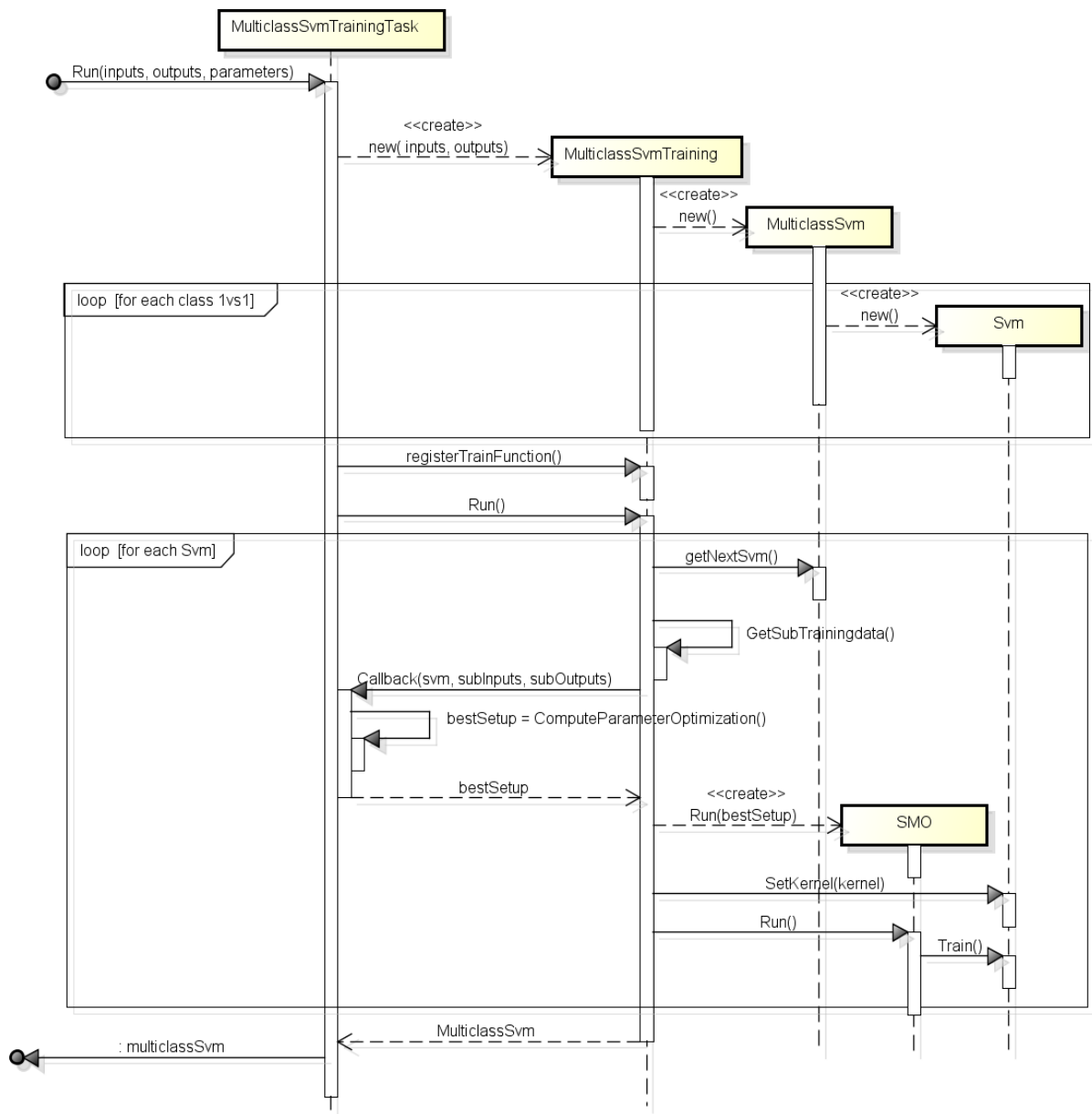
Abbildung 7.4: Domainmodell: Trainingsvorgang

Klasse	Beschreibung
Classifier Evaluation Handler	Der ClassifierEvaluationHandler ist der Einstiegspunkt für den Trainingsvorgang. Über die Mitgabe eines Konfigurationsobjektes sowie der Trainingsdaten führt er einen Trainings Vorgang aus und gibt die trainierte Multiclass SVM zurück
Multiclass SVM Training Task	Der SVM Training Task steht für einen Trainingsvorgang, bei dem eine Multiclass SVM ermittelt wird. Er registriert sich beim Multiclass Support Vector Training und startet für jede Sub SVM einer Multiclass SVM ein Parameteroptimierungsvorgang
Parameter Optimization Task	Der Parameteroptimierungs Task steht für den Vorgang, bei dem die besten Parameter für die Multiclass SVM ermittelt wird. Dafür registriert sich bei einem Grid Search Objekt und startet für jedes Setup eine K-Fold Cross Validation
Cross Validation Task	Der Cross Validation Task ist verantwortlich für die Validierung eines Setups. Dafür registriert sich bei einem Cross Validation Objekt. Bei jedem Fold der K-Fold Cross Validation startet er ein Training einer unabhängigen SVM mit dem festgelegten Setup und berechnet deren Erfolgsrate
Fold Training Task	Dieses Task Objekt führt ein Training einer Support Vector Maschine aus. Diejenige wird anschliessend mit den Validationsdaten des Folds getestet. Daraus wird die Erfolgsrate des Setups ermittelt
Multiclass Support Vector Machine Learning	Das IMulticlass SVM Training Objekt steht für das Training einer Multiclass SVM. Dazu führt es für jede Sub SVM ein Parameteroptimierungsprozess durch und trainiert anschliessend die Sub SVM mit dem besten Setup
Multiclass Svm	Das MulticlassSupportVectorMachine Objekt steht für eine Multiclass SVM, die es zu trainieren gilt

Sequential Minimal Optimization	Das SequentialMinimalOptimization Objekt steht für den Vorgang des effektiven Trainings der Sub SVM. Dabei wird der Algorithmus zur Lösung des Optimierungsproblems angewendet. Das Training wird durch die beiden Parameter <i>Complexity</i> und <i>WeightRatio</i> beeinflusst
Grid Search	Das IGridSearch Objekt steht für den Vorgang des GridSearch Algorithmus. Bei diesem wird aus einer Liste von Parametern alle Kombinationen erstellt und für jede einzelne ein Training ausgeführt. Eine Kombination wird hierbei Setup genannt. Zum Schluss wird das beste Setup zurückgegeben
Grid Search Result	Dieses Resultat hält alle Setups, die vom Grid Search Algorithmus ermittelt wurde sowie deren Fehlerraten
Cross Validation	Das CrossValidation Objekt steht für den K-Fold Cross Validation Vorgang. Dieser teilt ein Set von Trainingsdaten in $k$ Subsets auf. Anschliessend wird jedes Subset ein Mal für die Validation verwendet
Cross Validation Result	Diese Klasse hält das Resultat des K-Fold Cross Validation Vorgangs. Dazu gehört die mittlere Fehlerrate über alle $k$ Validierungen

### 7.3.3.1 Trainingsvorgang

Der Trainingsvorgang wurde überarbeitet und ist hier in ausführlicher Form beschrieben



powered by Astah

Abbildung 7.5: Ablauf: Trainingsvorgang Neuimplementation

### Ablaufbeschreibung

1. Der Trainingsvorgang wird gestartet
2. Es wird ein neues MulticlassSvm Training Objekt erstellt. Mitgegeben werden die Vektoren und die erwarteten Ausgabewerte
3. Es wird eine MulticlassSvm erstellt
4. Für jedes 1vs1 binäre Problem wird eine Sub Support Vektor Maschine erstellt
5. Es wird die Callbackfunktion registriert, die vor jedem Training einer Sub SVM aufgerufen wird
6. Der Lernprozess wird in Gang gesetzt

- a) Die Learning-Klasse holt sich eine Sub SVM und wählt für das Subtraining die Feature Vektoren und die Klassenzuweisung des entsprechenden binären Subproblems
  - b) Es wird die vorher definierte Callback-Methode aufgerufen.
  - c) In der Callback-Methode wird mithilfe der ComputeParameterOptimization Funktion das beste Parametersetup bestimmt.
  - d) Das Setup wird zurückgegeben.
  - e) Aus dem Setup wird der Kernel ermittelt. Dieser wird in der SVM gesetzt
  - f) Anhand des Setups und der Sub SVM wird ein SequentialMinimizingOptimization Objekt erstellt. Diese soll das quadratische Optimierungsproblem der Sub SVM lösen.
  - g) Der Optimierungsprozess wird gestartet. Dies führt das Training der Sub SVM aus
  - h) Konnte das Optimierungsproblem gelöst werden, ist der Trainingsvorgang einer einzelnen Sub SVM abgeschlossen
  - i) Solange noch weitere Subprobleme zu lösen sind, wird bei Punkt 6a fortgefahren
7. Sind alle Subprobleme erfolgreich gelöst worden. Steht nun eine trainierte Multiclass SVM zur Verfügung



## Optimierungsfunktion

Die vorhergehende Grafik beinhaltete die Optimierungsfunktion nur als einfacher Aufruf. Im folgenden Abschnitt wird auf deren Ablauf genauer eingegangen

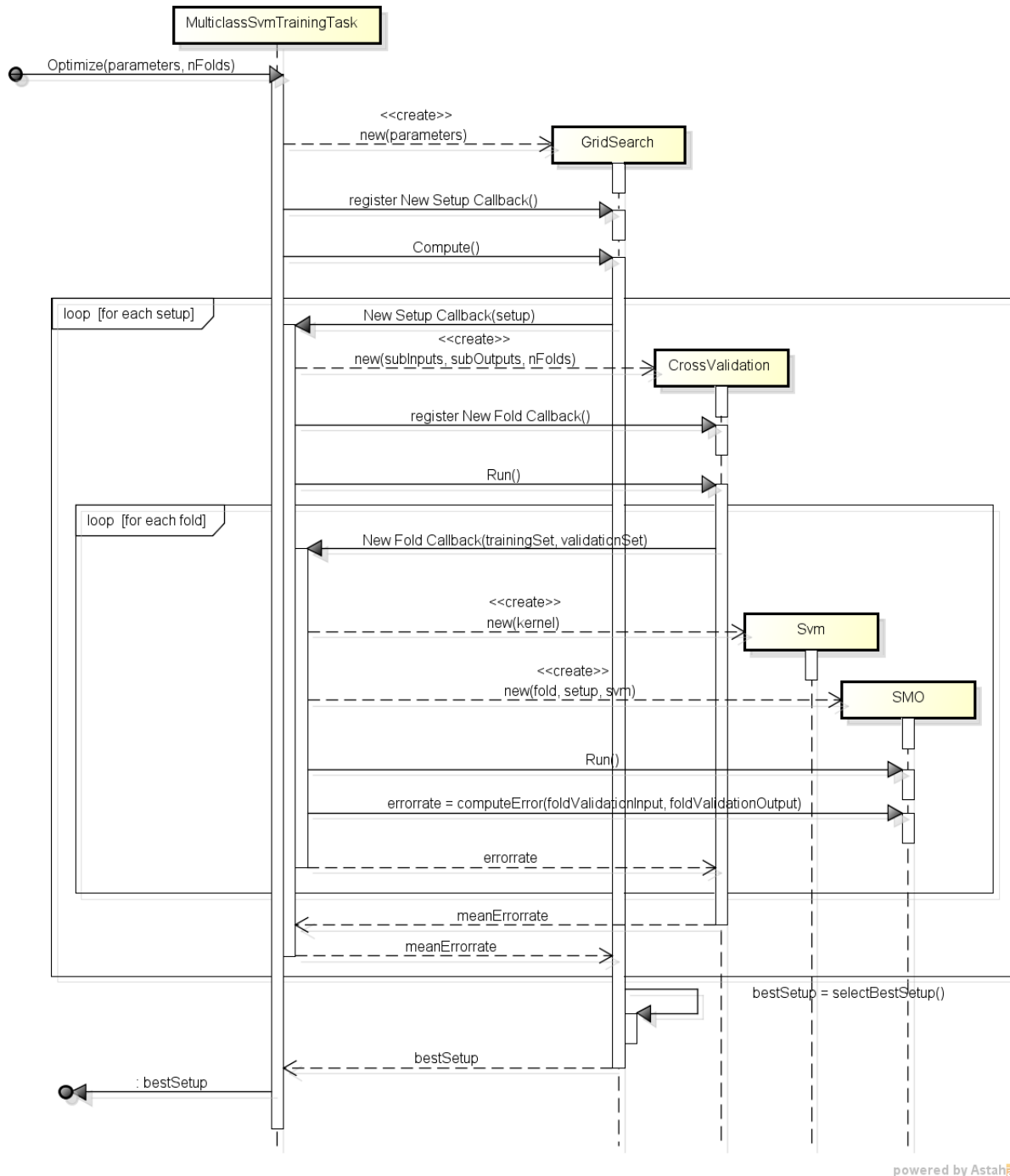


Abbildung 7.6: Ablauf: Parameteroptimierung

## Ablaufbeschreibung

1. Es wird die Funktion für die Parameteroptimierung aufgerufen. Übergeben werden die Parameterlisten, die Grösse der Cross Validierungssets und die Feature Vektoren des binären Subproblems
2. Es wird ein neues GridSearch-Objekt erstellt, welches aus den Parameterlisten alle Kombinationen erstellen kann. Eine Kombination wird Setup genannt.
3. Es wird eine Callback Methode registriert, welche vom GridSearch Objekt aufgerufen wird, sobald ein neues Setup zum Trainieren bereitgelegt wurde
4. Der GridSearch Algorithmus wird in Gang gesetzt
  - a) Die Callback Methode wird aufgerufen. Dazu wird das aktuelle Setup, sowie die Feature Vektoren mitgegeben
  - b) Mit den Sub Feature Vektoren wird ein neues CrossValidation Objekt erstellt. Dieses Objekt ist dafür verantwortlich, mehrere Sub Training- und Validierungssets zusammenzustellen.
  - c) Auch hier wird eine Callback Methode registriert. Diese wird aufgerufen, sobald mit einem neuen Set trainiert und getestet werden kann.
  - d) Die Cross Validierung wird gestartet.
    - i. Soll ein neues Set getestet werden, wird die Callbackmethode aufgerufen
    - ii. Für die Validierung des Setups wird eine neue SVM erstellt. Dieser wird der Kernel mitgegeben, der für das gesamte Training verwendet wird
    - iii. Für das Training der SVM wird ein SequentialMinimalOptimization Objekt erstellt. Dieses hat die Aufgabe, das Optimierungsproblem der SVM zu lösen. Übergeben wird das Trainingsset des aktuellen Cross Validierungsschritt, die SVM, sowie das Setup, das ausprobiert werden soll.
    - iv. Der Prozess für das Lösen des Optimierungsproblem wird gestartet
    - v. Ist die SVM trainiert worden, kann die Fehlerrate für den aktuellen Validierungsschritt berechnet werden. Dazu wird das Testset verwendet
    - vi. Solange nicht alle Validierungsschritte durchgeführt wurde, wird bei Punkt 4(d)i fortgefahren
  - e) Nachdem alle Validierungsschritte abgeschlossen sind, kann die mittlere Fehlerrate bestimmt und zurückgegeben werden
  - f) Solange nicht alle Setups ausprobiert wurden, wird bei Punkt 4a fortgefahren
5. Sind alle Setups durchprobiert worden, kann anhand deren mittleren Fehlerraten bestimmt werden, welches das beste Setup gewesen ist. Dieses wird zurückgegeben und der Optimierungsvorgang ist somit abgeschlossen.

## Bemerkungen zum neuen Vorgang

**Subtasks** Der Gridsearch- und der CrossValidation-Algorithmus werden nicht vom MulticlassSvm-TrainingTask aufgerufen. Stattdessen werden die Subtasks GridSearchTask und CrossValidationTask erstellt, die sich bei den Algorithmen auf die Funktionen registrieren und die die Erfolgsraten berechnen.

**Factory Methoden** Es ist ersichtlich, dass während dem Trainingsvorgang mehrere kurzlebige Instanzen von Objekten erstellt werden müssen. Die Instanziierung wird im Code von Factories durchgeführt, aber aus Gründen der Übersichtlichkeit nicht in den Diagrammen dargestellt.

Der Grund für die Verwendung von Factories ist der, dass die Objekte mit spezifischen Parametern erstellt werden müssen, die individuell für den Trainingsvorgang sind. Weiterhin wäre es auch möglich gewesen, auf Factories ganz zu verzichten und stattdessen die Klassen, die das jeweilige Objekt benötigen, als Ersteller zu bestimmen. Dies hätte aber dazu geführt, dass das System nur sehr schwer über Mocks simulierbar gewesen wäre.

**Interfaces** Die in der Darstellung verwendeten Klassen wie GridSearch, CrossValidation und SequentialMinimalOptimization werden von der Accord Library zur Verfügung gestellt. Da diese jedoch weder durch Interfaces gekapselt, noch virtuelle Methoden besitzen, sind sie in der Implementation durch eigene Wrapperklassen ersetzt worden.

Auch hier liegt der Grund darin, dass es auf diese Weise möglich ist, die Klassen mit der Moq Library zu simulieren. Weiterhin kann so erreicht werden, dass die Library nur an einem Ort eingebunden werden muss und somit auch andere Implementationen möglich sind.

Natürlich wäre es auch möglich gewesen, eine andere Mocking Library zu verwenden. So würde es beispielsweise die Library Typemock Isolator ermöglichen, auch nicht virtuelle und gar statische Methoden zu imitieren. Da diese Lösung jedoch nur käuflich erwerblich ist und das Kosten-Nutzen Verhältnis nicht stimmt, wurde darauf verzichtet.

**Callback Methoden** Wie in der Darstellung ersichtlich, wird in vielen Fällen mit Callback Methoden gearbeitet. In diesem Sinne bietet sie in den einzelnen Klassen Properties an, denen ein Delegate für den Callback angegeben werden kann. Dies bietet den Vorteil, dass der gesamte Ablauf des Trainingsvorgangs an einem Ort konfiguriert werden kann. Leider wird dadurch der Code erheblich unübersichtlicher und deshalb auch schwieriger zum Testen. Deshalb wurde die Ausführung der verschiedenen Verarbeitungsschritte Subtasks überlassen.

**Parallelität** Die Accord Library ist auf hohe Parallelität ausgelegt, was gerade auf Maschinen mit vielen CPUs für erheblichen Geschwindigkeitsschub sorgt. Die parallelen Arbeiten werden als voneinander unabhängige Tasks auf dem Threadpool der System-Library ausgeführt. Das bedeutet, dass der Vorgang auf schwachen wie auf stark parallelen Systemen gut skaliert.

Als Beispiel wird eine Multiclass SVM mit 4 Klassen (6 Maschinen), 2 \* 3 Parametern (9 Setups) und eine K-Cross Validierung Setgröße von 10 genommen

Eigenschaft	Anzahl	Anz. Tasks
Maschinen	6	6
Setups	9	9
K-Cross Schritte	10	10
Total		$6 * 9 * 10 = 540$

**Cache Grösse** Die gesamte Speichernutzung nimmt mit der starken Parallelisierung um einen beträchtlichen Faktor zu. Dies ist auf den Cache der vielen Sequential Minimal Optimization Objekten zurückzuführen. In Summe wird bei jedem K-Cross Validierungsschritt eine Instanz dessen erstellt. Das bedeutet für das oben erwähnte Beispiel, dass im schlimmsten Falls bis zu 540 Instanzen gleichzeitig existieren können. Bei einem Cache von 200 ( $200^2 * 8 = 320kB$ ) würde das eine Gesamtspeicherausnutzung von bereits 170 MByte bewirken. Nicht betrachtet sind hierbei die Feature Vektoren.

## 7.4 Kategorisierungsvorgang

Die Kategorisierung von eingehenden Nachrichten wird vom Categorization Handler durchgeführt. Dazu nimmt er die Nachricht entgegen und führt den Klassifizierungsvorgang aus. Somit ist er die eigentliche Schnittstelle zur E-Mail Inbox.

Soll ein neues E-Mail kategorisiert werden, wird es der Inbox in Form einer Request übergeben. Darin enthalten ist die Nachricht, mit dessen Textinhalt und einer eindeutigen Nachrichten Id. Die Inbox steuert anschliessend ihren vorkonfigurierten Classification Handler an und übergibt ihm die Nachricht.

Vom Categorization Handler wird daraufhin ein Klassifizierungsvorgang gestartet und eine Zuweisung ermittelt. Dazu werden die Antworten des adaptiven Klassifizierers und des Language Identifiers in einer Response zusammengefasst. Diese kann anschliessend von der E-Mail Inbox entgegengenommen werden und für weitere Zwecke, wie das Verschieben oder das Beschriften des Mails, verwendet werden.

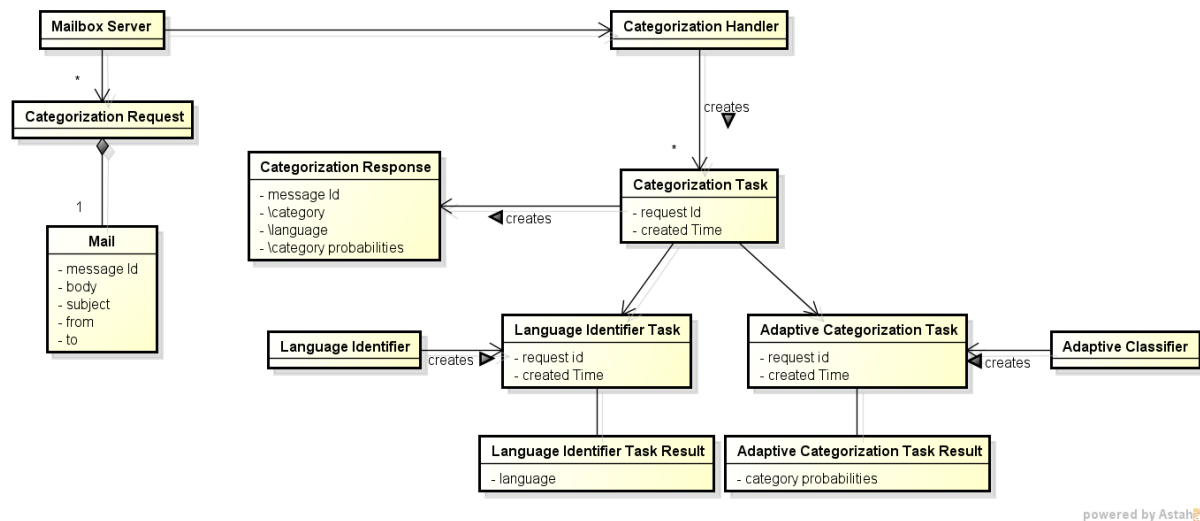
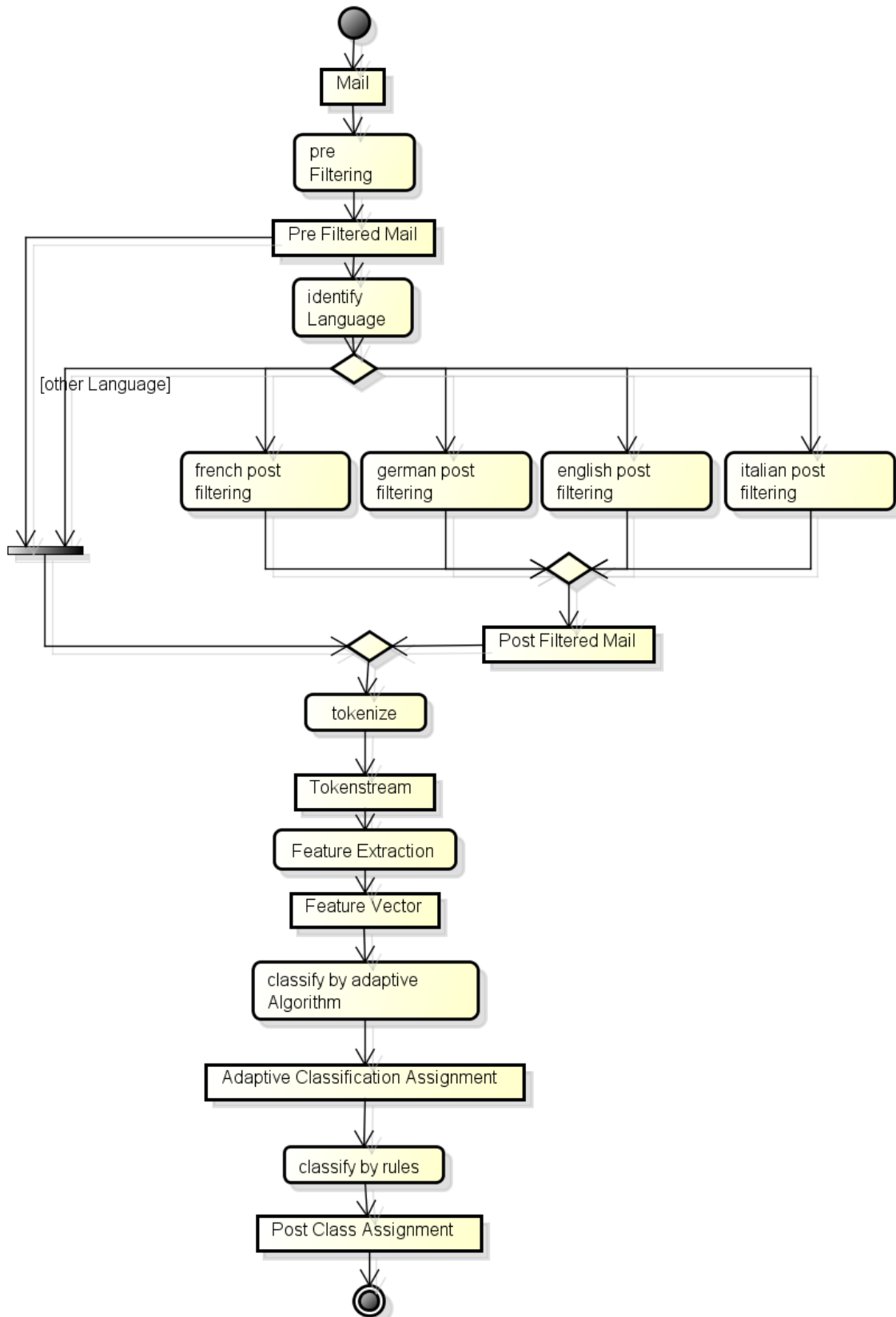


Abbildung 7.7: Modell: Klassifizierung

## 7.4.1 Ablauf



powered by Astah

Abbildung 7.8: Ablauf: Klassifizierungsvorgang Activity

## Ablaufbeschreibung

1. Es wird ein Mail erhalten. Das Mail besteht aus verschiedenen Feldern. Zu diesen gehören der Absender, Betreff, CC und der Inhalt.
2. Mails enthalten viel überflüssigen Overhead, der für die Klassifizierung nicht nötig ist. Dieser muss über eine Vorfiltrierung entfernt werden.
3. Anhand des gefilterten Textes kann nun entschieden werden, zu welcher Sprache der Text angehört.
4. Wurde eine Sprache festgelegt, kann ein sprachbezogenes Filtering betrieben werden. So werden beispielsweise Stopp-Wörter entfernt und ein Wordstemming angewendet.
5. Der Betreff und der Text wird nun in die einzelnen Sprachelementen (Tokens) aufgetrennt. Dieser Schritt wird Tokenization genannt. Anschliessend steht ein Stream der Tokens zur Verfügung.
6. Die Tokens müssen nun in einen Vektor überführt werden. Dazu wird eine Feature Selection durchgeführt.
7. Nun findet die primäre Klassifizierung statt. Der Feature Vektor wird hierzu in einen Vektorraum platziert. Anhand dessen Position wird folglich entschieden, zu welcher Kategorie er zugehören könnte.
8. Da die adaptive Klassifizierung keine 100 prozentige Aussage über die Klassenzugehörigkeit eines Textes machen kann, wird stattdessen ein Score berechnet und zurückgegeben
9. Anhand des Vorschlags des adaptiven Klassifizierer kann nun entschieden werden, zu welcher Kategorie das Mail hinzugefügt werden soll. Wenn die Scores der einzelnen Kategorien jedoch zu nahe beieinander liegen kann auch entschieden werden, das Mail einer vordefinierten Default-Category hinzuzufügen.
10. Die endgültige Klassenzuteilung wird zum Schluss zurückgegeben

### 7.4.2 Categorization Handler Chaining

Mit diesem Konzept wird es ermöglicht, verschiedene Klassifizierer aneinanderzureihen, um so ein Baum mit Subkategorien erstellen zu können.

Dieses Konzept bietet mehrere Vorteile:

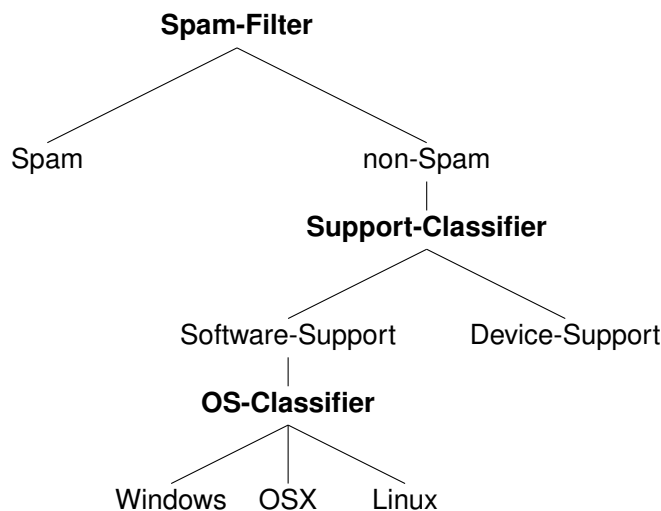
- Die Komplexität eines einzelnen adaptiven Klassifizierer wird verringert, da weniger Kategorien unterschieden werden müssen.
- Beim Hinzufügen von neuen Trainingsdaten zu einer Kategorie, muss nur der adaptive Klassifizierer, der die Kategorie kennt, neu trainiert werden. Dies beschleunigt den Trainingsvorgang.
- Es wird eine feingranularere Unterteilung in Subkategorien ermöglicht.

Grundsätzlich könnte auch ein gesamter Klassifizierer mit allen Kategorien und Subkategorien trainiert werden. Dies birgt aber die Gefahr, dass der Trainingsvorgang sehr lange dauert, da die Anzahl Features und die zu trainierenden Maschinen proportional zu den Kategorien steigen.

### 7.4.2.1 Beispiel

Eine Firma möchte gerne E-Mails des öffentlichen Mail-Kanals kategorisieren. Da sie ständig mit Spam zu kämpfen hat, möchte sie in erster Instanz Spam- und nicht-Spam voneinander trennen. Dazu trainiert sie einen Klassifizierer mit zwei Kategorien "Spam " und "non-Spam". In einem weiteren Schritt, möchte sie die nicht-Spam-Mails in die gewünschten Kategorien "Software-Support" und "Device-Support" unterteilen. Bei der Software wird zusätzlich die Unterscheidung zwischen Windows-, Linux- und OSX-Betriebssystem gemacht.

Mit diesen Anforderungen sieht der Baum folgendermassen aus:



Die fett geschriebenen Knotenpunkte stellen jeweils die Klassifizierer dar. Die übrigen Punkte sind die Kategorien.

Beim Klassifizierungsvorgang wird nun das Mail in erster Phase vom Spam-Filter analysiert. Wird das Mail der Kategorie Spam zugeteilt, ist die Klassifizierung abgeschlossen. Anders ist es bei der Kategorie "non-Spam". Dann wird nämlich der Support-Classier hinzugezogen, um dessen Unterkategorisierung vorzunehmen. Es wird so weitergefahren, bis ein Blattknoten erreicht wurde.

### 7.4.2.2 Ablauf

Zur Klassifizierung eines E-Mails müssen verschiedene Prozesse durchlaufen werden. Dazu gehört das Pre Processing, dass das E-Mail in eine Vektor umwandelt und so vom adaptiven Klassifizierer kategorisiert werden kann.



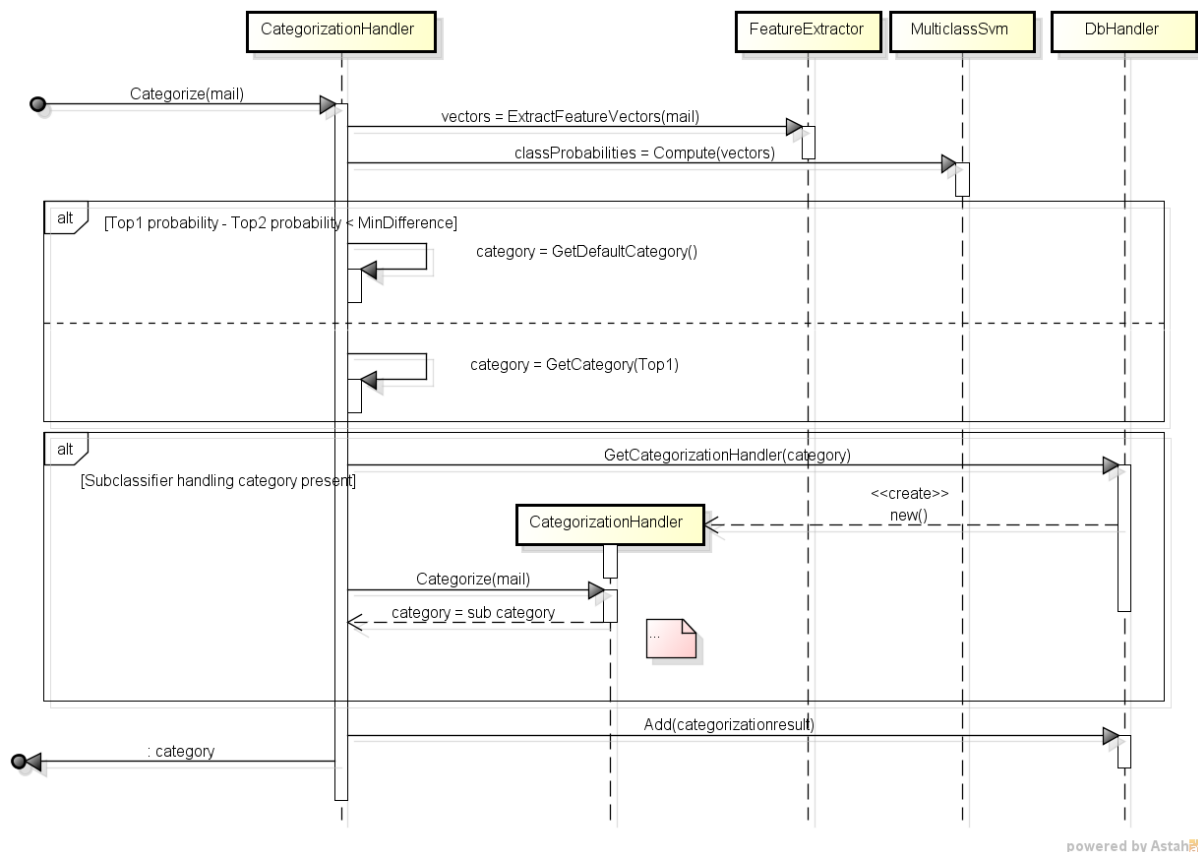
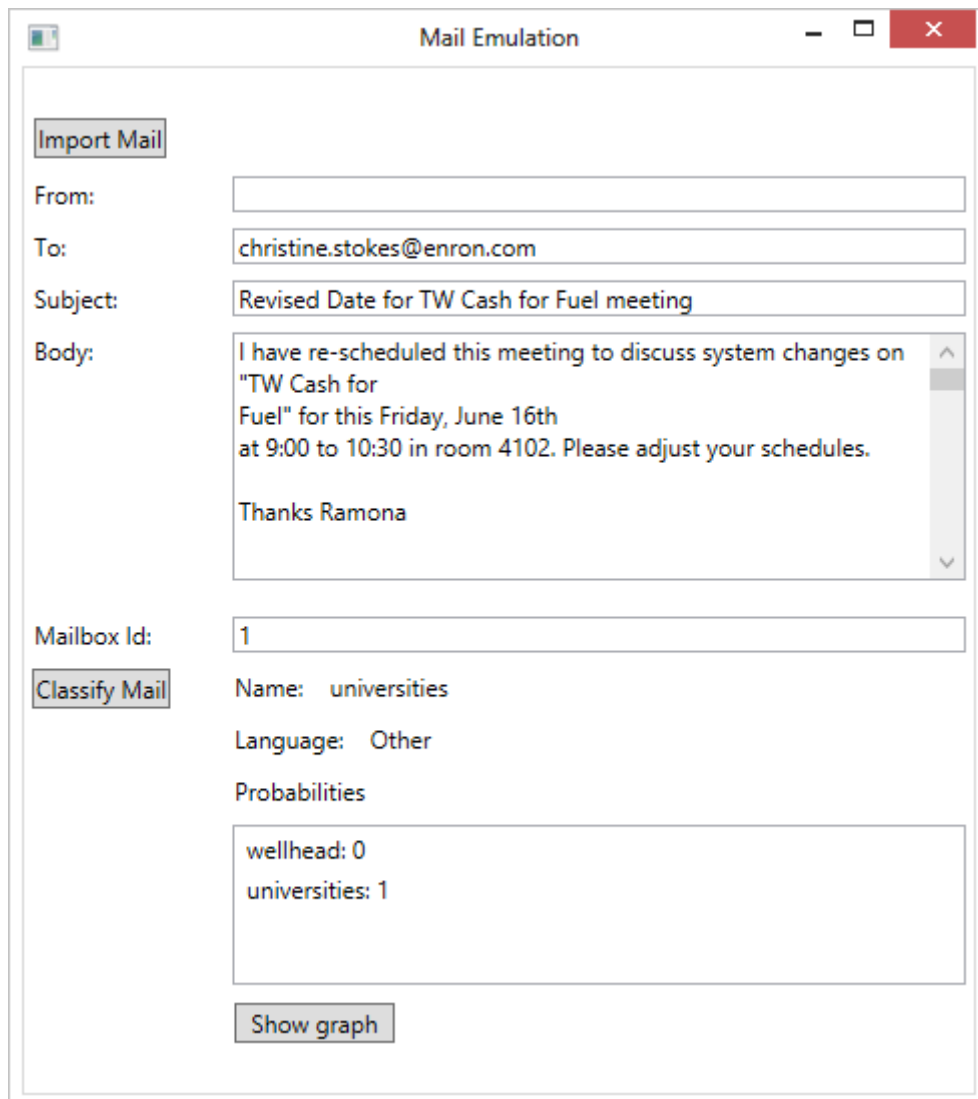


Abbildung 7.9: Ablauf: Klassifizierungsvorgang mit Chaining

### Ablaufbeschreibung

1. Es wird ein Klassifikationsvorgang gestartet
2. Das erhaltene Mail wird mithilfe des FeatureExtractors in einen Feature Vektor gewandelt
3. Der Feature Vektor wird dem adaptiven Klassifizierer übergeben, der für jede Klasse die Zugehörigkeitswahrscheinlichkeit berechnet
4. Die beiden grössten Wahrscheinlichkeiten werden miteinander verglichen
  - a) Ist der Unterschied kleiner als die vorgegebene Minimaldifferenz, wird die eingestellte Default Kategorie verwendet
  - b) Ist der Unterschied grösser oder gleich der vorgegebenen Minimaldifferenz, wird die Kategorie mit der höchsten Wahrscheinlichkeit verwendet
5. Es wird überprüft, ob Subklassifizierer existieren, die die ermittelte Kategorie behandeln
  - a) Ist ein Subklassifizierer vorhanden, wird der komplette Vorgang ein weiteres Mal durchlaufen
6. Die endgültige Zuweisung wird in die Datenbank geschrieben und zurückgegeben

### 7.4.3 Verwendung des Klassifizierers



**Abbildung 7.10:** Screenshot: Demo Applikation

Um den Klassifizierer verwenden zu können, wurde ein Demo Applikation entwickelt. Diese Demo Applikation besteht aus einem GUI, in dem eigene E-Mails erfasst oder ein bestehende E-Mails im MIME Format eingelesen werden können. Danach kann das zu verwendende Mailbox Objekt, an dem die ganze Konfiguration hängt, per ID angegeben werden. Der Klassifizierungsvorgang wird per Knopfdruck gestartet und das Resultat herausgegeben.

Zusätzlich kann das Resultat visuell als Graph dargestellt werden. Wurde ein Categorization Handler Chaining verwendet kann so die Entscheidung besser nachvollzogen werden.

Für die Anbindung an andere Applikationen wurde das Interface *ICategorizationHandlerService* entworfen. Das *ICategorizationHandlerService* Interface stellt eine Methode zur Verfügung, um die Kategorisierung vorzunehmen.

Der Ablauf, um eine Klassifizierung zu starten, ist wie folgt:

1. Das zu klassifizierende E-Mail wird in ein *MailDto* Objekt konvertiert werden.

2. Ein *CategorizationRequest* Objekt wird erstellt. Dies beinhaltet das *MailDto* Objekt sowie die Information welche Mailbox zur Klassifizierung verwendet werden soll.
3. Der *CategorizationRequest* wird dem *ICategorizationHandlerService* übergeben.
4. Der *ICategorizationHandlerService* liefert eine *CategorizationResponse* zurück.
5. Aus *CategorizationResponse* kann das Resultat der Klassifizierung entnommen werden.

Detailliertere Informationen zu den oben erwähnten Interfaces und DTOs kann im Kapitel 7.5.5 nachgelesen werden.

#### 7.4.4 Anbindung an ein Mailsystem

Im Rahmen der Bachelorarbeit wurde die Anbindung an ein Mailsystem nicht realisiert. Eine spätere Anbindung an ein Mailsystem wurde aber beim Erstellen der Architektur berücksichtigt.

Geplant ist, dass die Konfiguration, wie ein Mailsystem angesprochen werden soll, im *Mailbox* Objekt gespeichert wird. Auch Informationen, wie zum Beispiel in welchen Ordner ein klassifiziertes E-Mail verschoben werden soll, soll dort eingestellt werden können.

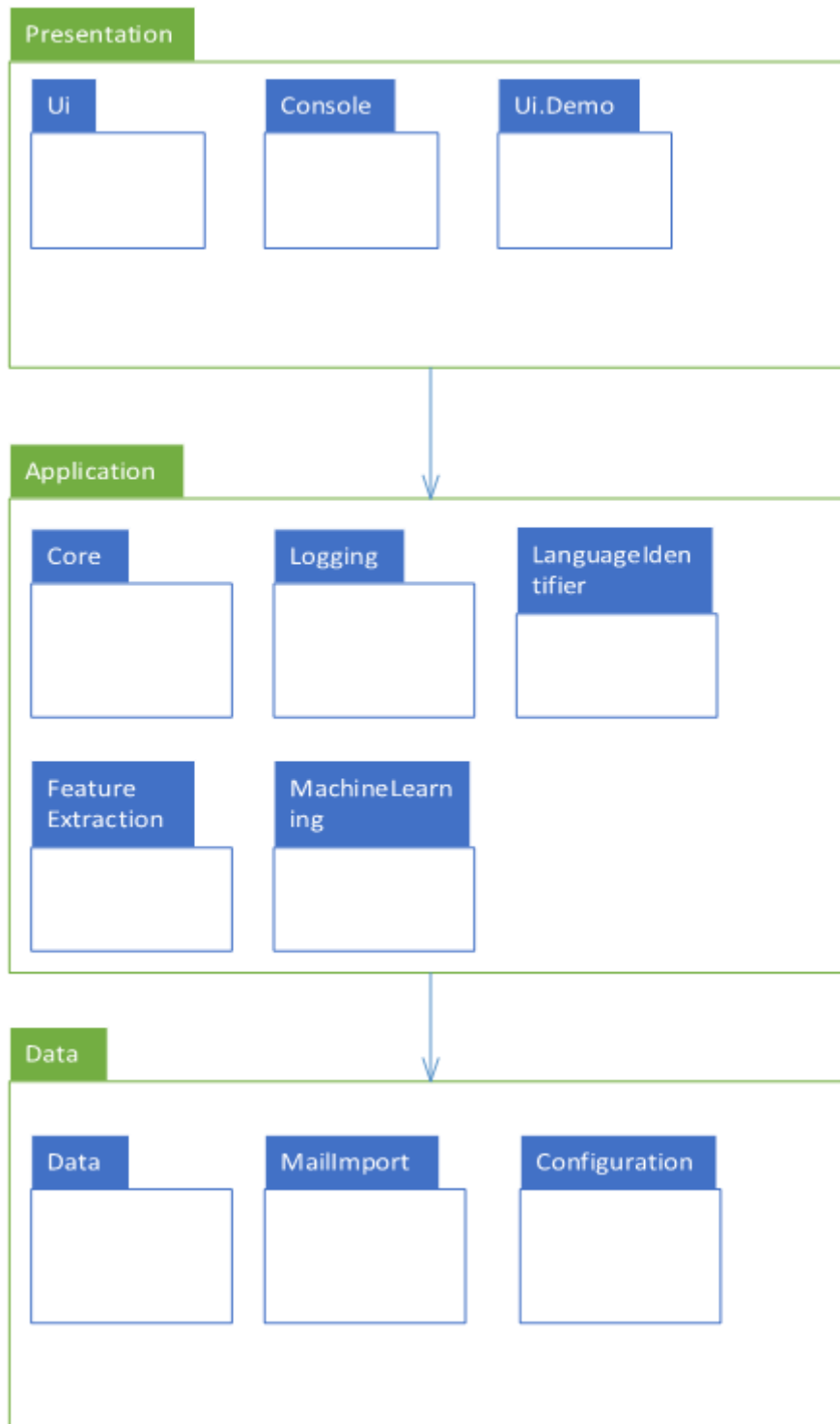
Wie die detaillierte Architektur für eine Mailanbindung aussehen wird, kann zu diesem Zeitpunkt noch nicht gesagt werden. Dies ist stark abhängig von den Schnittstellen, die das Mailsystem zur Verfügung stellt. Es kann sein, dass eine Multithreaded Architektur verwendet wird, so dass jede Mailbox einen eigenen Thread für die Klassifizierung besitzt. Andernfalls, wenn die Schnittstellen zum Mailsystem mittels eventbasierten Benachrichtigungen umgesetzt ist, könnte auch eine Singlethreaded Lösung in Betracht gezogen werden.

Um eine sinnvolle Aussage machen zu können, müssten die Schnittstellen des Mailsystems genauer analysiert und die Anforderungen abgeklärt werden.

## 7.5 Architektur

### 7.5.1 Layer Architektur

Die Architektur der Applikation lässt sich in drei Layer aufteilen: Presentation, Application und Data. Ziel des Layerings ist es eine höhere Trennung zwischen einzelnen Komponenten und somit eine höhere Wiederverwendbarkeit zu erreichen. Dabei darf es nur Abhängigkeiten von "höheren" Layern zu "tieferen" Layern geben.



**Abbildung 7.11:** Layerübersicht der Architektur

<b>Layer</b>	<b>Beschreibung</b>
Presentation	Der Presentation Layer beinhaltet das User Interface und Konsolen Applikationen (z.B: Performance Tests). Generell gesagt beinhaltet dieser Layer alles, was mit der Interaktion der Applikation zu tun hat.
Application	Der Application Layer beinhaltet die Kernlogik der Applikation.
Data	Der Data Layer regelt den Zugriff auf die Datenbank und auf Konfigurationsfiles die auf der Harddisk abgelegt sind.

## 7.5.2 Projektstruktur

Die gesamte Visual Studio Solution besteht aus insgesamt 18 Projekten. Vier davon sind Testprojekte, die anderen vierzehn enthalten produktiven Code. Die Solution wurde in so vielen Projekte aufgeteilt weil ein möglichst modularer Aufbau angestrebt wurde. Nachfolgend wird jedes Projekt kurz aufgelistet und beschrieben.

### 7.5.2.1 EmailClassifier.Configuration

Dieses Projekt beinhaltet die Logik für das Parsen des Trainingskonfigurationsfile. Dabei werden auch Falschwerte abgefangen. Wird irgendwo in der Solution die Trainingskonfiguration geladen geschieht dies über dieses Projekt.

### 7.5.2.2 EmailClassifier.Console

Beinhaltet einige Konsolenanwendungen, die für die Entwicklung verwendet wurden. Zum Beispiel handelt es sich auch um Beispiel-Code um das Verwenden einzelner Komponenten zu testen.

### 7.5.2.3 EmailClassifier.Core

Die Kernlogik der ganzen Applikation ist in diesem Projekt. Die Komponenten in diesem Projekt beinhalten den ganzen Trainings- und Klassifizierungs-Ablauf.

### 7.5.2.4 EmailClassifier.Data

Das *EmailClassifier.Data* Projekt regelt den Zugriff auf die Datenbank.

### 7.5.2.5 EmailClassifier.FeatureExtraction

Die Komponenten in diesem Projekt behandeln die Thematik der Feature Extraction. Dabei werden Tokens gebildet & und Feature Vektoren erstellt.

### 7.5.2.6 EmailClassifier.Languagelidentifier

Die Komponenten zur Spracherkennung befinden sich in diesem Projekt.

### 7.5.2.7 EmailClassifier.Logging

Dieses Projekt implementiert die Logging-Funktionalität. Das Projekt beinhaltet eine Konfigurations-XML-Datei. Übere diese Datei kann das Loggingverhalten konfiguriert werden.

### 7.5.2.8 EmailClassifier.MachineLearning

Beinhaltet die Implementierung der Support Vector Machine und dazugehörigen Klassen. Auch die Klassen für das konkrete Training befindet sich in diesem Projekt.

### 7.5.2.9 EmailClassifier.Ui

Das Konfigurations-UI wurde hier implementiert. Verwendet wurde dabei WPF.

### 7.5.2.10 EmailClassifier.Ui.Demo

In diesem Projekt wurde ein WPF-UI zu Demozwecken realisiert. Es kann bereits trainierte Klassifizierer verwenden und den Entscheidungsbaum darzustellen.

### 7.5.2.11 EmailClassifier.Util

Beinhaltet eine kleine Anzahl von Erweiterungsmethoden, die den .Net Alltag erleichtern.

### 7.5.2.12 EmailClassifier.MailImport

Dieses Projekt implementiert die Funktionalität Emails im MIME Format zu importieren. Die Importierten Mails können nachher weiter verwendet werden. Die Mails können auch als ZIP vorliegen.

### 7.5.2.13 EmailClassifier.Interfaces

Dieses Projekt besteht nur aus Intefaces und DTOs.

### 7.5.2.14 EmailClassifier.MachineLearning.Interfaces

Dieses Proejkt beinhaltet die Interfaces, die vom *EmailClassifier.MachineLearning*-Projekt verwendet werden. Um eine gute Übersicht zu bewahren wurde ein eigenes Projekt für diese Interfaces erstellt.

### 7.5.2.15 EmailClassifier.Configuration.Test

Beinhaltet Tests für das Projekt *EmailClassifier.Configuration*.

### 7.5.2.16 EmailClassifier.MachineLearning.Test

Beinhaltet Tests für das Projekt *EmailClassifier.MachineLearning*.



### 7.5.2.17 EmailClassifier.UnitTest

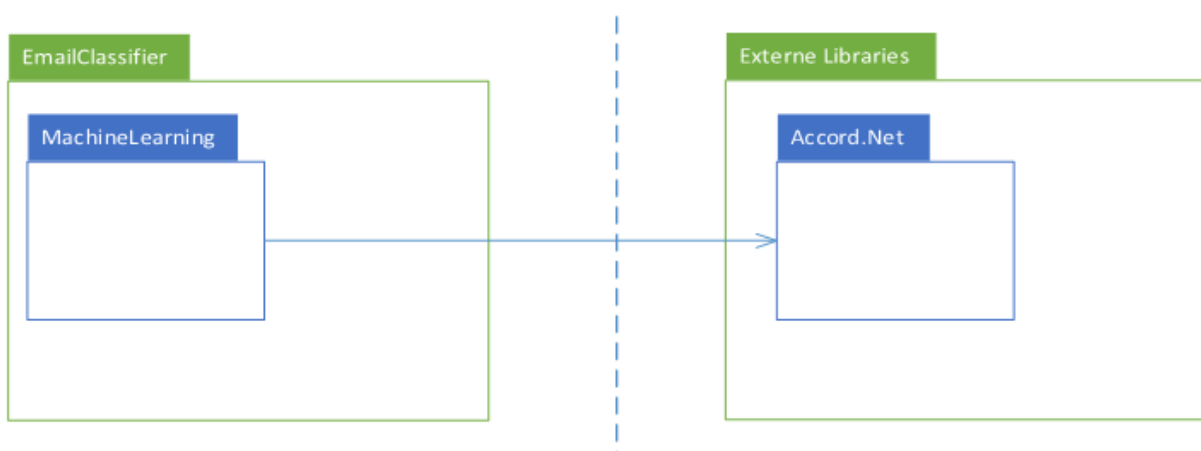
Dieses Projekt beinhaltet Unit Tests.

### 7.5.2.18 EmailClassifier.IntegrationTest

Dieses Projekt beinhaltet integrations Tests.

## 7.5.3 Externe Abhängigkeiten

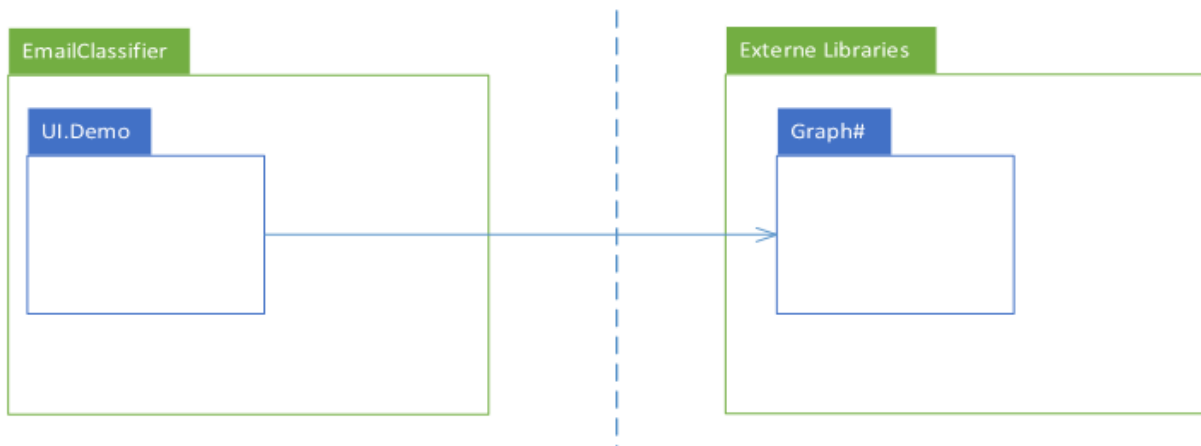
### 7.5.3.1 EmailClassifier.MachineLearning



**Abbildung 7.12:** Abhängigkeiten: EmailClassifier.Core

Das Projekt *EmailClassifier.MachineLearning* kapselt die Klassen der *Accord.Net* Library. Es implementiert Klassen für das Trainieren und das Optimieren von SVMs. Für diese Funktionalitäten werden sehr viele Algorithmen und Funktionen der *Accord.Net* Library verwendet.

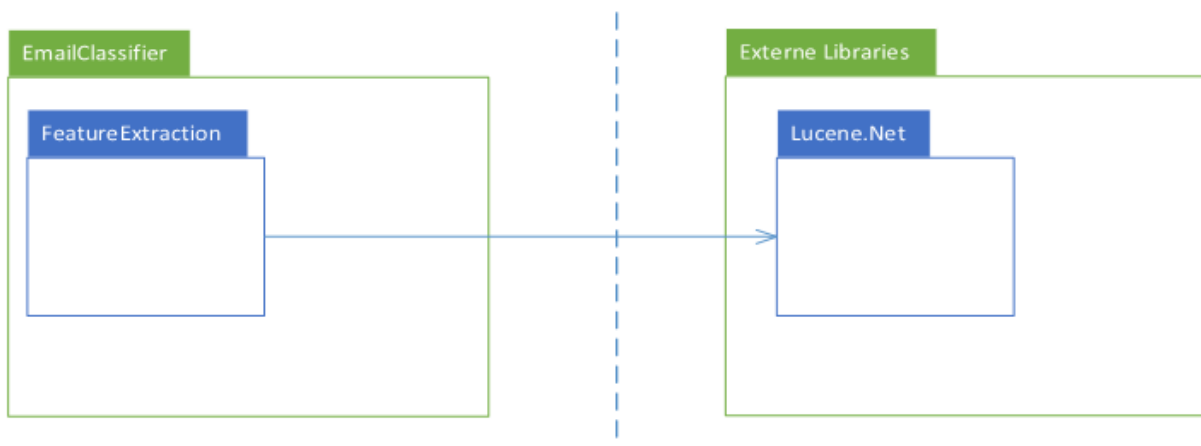
### 7.5.3.2 EmailClassifier.UI.Demo



**Abbildung 7.13:** Abhängigkeiten: EmailClassifier.Ui.Demo

Das Projekt *EmailClassifier.UI.Demo* verwendet die externe Library *Graph#* zur Visualisierung des CategorizationHandling Chaining. Das Resultat wird dabei als Baum dargestellt. Die *Graph#* Library stellt Datenstrukturen und ein WPF Element zur Verfügung.

### 7.5.3.3 EmailClassifier.FeatureExtraction



**Abbildung 7.14:** Abhängigkeiten: EmailClassifier.FeatureExtraction

Das *EmailClassifier.FeatureExtraction* Projekt verwendet die externe *Lucene.Net* Library. Die Library wird für das Tokenizing, das Wordstemming und zur Berechnung des TF/IDF Wertes verwendet.

### 7.5.3.4 EmailClassifier.Logging

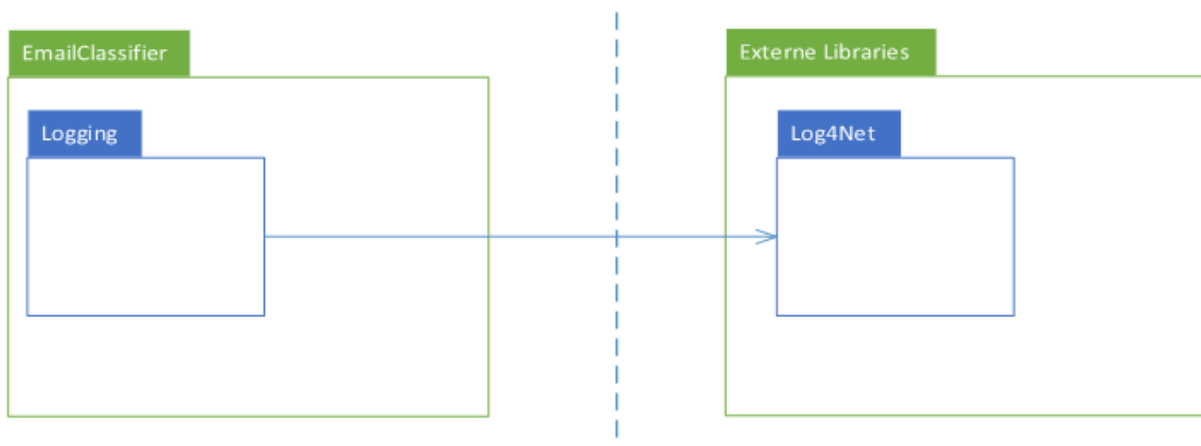


Abbildung 7.15: Abhängigkeiten: EmailClassifier.Logging

Das *EmailClassifier.Logging* Projekt verwendet die externe *Log4Net* Library. *EmailClassifier.Logging* implementiert die Logging-Funktionalität des Projektes. Das Projekt ist primär ein Wrapper für *Log4Net*, damit nicht alle Projekte von *Log4Net* abhängig sind. Somit wäre es auch erlaubt, andere Logging-Libraries zu verwenden.

### 7.5.3.5 EmailClassifier.MaillImport

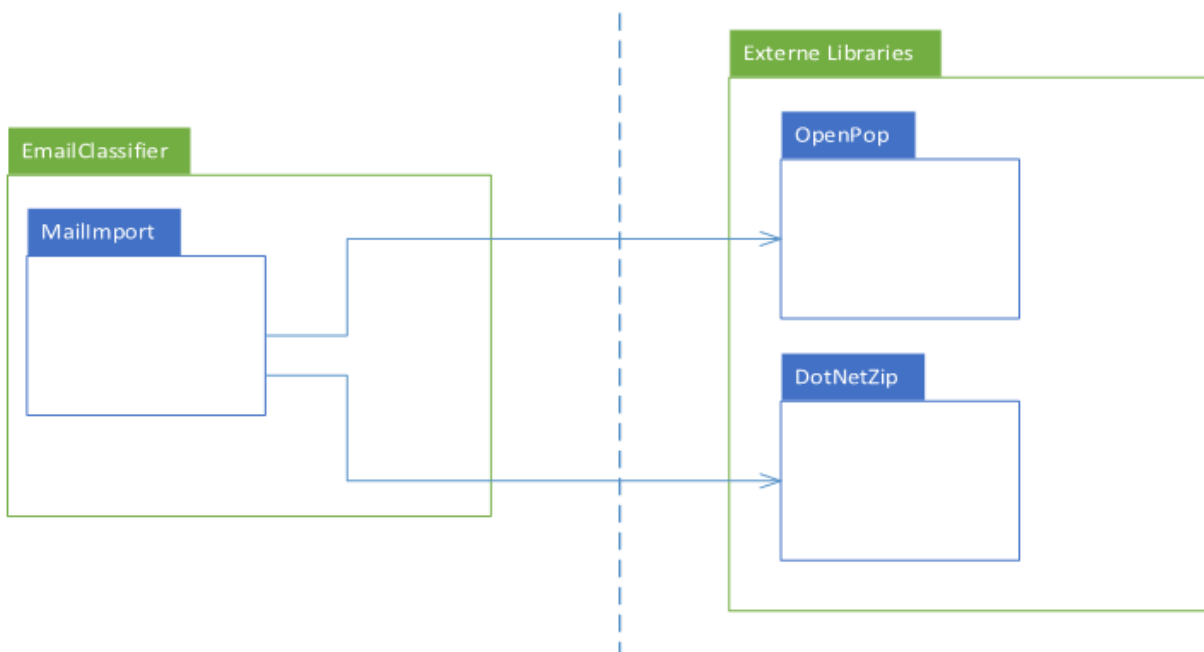


Abbildung 7.16: Abhängigkeiten: EmailClassifier.MaillImport

Das *EmailClassifier.MaillImport* Projekt verwendet die externen Libraries *DotNetZip* und *OpenPop.NET*.

Über dieses Projekt können Trainingsdaten effizient importiert und verwendet werden. Mehrere Trainingsdaten können im Zip Format vorliegen. Um Zip Dateien verarbeiten zu können, wird *DotNetZip* genutzt. Eine einzelne Trainingsdatei ist im MIME Format gespeichert. Zum parsen von MIME Dateien wird *OpenPop.NET* verwendet.

### 7.5.3.6 EmailClassifier.LanguageIdentifizier

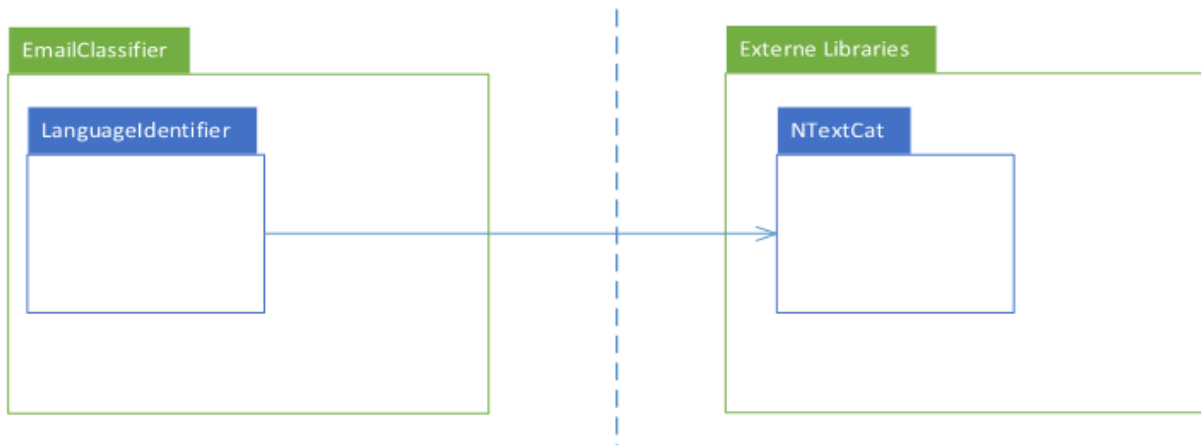


Abbildung 7.17: Abhängigkeiten: EmailClassifier.LanguageIdentifizier

Das *EmailClassifier.LanguageIdentifizier* Projekt verwendet die externe Library *NTextCat* für die Implementation der Spracherkennung. *NTextCat* liefert dabei Wörterbücher zu den unterstützten Sprachen mit.

### 7.5.3.7 EmailClassifier.UnitTest

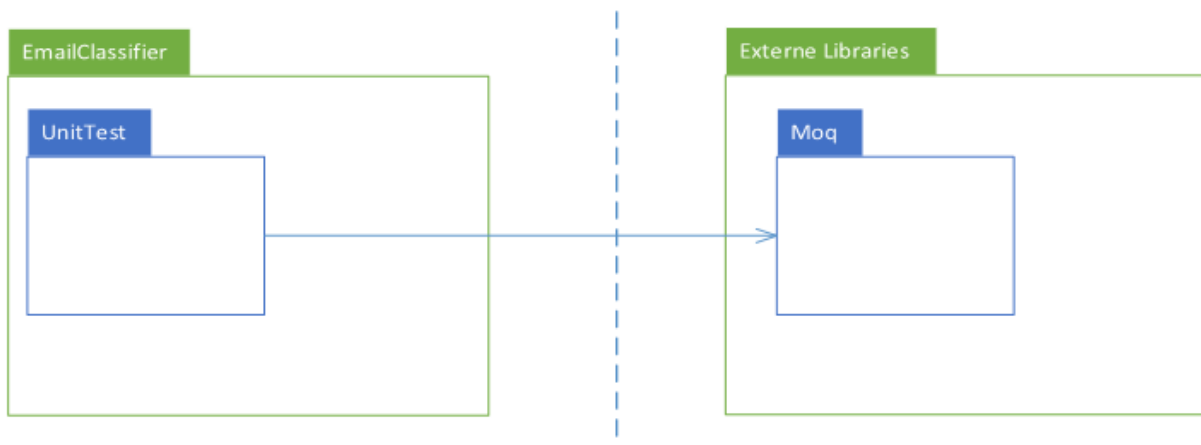


Abbildung 7.18: Abhängigkeiten: EmailClassifier.UnitTest

Das *EmailClassifier.UnitTest* Projekt beinhaltet alle Unit Tests. Für das Unittesting wird mit dem Mocking Framework *Moq* gearbeitet. *Moq* erlaubt es mit wenig Aufwand, Software Komponenten, die durch ein Interface repräsentiert werden, zu simulieren (mocken).

### 7.5.4 Datenmodell

Um bereits trainierte Klassifizierer, ihre Einstellungen, definierte Kategorien und deren Trainingsdaten und alle Entscheidungen beim Klassifizieren eines Mails persistent zu speichern, wird eine Datenbank verwendet. Dies hat den Zweck, dass nach einem Klassifizierungsvorgang nachvollzogen werden kann, wie der Entscheid zustande kam. Zudem bietet dies die Möglichkeit, die gefundenen Idealen Parameter eines Klassifizierers für ein erneutes Training wieder zu verwenden. In diesem Abschnitt wird das Datenmodell, die einzelnen Klassen und die Relationen genauer beschrieben.

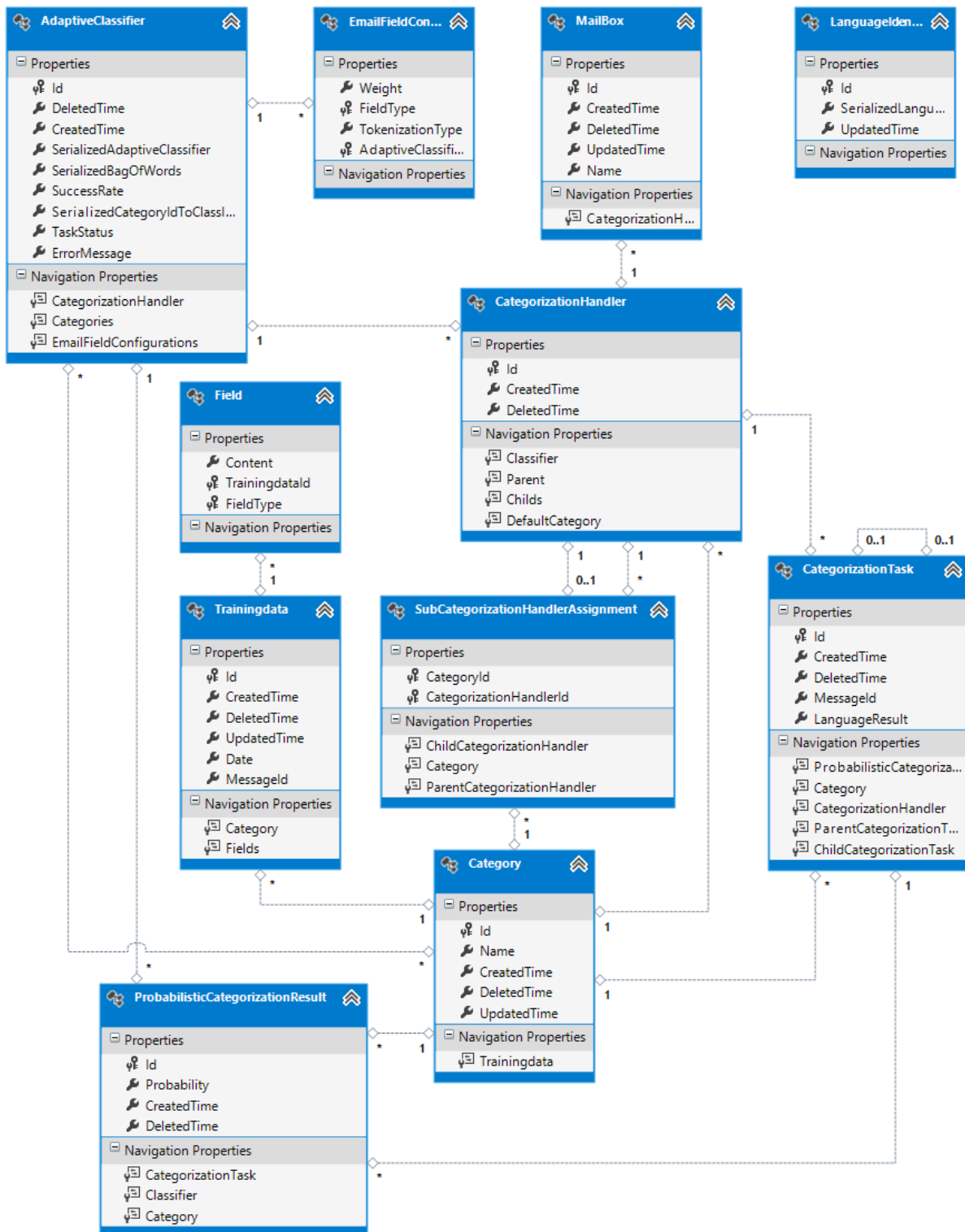


Abbildung 7.19: Übersicht des gesamten Datenmodells

### 7.5.4.1 Datenbankklassen

#### AdaptiveClassifier

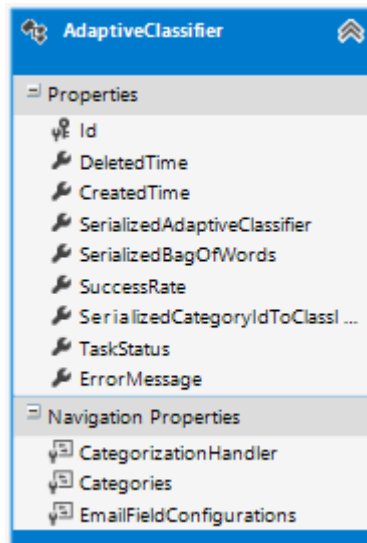


Abbildung 7.20: Datenbankklasse: AdaptiveClassifier

#### Zweck

Die AdaptiveClassifier Klasse repräsentiert einen trainierten Klassifizierer. Die Klasse beinhaltet einige serialisierte Objekte, die falls der Klassifizierer ein Mail klassifizieren muss, deserialisiert werden können.

#### Attribute

Attribute	Datentyp	Beschreibung
Id	Int32	Eindeutig identifizierbarer Primärschlüssel. Wird automatisch vom DBMS beim speichern erzeugt.
DeletedTime	DateTime	Timestamp der gesetzt wird, falls der Klassifizierer aus dem UI gelöscht wird.
CreatedTime	DateTime	Timestamp der gesetzt wird, sobald der Klassifizierer in die Datenbank geschrieben wird.
SerializedAdaptiveClassifier	Binary	Serialisierte Instanz einer Multiclass SVM. Das deserialisierte Objekt kann wiederverwendet werden, um Nachrichten zu klassifizieren
SerializedBagOfWords	Binary	Serialisierte Instanz des BagOfWords. Sie wird benötigt, um Feature Vektoren aus Texten generieren zu können.
Successrate	Double	Die voraussichtliche Erfolgsrate für zukünftige Klassifizierungen.
SerializedCategoryIdToClassIdMapping	Binary	Serialisierte Instanz einer Map, welche die Ausgaben der SVM auf die Kategorien abbildet.
TaskStatus	TaskStatus	Ein Enumwert, der über den Abschlussstatus des Trainings Auskunft gibt.
ErrorMessage	String	Falls der Trainingsvorgang des Klassifizierers fehlgeschlagen oder abgebrochen wurde, wird in diesem Attribut eine detailliertere Fehlermeldung gespeichert.

## Assoziationen

Property	Multiplizität	Beschreibung
CategorizationHandler	*	CategorizationHandler, welche diesen Classifier für die Klassifizierung verwenden.
Categories	*	Kategorien, die dem Klassifizierer bekannt sind. Dies sind immer mindestens zwei.
EmailFieldConfigurations	*	Konfiguration, welche Felder eines E-Mails relevant sind und wie diese zu gewichten sind.

## EmailFieldConfiguration

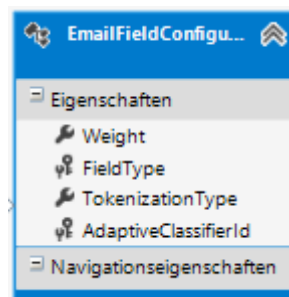


Abbildung 7.21: Datenbankklasse: EmailFieldConfiguration



## Zweck

Beschreibt, wie die Features eines E-Mail Feldes gewichtet werden sollen. Zudem wird festgelegt, wie die Tokengenerierung durchgeführt werden soll.

## Attribute

Attribute	Datentyp	Beschreibung
Weight	Double	Double Wert, der aussagt, wie stark ein entsprechendes E-Mail Feld gewichtet werden soll.
FieldType	FieldType	Enum, der festlegt, welches E-Mail Feld gewichtet werden soll. Teil des PKs.
TokenizationType	TokenizationType	Enum, der angibt, wie tokenisiert werden soll.
AdaptiveClassifierId	Int32	Teil des PKs und Ids des zugeordneten AdaptiveClassifiers.

## MailBox

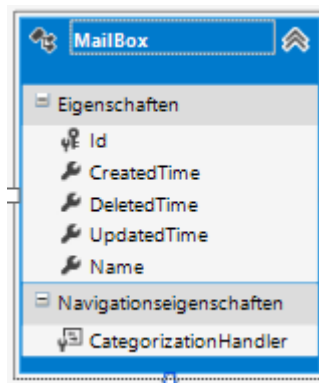


Abbildung 7.22: Datenbankklasse: MailBox

## Zweck

Eine Mailbox repräsentiert die Schnittstelle nach ausserhalb der Applikation. Ihr ist immer ein CategorizationHandler zugeordnet, der die automatisierte Klassifizierung der E-Mails vornimmt.

Wenn ein Klassifizierungsvorgang gestartet werden soll, muss die Mailbox Id bekannt sein.

## Attribute

Attribute	Datentyp	Beschreibung
Id	Int32	Primary Key, um die Mailbox eindeutig zu identifizieren. Wird automatisch gesetzt sobald die Mailbox in die Datenbank gespeichert wird.
CreatedTime	DateTime	Timestamp, der gesetzt wird, wenn die Mailbox der Datenbank hinzugefügt wird.
DeletedTime	DateTime	Timestamp, der gesetzt wird, wenn die Mailbox aus dem UI gelöscht wird.
UpdateTime	DateTime	Timestamp, der gesetzt wird, wenn die Mailbox verändert wird.
Name	String	Ein Name, der der Mailbox zugewiesen werden kann.

## Assoziationen

Property	Multiplizität	Beschreibung
CategorizationHandler	1	CategorizationHandler, der den ganzen Klassifizierungsvorgang koordiniert.

## CategorizationHandler

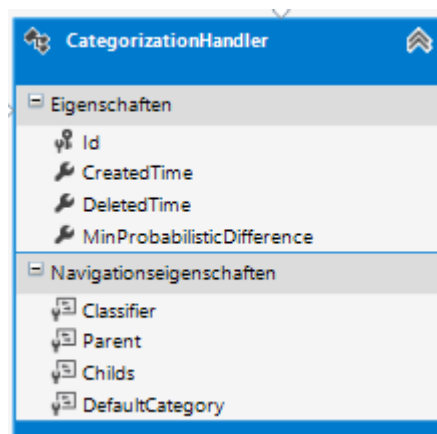


Abbildung 7.23: Datenbankklasse: CategorizationHandler

### Zweck

Der CategorizationHandler ist für den gesamten Klassifizierungsvorgang verantwortlich. Pro Kategorie, die sein AdaptiveClassifier kennt, kann ein ChildCategorizationHandler konfiguriert werden. Derjenige kann hinzugezogen werden, um eine verfeinerte Klassifizierung durchzuführen.

### Attribute

Attribute	Datentyp	Beschreibung
Id	Int32	Primary Key, um den Handler eindeutig zu identifizieren. Wird automatisch gesetzt, sobald der Handler in die Datenbank gespeichert wird.
CreatedTime	DateTime	Timestamp, der gesetzt wird, wenn die Mailbox der Datenbank hinzugefügt wird.
DeletedTime	DateTime	Timestamp, der gesetzt wird, wenn die Mailbox aus dem UI gelöscht wird.

### Assoziationen

Property	Multiplizität	Beschreibung
Classifier	1	Klassifizierer, der verwendet wird, um das Mail zu klassifizieren.
Parent	0..1	Eine Zuweisung zu einem übergeordneten CategorizationHandler.
Childs	*	Eine Zuweisung zu einem untergeordneten CategorizationHandler.
DefaultCategory	1	Standard Kategorie, welche angewandt wird, wenn die Klassenzugehörigkeit einer Eingabenachricht nicht eindeutig bestimmt werden kann.

### LanguagelIdentifierConfiguration

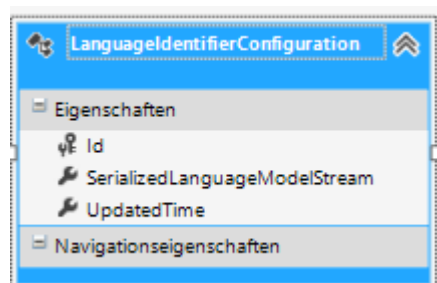


Abbildung 7.24: Datenbankklasse: LanguagelIdentifierConfiguration

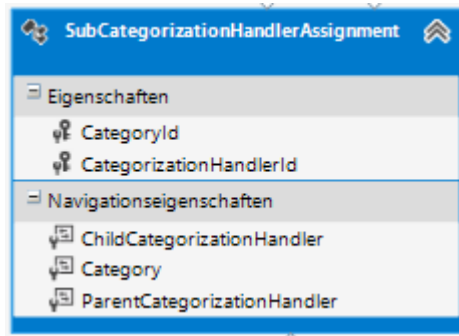
### Zweck

Konfigurationsobjekt für den LanguagelIdentifier. Für jede unterstützte Sprache kann eine Wörterliste eingetragen werden.

### Attribute

Attribute	Datentyp	Beschreibung
Id	Language	Primary Key, bestehend aus einem Enum.
SerializedLanguageModelStream	Binary	Stream einer Wörterliste für den LanguagelIdentifier.
UpdatedTime	DateTime	Timestamp, der gesetzt wird, wenn die Konfiguration der Datenbank hinzugefügt wird.

## SubCategorizationHandlerAssignment



**Abbildung 7.25:** Datenbankklasse: SubCategorizationHandlerAssignment

**Zweck** Ein SubCategorizationHandlerAssignment beschreibt die Eltern/Kind-Beziehung zwischen den CategorizationHandler. Ein CategorizationHandler kann pro Kategorie, die er über seinen AdaptiveClassifier kennt, einen ChildCategorizationHandler haben.

### Attribute

Attribute	Datentyp	Beschreibung
CategoryId	Int32	Teil des zusammengesetzten PKs. Die CategoryId ist die Id der Kategorie, die für den ChildCategorizationHandler definiert wird.
CategorizationHandlerId	Int32	Teil des zusammengesetzten PKs. Ist die Id des CategorizationHandler, der als Child definiert wird.

### Assoziationen

Property	Multiplizität	Beschreibung
Category	1	Kategorie über die sich die Beziehung definiert.
ChildCategorizationHandler	1	CategorizationHandler, der für die entsprechende Kategorie als Child definiert wird.
ParentCategorizationHandler	1	CategorizationHandler, der für die entsprechende Kategorie als Parent, definiert wird.

## Category

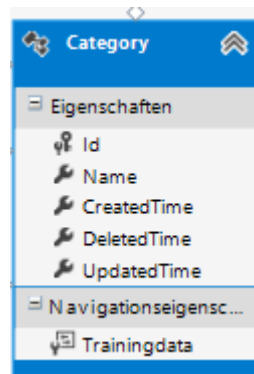


Abbildung 7.26: Datenbankklasse: Category

### Zweck

Eine Category Instanz steht für eine Kategorie, der ein E-Mail zugeteilt werden kann. Jede Kategorie besitzt Trainingsdaten.

### Attribute

Attribute	Datentyp	Beschreibung
Id	Int32	PK, der die Kategorie eindeutig identifiziert. Wird automatisch gesetzt sobald die Kategorie in der Datenbank gespeichert wird.
Name	String	Name für die Category.
CreatedTime	DateTime	Zeitstempel, der beim Speichern der Kategorie in der Datenbank gesetzt wird.
UpdatedTime	DateTime	Zeitstempel, der beim Updaten der Kategorie in der Datenbank gesetzt wird.
DeletedTime	DateTime	Zeitstempel, der beim Löschen der Kategorie in der Datenbank gesetzt wird.

### Assoziationen

Property	Multiplizität	Beschreibung
Trainingdata	*	Trainingsdaten der Kategorie.

## Trainingdata

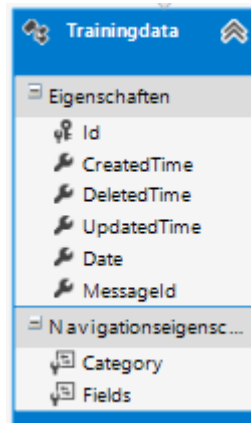


Abbildung 7.27: Datenbankklasse: Trainingdata

### Zweck

Eine Trainingdata Instanz ist eine E-Mail, die einer Kategorie zugeordnet ist. Sie kann für das Training eines Klassifizierers verwendet werden. Ein Trainingsdatenelement hat mehrere Felder, die die wesentlichen Daten enthalten.

### Attribute

Attribute	Datentyp	Beschreibung
Id	Int32	PK einer Trainingsdaten Instanz. Wird beim Speichern in der Datenbank automatisch erzeugt.
Messageld	String	Eindeutige Id der ursprünglichen Email.
CreatedTime	DateTime	Zeitstempel, der beim Speichern des Trainingsdatenelements in der Datenbank gesetzt wird.
UpdatedTime	DateTime	Zeitstempel, der beim Updaten des Trainingsdatenelements in der Datenbank gesetzt wird.
DeletedTime	DateTime	Zeitstempel, der beim Löschen des Trainingsdatenelements in der Datenbank gesetzt wird.

### Assoziationen

Property	Multiplizität	Beschreibung
Fields	*	Die Felder, aus denen eine Trainingsdata Instanz besteht.
Category	1	Kategorie, die dem Trainingsdatenelement zugeordnet wird.

## Field



Abbildung 7.28: Datenbankklasse: Field

### Zweck

Die Field Entity ist ein Bestandteil der Trainingsdata Entity. Sie definiert die eigentliche Nutzdaten eines Trainingsdaten Instanz. Das Feld steht hierbei für die Sektionen eines E-Mails; z.B der Betreff und der Textkörper.

### Attribute

Attribute	Datentyp	Beschreibung
TrainingsdataId	Int32	Teil des PKs. Ist der Primary Key, der zugeordneten Trainingsdata Entity.
FieldType	FieldType	Teil des PKs. Enum, der darüber Auskunft gibt, um was für ein Art Feld es sich handelt.
Content	String	Der eigentliche Inhalt des Felds.

### ProbabilisticCategorizationResult

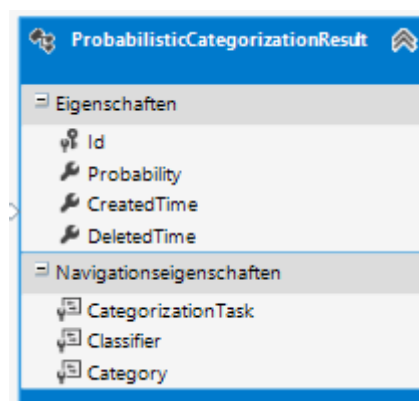


Abbildung 7.29: Datenbankklasse: ProbabilisticCategorizationResult

### Zweck

Die ProbabilisticCategorizationResult Entity repräsentiert eine probabilistische Zuweisung eines E-Mails zu einer Kategorie. Sie bestimmt, wie hoch die Wahrscheinlichkeit ist, dass das betrachtete Mail zu jener Kategorie gehört.

Bei einem Klassifizierungsvorgang wird für jede Kategorie, die der Klassifizierer unterstützt, eine solche Instanz erzeugt.

### Attribute

Attribute	Datentyp	Beschreibung
Id	Int32	PK einer Trainingdata Instanz. Wird beim Speichern in der Datenbank automatisch erzeugt.
Probability	Double	Die Wahrscheinlichkeit, mit wie hoher Bestimmtheit das Mail zu der zugehörigen Kategorie gehört.
CreatedTime	DateTime	Zeitstempel, der beim erstmaligen Speichern der Entity in der Datenbank gesetzt wird.
DeletedTime	DateTime	Zeitstempel, der beim Löschen der Entity in der Datenbank gesetzt wird.

### Assoziationen

Property	Multiplizität	Beschreibung
CategorizationTask	1	Ist der zugehörige CategorizationTask für den die Klassenzugehörigkeit berechnet wird.
Category	1	Kategorie, für welche die Wahrscheinlichkeit der Zugehörigkeit berechnet wird.
Classifier	1	Die Klassifizierer Instanz, die das Resultat berechnet hat.

### CategorizationTask

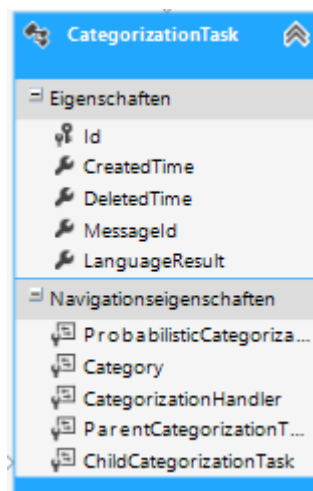


Abbildung 7.30: Datenbankklasse: CategorizationTask

### Zweck

Für jedes E-Mail, das klassifiziert werden soll, wird ein Klassifizierungsvorgang gestartet. Dieser wird über eine CategorizationTask Instanz repräsentiert. Über sie kann erfahren werden, welche



Entscheidungen von welchem adaptiven Klassifizierer oder CategorizationHandler getroffen wurden. Durch das Aneinanderreihen von Handlern kann nach einem abgeschlossenen Vorgang direkt ein weiterer folgen. Die Properties ParentCategorizationTask und ChildCategorizationTask helfen dabei, die Navigation zwischen den Klassifizierungsvorgängen zu ermöglichen.

Besitzt ein Klassifizierungsvorgang kein Child, so ist er der endgültige Entscheidungsträger bei der Bestimmung der Klassenzugehörigkeit.

### Attribute

Attribute	Datentyp	Beschreibung
Id	Int32	PK einer CategorizationTask Instanz. Wird beim Speichern in der Datenbank automatisch erzeugt.
CreatedTime	DateTime	Zeitstempel, der beim erstmaligen Speichern der Entity in der Datenbank gesetzt wird.
DeletedTime	DateTime	Zeitstempel, der beim Löschen der Entity in der Datenbank gesetzt wird.
MessageId	String	MessageId des E-Mails, das klassifiziert werden soll. Über dieses Attribut kann zu einem späteren Zeitpunkt nachvollzogen werden, welches Mail klassifiziert worden ist.
LanguageResult	Language	Ein Enum Wert, der die von der Software erkannte Sprache definiert.

### Assoziationen

Property	Multiplizität	Beschreibung
ProbabilisticCategorizationResult	*	Beschreibt die Wahrscheinlichkeit einer Zugehörigkeit zu einer Kategorie.
Category	1	Kategorie, der das E-Mail bei dem aktuellen Vorgang zugeteilt wurde.
CategorizationHandler	1	CategorizationHandler, der die Resultate berechnet hat.
ParentCategorizationTask	0..1	Ist durch das CategorizationHandler Chaining vorgängig schon eine Kategorie bestimmt worden, so ist dieses Resultat über dieses Property ersichtlich.
ChildCategorizationTask	0..1	Wird durch CategorizationHandler Chaining nachfolgend noch weiter eine Kategorie berechnet, so ist das nachfolgende Resultat über dieses Property ersichtlich.

#### 7.5.4.2 Enums

Im Folgenden Abschnitt werden die verschiedenen Enums die in der Datenbank verwendet werden kurz aufgelistet und beschrieben. Als unterliegenden Datentyp wird immer ein Int32 verwendet.

**FieldType**

Ein E-Mail besteht aus verschiedenen Feldern, die Textelemente enthalten. Um diese Daten strukturiert in der Datenbank ablegen zu können, wird der Enum FieldType verwendet.

Werte des Enums:

- Body: Textkörper
- Subject: Betreffzeile
- From: Sender
- To: Empfängerliste

**KCrossValidationType**

Wirkt als diskriminierendes Element für die Auswahl des K-Cross Validierungs Parameter.

Werte des Enums:

- FixedSize: Die Validierung wird mit Sets fixer Grösse durchgeführt
- NSets: Die Validierung wird mit einer bestimmten Anzahl Sets durchgeführt

**Language**

Definiert, welcher Sprache ein E-Mail angehört. Falls keine Sprache erkannt werden konnte, wird der Wert *Other* verwendet.

Werte des Enums:

- English
- German
- French
- Italian
- Other

**SvmKernel**

Gibt vor, welche Art von Kernel Funktion für die SVM verwendet wird.

Werte des Enums:

- Linear
- Gaussian
- ChiSquare
- Polynomial

**TaskStatus**

Status wie ein Training der SVM beendet wurde.

Werte des Enums:

- Cancelled: Der Trainingsvorgang wurde durch den Benutzer manuell beendet
- FinishedSuccessfully: Der Trainingsvorgang wurde korrekt beendet.
- Failed: Der Optimierungsvorgang konnte keine konvergente Lösung finden.

**TokenizationType**

Beschreibt, was für eine Art von Tokenization durchgeführt werden soll.

Werte des Enums:

- StemmingTokenization: Es wird ein Stemming Algorithmus angewendet, der versucht, die Wörter in eine Grundform zu bringen.
- WhiteSpaceTokenization: Die Texte werden nur anhand von Leer- und Satzzeichen in Wörter aufgetrennt und anschliessend in Kleinschreibweise gebracht.

## 7.5.5 Interfaces: Core.Interfaces

Beim Design der Applikation wurde möglichst darauf geachtet, dass die Architektur modular aufgebaut ist. Alle einzelnen Teile der Applikation werden durch Interfaces oder abstrakte Klassen beschrieben. Auch im Code werden nur über diese die Objekte referenziert. Daraus resultiert eine geringe Kopplung (low coupling) und hohe Kohäsion (high cohesion) der einzelnen Komponenten. Dieser Ansatz ermöglicht später das Austauschen eines Algorithmus oder Methode ohne Änderung an der bestehenden Architektur oder anderen Komponenten. Des Weiteren wird so das Schreiben von Unit Tests vereinfacht, da Interfaces mit Hilfe eines Mocking Frameworks gemockt werden können.

Folgend kommt eine Übersicht und Beschreibung der einzelnen Interfaces und abstrakten Klassen. Wenn möglich wurden Interfaces verwendet, um die Schnittstellen zu den einzelnen Komponenten zu beschreiben. Es wurden abstrakte Klassen verwendet falls in der Vaterklasse gemeinsame Logik vorhanden ist.

Alle Interfaces beginnen mit *I*, alle abstrakte Klassen mit *Abstract*.

### 7.5.5.1 IBagOfWords

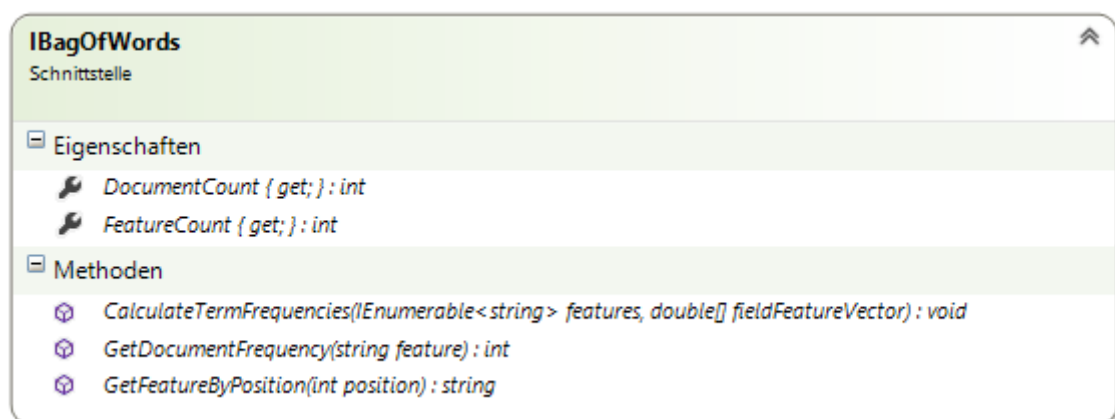


Abbildung 7.31: Interface: BagOfWords

Das *IBagOfWords*-Interface beschreibt den *Bag of Words*, der bei einer Klassifizierung benötigt wird. Ein *Bag of Words* beinhaltet alle Features (Schlüsselwörter) in unstrukturierter und unsortierter Form.

#### 7.5.5.1.1 Properties

- *int DocumentCount* - Anzahl Dokumente, die für die Erstellung des *IBagOfWords* verwendet wurden.
- *int FeatureCount* - Anzahl Features, die der *IBagOfWords* beinhaltet.

### 7.5.5.1.2 Methoden

```
1 void CalculateTermFrequencies(IEnumerable<string> features, double[]  
    fieldFeatureVector)
```

Berechnet die Termfrequency basierend auf der Liste von Features, die mitgegeben werden, und speichert die berechneten Werte im mitgegebenen Parameter *fieldFeatureVector*.

#### Parameter

- *IEnumerable < string > features* - Liste von Features.
- *double[] fieldFeatureVector* - Term Vektor, in dem die Berechnung der Termfrequency gespeichert wird.

```
1 int GetDocumentFrequency(string feature)
```

Gibt die Documentfrequency für ein mitgegebenes Feature zurück.

#### Parameter

- *feature* - Schlüsselwort, für das die DocumentFrequency zurückgegeben werden sollte.

#### Returnwert

Typ: int

Die Documentfrequency.

```
1 string GetWordByPosition(int position)
```

Gibt das Feature für die angegebene Position im Feature Vektor zurück.

#### Parameter

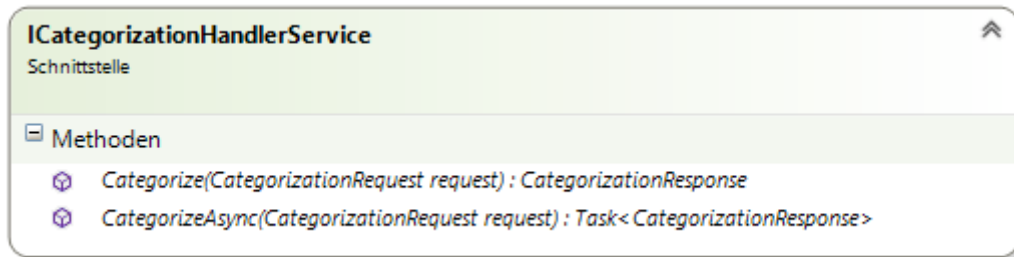
- *int position* - Position eines Wortes im Feature Vektor

#### Returnwert

Typ: string

Das Wort im Feature Vektor an der gewünschten Stelle.

### 7.5.5.2 ICategorizationHandlerService



**Abbildung 7.32:** Interface: CategorizationHandlerService

Das *ICategorizationHandlerService*-Interface spezifiziert den *CategorizationHandlerService*. Der *CategorizationHandlerService* ist die Schnittstelle, die es anzusprechen gilt, um ein E-Mail zu kategorisieren. Das Interface stellt zwei Methoden zur Verfügung, eine synchrone und eine asynchrone, um den Klassifizierungsvorgang zu starten.

#### 7.5.5.2.1 Methoden

```
1 Task<CategorizationResponse> CategorizeAsync(CategorizationRequest request)
```

Asynchrone Methode, um eine Klassifizierung eines Mails durchzuführen.

##### Parameter

- *CategorizationRequest request* - Kapselt alle benötigten Daten, die für einen Klassifizierungsvorgang benötigt werden.

##### Returnwert

Typ: `Task<CategorizationResponse>`

Task Objekt, das die *CategorizationResponse* beinhaltet. Der *CategorizationResponse* kann die zugewiesene Kategorie entnommen werden. Zusätzlich beinhaltet die *CategorizationResponse* die Wahrscheinlichkeiten für die anderen Kategorien und die erkannte Sprache. Wurde für die Klassifizierung eine *CategorizationHandler Chain* spezifiziert, kann über die *CategorizationResponse* der ganze Prozess nachvollzogen werden.

```
1 CategorizationResponse Categorize(CategorizationRequest request)
```

Führt die synchrone Klassifizierung eines E-Mails durch.

##### Parameter

- *CategorizationRequest request* - Kapselt alle benötigten Daten, die für einen Klassifizierungsvorgang benötigt werden.

### Returnwert

Typ: CategorizationResponse

Der CategorizationResponse kann die zugewiesene Kategorie entnommen werden. Zusätzlich beinhaltet die CategorizationResponse die Wahrscheinlichkeiten für die anderen Kategorien und die erkannte Sprache. Wurde für die Klassifizierung eine CatgorizationHandler Chain spezifiziert, kann über die CategorizationResponse der ganze Prozess nachvollzogen werden.

### 7.5.5.3 ICategoryDecider

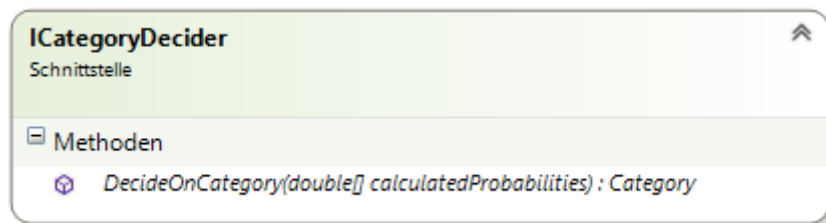


Abbildung 7.33: Interface: ICategoryDecider

Das *ICategoryDecider*-Interface beschreibt die Komponente, die die Entscheidung trifft, welcher Kategorie ein Mail schlussendlich zuzuordnen ist. Für diese Entscheidung braucht der *ICategoryDecider* die Wahrscheinlichkeiten aller anderen Kategorien.

#### 7.5.5.3.1 Methoden

```
1 Category DecideOnCategory(double [] calculatedProbabilities)
```

Entscheidet sich für eine Kategorie. In der Regel wird es die Kategorie mit der höchsten Wahrscheinlichkeit sein. Tritt der Fall ein, dass zwei Kategorie die gleiche höchste Wahrscheinlichkeit besitzt, so wird die vordefinierte Default Kategorie ausgewählt.

#### Parameter

- `double[] calculatedProbabilities` - Wahrscheinlichkeiten der Zugehörigkeit pro SVM Klasse.

#### Returnwert

Typ: Category

Kategorie, die einem E-Mail definitiv zugeordnet wird.

### 7.5.5.4 IConcreteCategorizationHandler

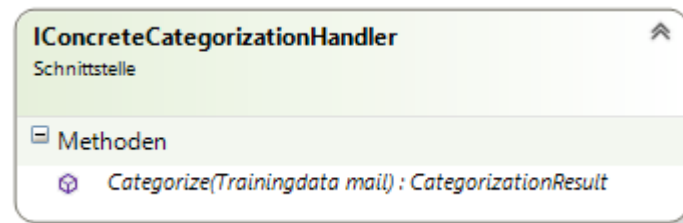


Abbildung 7.34: Interface: IConcreteCategorizationHandler

Das *IConcreteCategorizationHandler*-Interface beschreibt die Komponente, welche die konkrete Klassifizierung im Klassifizierungsprozess durchführt.

#### 7.5.5.4.1 Methoden

```
1 CategorizationResult Categorize(Trainingdata mail)
```

Führt eine Klassifizierung eines E-Mails durch.

#### Parameter

- *Trainingdata mail* - E-Mail, das Klassifiziert werden soll.

#### Returnwert

Typ: *CategorizationResult*

Dem *CategorizationResult* kann die zugewiesene Kategorie entnommen werden. Zusätzlich beinhaltet das *CategorizationResult* die Wahrscheinlichkeiten für die anderen Kategorien und die erkannte Sprache.

### 7.5.5.5 IFeatureExtractor

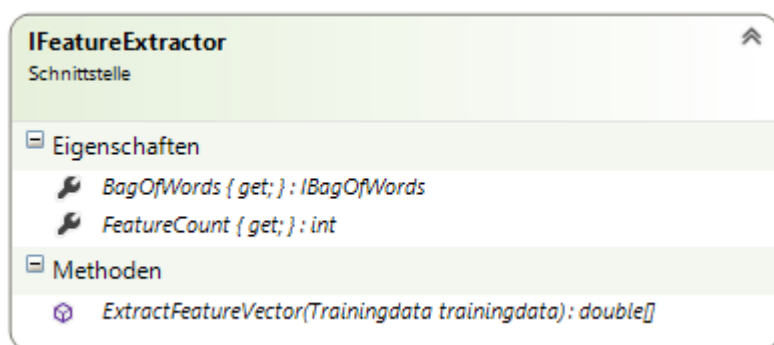


Abbildung 7.35: Interface: IFeatureExtractor



Das *IFeatureExtractor*-Interface beschreibt die Komponente, welche die Feature Vektoren berechnet.

#### 7.5.5.5.1 Properties

- *IBagOfWords* *BagOfWords* - Beinhaltet alle Features, die verwendet werden.
- *int* *FeatureCount* - Anzahl Features, die der *IBagOfWords* beinhaltet.

#### 7.5.5.5.2 Methoden

```
1 double[] ExtractFeatureVector(Trainingdata trainingdata)
```

Berechnet den Feature Vektor eines E-Mails mittels TF-IDF.

#### Parameter

- *Trainingdata* *trainingdata* - E-Mail, das Klassifiziert werden soll.

#### Returnwert

Typ: double[]

Die Repräsentation des E-Mails als Feature Vektor im Vektorraum.

#### 7.5.5.6 ILanguageIdentifizier

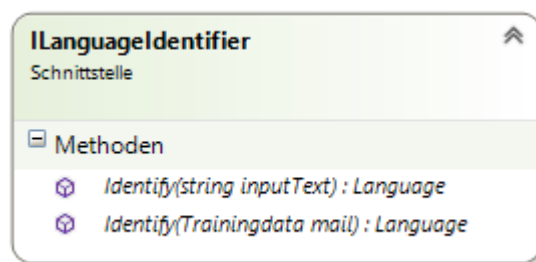


Abbildung 7.36: Interface: ILanguageIdentifizier

Das *ILanguageIdentifizier*-Interface beschreibt die Komponente, welche die geschriebene Sprache eines E-Mails oder eines Strings identifiziert.

#### 7.5.5.6.1 Methoden

```
1 Language Identify(string inputText)
```

Erkennt die geschriebene Sprache eines Strings.

## Parameter

- *string inputText* - String, für den die Sprache erkannt werden soll.

## Returnwert

Typ: Language

Die erkannte Sprache. Konnte die Sprache nicht eindeutig identifiziert werden, wird der Wert *Other* zurückgegeben.

```
1 Language Identify(Trainingdata mail)
```

Erkennt die geschriebene Sprache eines E-Mail.

## Parameter

- *Trainingdata mail* - E-Mail, für das die Sprache erkannt werden soll.

## Returnwert

Typ: Language

Die erkannte Sprache. Konnte die Sprache nicht eindeutig identifiziert werden, wird der Wert *Other* zurückgegeben.

## 7.5.6 DTOs: Core.Interfaces/Dto

### 7.5.6.1 MailDto

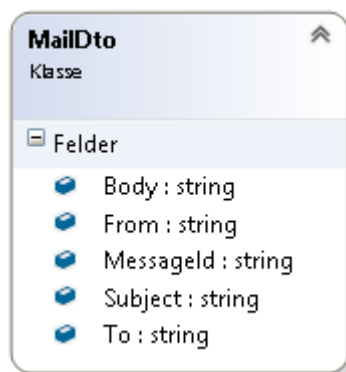


Abbildung 7.37: DTO: MailDto

Das *MailDto* Objekt repräsentiert ein E-Mail, das in einer Request mitgegeben wird. Dieses Mail wird einer Kategorie zugeordnet.

### 7.5.6.1.1 Properties

- *string Body* - Der Fliesstext des Mails.
- *string From* - Absender des Mails.
- *string MessageId* - Eindeutig ID, mit deren Hilfe das Mail identifiziert werden kann.
- *string Subject* - Betreff des E-Mails.
- *string To* - Liste von Adressen, an die das E-Mail gerichtet ist.

### 7.5.6.2 CategorizationRequest

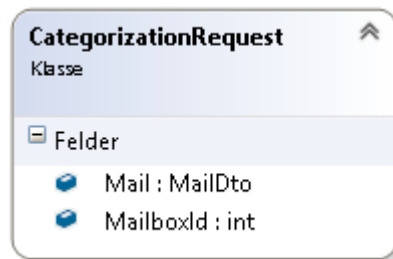


Abbildung 7.38: DTO: CategorizationRequest

Das *CategorizationRequest* Objekt wird benötigt, wenn ein Mail kategorisiert werden soll. Es kapselt das Mail für den Kategorisierungsservice.

### 7.5.6.2.1 Properties

- *MailDto Mail* - Das Mailobjekt, das einer Kategorie zugeordnet werden soll.
- *int MailboxId* - Id der Mailbox, die angesprochen werden soll. Muss vorher konfiguriert worden sein, so dass der Klassifizierungsvorgang erfolgreich durchgeführt werden kann.

### 7.5.6.3 CategorizationResponse

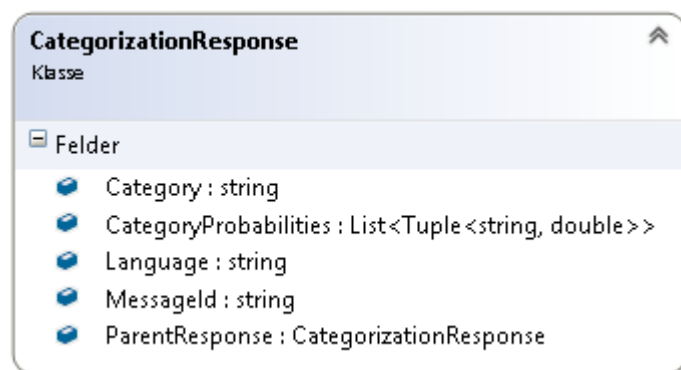


Abbildung 7.39: DTO: CategorizationResponse

Das *CategorizationResponse* Objekt wird als Antwort auf eine abgearbeitete *CategorizationRequest* erhalten. Der *CategorizationResponse* können alle Wahrscheinlichkeiten der Kategorien sowie auch die erkannte Sprache entnommen werden

### 7.5.6.3.1 Properties

- *string Category* - Zugeordnete Kategorie.
- *List < Tuple < string, double >> CategoryProbabilities* - Liste der Wahrscheinlichkeiten der Klassenzugehörigkeit.
- *string Language* - Von der Applikation erkannte Sprache, in der das Mail verfasst wurde.
- *string MessageId* - Eindeutige ID des Mails, das in der Request mitgegeben wurde.
- *CategorizationResponse ParentResponse* - Falls die Zuweisung durch eine Categorization Handler Chain erstellt wurde, kann über dieses Property der ganze Klassifizierungsprozess nachvollzogen werden.

## 7.5.7 Interfaces: MachineLearning.Interfaces

### 7.5.7.1 IMulticlassSupportVectorMachine

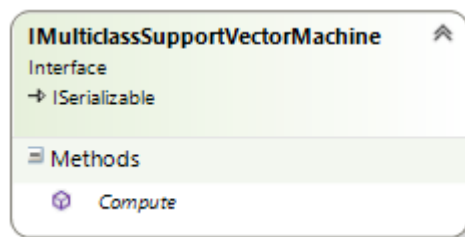


Abbildung 7.40: Interface: IMulticlassSupportVectorMachine

Das *IMulticlassSupportVectorMachine* Interface steht für eine trainierte Multiclass SVM, die eingehende Feature Vektoren klassifizieren kann.

#### 7.5.7.1.1 Methoden

```
1 int Compute(double[] input, out double[] probabilities)
```

Führt den Klassifikationsvorgang für den eingegebenen Feature Vektor aus.

#### Parameter

- *double[] inputs* - Feature Vektor, für den die Klassenzugehörigkeit bestimmt werden soll.
- *out double[] probabilities* - Wahrscheinlichkeit der Klassenzugehörigkeit für jede Klasse der Multiclass SVM.

## Returnwert

Typ: int

Die berechnete SVM Klasse, zu der der Feature Vektor angehört.

### 7.5.7.2 IKernel

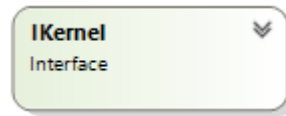


Abbildung 7.41: Interface: IKernel

Das *IKernel* Interface steht als Marker Interface für die Kernelfunktionen.

### 7.5.7.3 IMulticlassSvmTraining

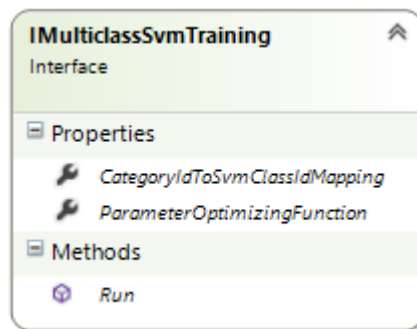


Abbildung 7.42: Interface: IMulticlassSvmTraining

Das *IMulticlassSvmTraining* Interface steht für das Training einer Multiclass SVM. Dazu führt es für jede Sub SVM ein Parameteroptimierungsprozess durch und trainiert anschliessend die Sub SVM mit dem besten Setup.

#### 7.5.7.3.1 Properties

- *ParameterOptimizingFunction* *ParameterOptimizingFunction* - Über dieses Property kann eine externe Komponente eine Funktion für die Ausführung des Parameteroptimierungsprozesses anbinden. Die Funktion wird für jedes binäre Problem der Multiclass SVM aufgerufen. Die externe Komponente berechnet das beste Setup und gibt es zurück.
- *CategoryIdToSvmClassIdMapping* *CategoryIdToSvmClassIdMapping* - Das Mapping zwischen der Kategorie-Id und der bei 0 beginnenden Multiclass SVM Klassen-Id.

#### 7.5.7.3.2 Methoden

```
1 CrossValidationResult Run()
```

Führt das Training der Multiclass SVM aus.

### Returnwert

Typ: `IMulticlassSupportVectorMachine`

Die SVM, die beim Training ermittelt wurde.

#### 7.5.7.4 `ISequentialMinimalOptimization`

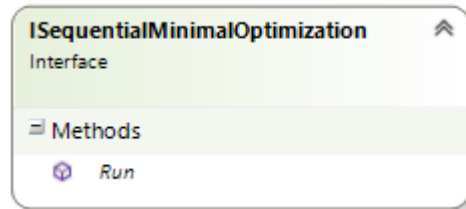


Abbildung 7.43: Interface: `ISequentialMinimalOptimization`

Das `ISequentialMinimalOptimization` Interface steht für den Vorgang des effektiven Trainings der Sub SVM. Dabei wird ein Algorithmus zur Lösung des Optimierungsproblem angewendet.

##### 7.5.7.4.1 Methoden

```
1 CrossValidationResult Run()
```

Führt das finale Training aus.

### Returnwert

Typ: `double`

Die berechnete Fehlerrate der Sub SVM.

#### 7.5.7.5 `IGridSearch`

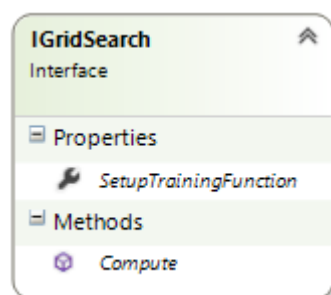


Abbildung 7.44: Interface: `IGridSearch`

Das `IGridSearch` Interface steht für den Vorgang des GridSearch Algorithmus. Bei diesem wird aus einer Liste von Parametern alle möglichen Kombinationen ermittelt und für jede einzelne ein Training ausgeführt. Eine Kombination wird hierbei Setup genannt.

Zum Schluss wird das beste Setup zurückgegeben.

### 7.5.7.5.1 Properties

- *SetupTrainingFunction SetupTrainingFunction* - Über dieses Property kann eine externe Komponente eine Funktion für Ausführung eines Setup-Trainings anbinden. Die Funktion wird für jede mögliche Kombination der Parameter aufgerufen. Die externe Komponente berechnet anschliessend die Fehlerrate des Setups und gibt diese zurück.

### 7.5.7.5.2 Methoden

```
1 CrossValidationResult Compute()
```

Führt den GridSearch Vorgang aus und evaluiert die beste Parametereinstellung.

### Returnwert

Typ: GridSearchResult

Objekt, das das beste Setup sowie dessen Fehlerrate enthält. Erweiternd werden auch alle trainierten Setups und deren Qualitätsaussagen angegeben.

### 7.5.7.6 ICrossValidation

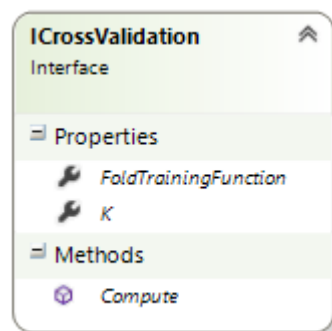


Abbildung 7.45: Interface: ICrossValidation

Das *ICrossValidation* Interface steht für den K-Fold Cross Validation Vorgang, der über die Methode *Compute()* gestartet werden kann. Dabei werden die eingehenden Trainingsdaten in  $k$  Subsets aufgeteilt und jedes einzelne davon einmalig für die Validierung verwendet. Die anderen Subsets werden vorhergehend für das Training der Maschine verwendet.

#### 7.5.7.6.1 Properties

- *FoldTrainingFunction FoldTrainingFunction* - Über dieses Property kann eine externe Komponente eine Funktion für die Behandlung der Schritte der K-Fold Cross Validierung anbinden. Die Funktion wird bei jedem Zwischenschritt der Validierung aufgerufen. Die externe Komponente berechnet anschliessend die Fehlerrate des Folds und gibt diese zurück.
- *int K* - Die Anzahl Folds, die festgelegt wurden.

### 7.5.7.6.2 Methoden

```
1 CrossValidationResult Compute()
```

Führt die Cross Validierung aus und berechnet die mittlere Fehlerrate über die Gesamtheit der durchgeführten Schritte.

#### Returnwert

Typ: CrossValidationResult

Die mittlere Fehlerrate.

### 7.5.7.7 IFoldTraining

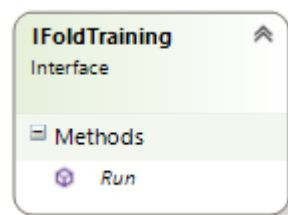


Abbildung 7.46: Interface: IFoldTraining

Das *IFoldTraining* Interface steht für den einzelnen Vorgang eines K-Fold Cross Validation Zwischenschrittes. Dazu gehört das Training einer temporären SVM mit den angegebenen Parametern sowie die Validation nach dem Training.

#### 7.5.7.7.1 Methoden

```
1 CrossValidationResult Run()
```

Führt den Zwischenschritt aus.

#### Returnwert

Typ: double

Die berechnete Fehlerrate der Validierung.



## 7.5.8 Delegates: MachineLearning.Interfaces

### 7.5.8.1 ParameterOptimizingFunction

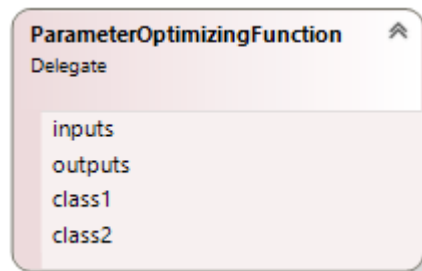


Abbildung 7.47: Delegate: ParameterOptimizingFunction

Der *ParameterOptimizingFunction* Delegate steht für die Ausführung des Parameteroptimierungsprozesses einer Sub SVM.

#### 7.5.8.1.1 Parameter

- `double[][] inputs` - Feature Vektoren des 2-Klassen Problems.
- `int[] outputs` - Die erwarteten Ausgabewerte der entsprechenden Feature Vektoren.
- `int class1` - Id der ersten Klasse des 2-Klassen Problems.
- `int class2` - Id der zweiten Klasse des 2-Klassen Problems.

#### 7.5.8.1.2 Rückgabe

- `double` - Die beste Fehlerrate, die bei der Parameteroptimierung erreicht wurde.

### 7.5.8.2 SetupTrainingFunction

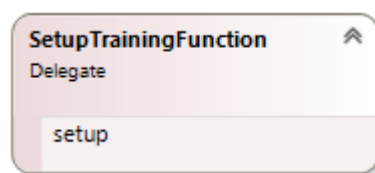


Abbildung 7.48: Delegate: SetupTrainingFunction

Der *SetupTrainingFunction* Delegate steht für die Ausführung eines Trainings mit bestimmten Parametereinstellungen.

#### 7.5.8.2.1 Parameter

- `Setup setup` - Parametereinstellungen, die für das Training verwendet werden sollen.

### 7.5.8.2.2 Rückgabe

- *double* - Die Fehlerrate, die mit diesem Setup erreicht wurde.

### 7.5.8.3 FoldTrainingFunction

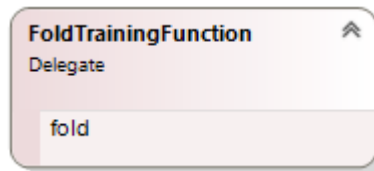


Abbildung 7.49: Delegate: FoldTrainingFunction

Der *FoldTrainingFunction* Delegate steht für den Vorgang eines Schrittes der K-Fold Cross Validation.

#### 7.5.8.3.1 Parameter

- *Fold fold* - Parameterobjekt für die Übergabe der Trainings- und Validierungsdaten.

#### 7.5.8.3.2 Rückgabe

- *double* - Die Fehlerrate des Folds.

## 7.5.9 DTOs: MachineLearning.Interfaces/Dto

### 7.5.9.1 ClassPair

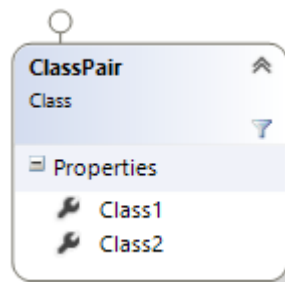


Abbildung 7.50: DTO: ClassPair

Das *ClassPair* Objekt repräsentiert ein Schlüssel für ein binäres Problem, bei dem zwei verschiedene Klassen einer SVM gegeneinander antreten.

#### 7.5.9.1.1 Properties

- *string Class1* - Klasse 1, identifiziert durch ihren Namen.
- *string Class2* - Klasse 2, identifiziert durch ihren Namen.

### 7.5.9.2 Setup

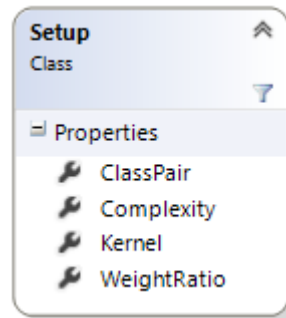


Abbildung 7.51: DTO: Setup

Das *Setup* Objekt hält alle Parameter, die für ein Training genutzt werden können. Sie werden verwendet für das Training einer SVM.

#### 7.5.9.2.1 Properties

- *ClassPair ClassPair* - Identifizierung des 2-Klassen Problems.
- *double Complexity* - Complexity Wert, der für das Training verwendet wird.
- *IKernel Kernel* - Kernelfunktion, für die das 2-Klassen Problem gelöst werden soll.
- *double WeightRatio* - Die Gewichtung der Complexity auf die beiden Klassen.

### 7.5.9.3 Fold

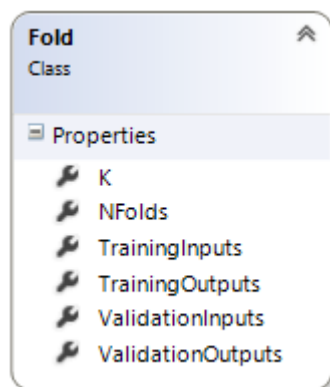


Abbildung 7.52: DTO: Fold

Das *Fold* Objekt steht für ein Validierungsschritt des K-Fold Cross Validation Algorithmus. Darin sind die Trainings- und die Validationsdaten.

### 7.5.9.3.1 Properties

- *int K* - Aktueller Fold-Index des Vorgangs.
- *int NFolds* - Anzahl Folds, die für die K-Fold Cross Validierung gemacht werden.
- *double[][] TrainingInputs* - Die Feature Vektoren für den Trainingsvorgang.
- *int[] TrainingOutputs* - Die Klassenzugehörigkeiten der Feature Vektoren der Trainingsphase. Die Reihenfolge entspricht derselben der Eingabevektoren.
- *double[][] ValidationInputs* - Die Feature Vektoren für den Validationsvorgang.
- *int[] ValidationOutputs* - Die Klassenzugehörigkeiten der Feature Vektoren der Validationsphase. Die Reihenfolge entspricht derselben der Eingabevektoren.

### 7.5.9.4 GridSearchResult

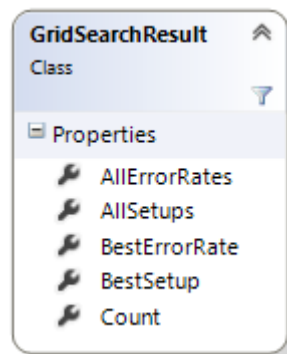


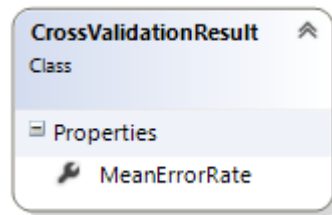
Abbildung 7.53: DTO: GridSearchResult

Das *GridSearchResult* Objekt steht für das Resultat des GridSearch-Algorithmus.

#### 7.5.9.4.1 Properties

- *double[] AllErrorRates* - Die Fehlerraten aller trainierten Setups.
- *Setup[] AllSetups* - Alle Setups, die trainiert wurden. Die Reihenfolge der Elemente deckt diejenige der Fehlerraten.
- *double BestErrorRate* - Die beste Fehlerrate, die erreicht wurde.
- *Setup BestSetup* - Das Setup, das die beste Fehlerrate erzielt hat.
- *int Count* - Anzahl an Setups.

### 7.5.9.5 CrossValidationResult



**Abbildung 7.54:** DTO: CrossValidationResult

Das *CrossValidationResult* Objekt steht für das Resultat des K-Fold Cross Validation Vorgangs.

#### 7.5.9.5.1 Properties

- *double MeanErrorRate* - Die gemittelte Fehlerrate über alle Validationsschritte.

## 7.6 Konfiguration

In diesem Kapitel wird die Konfigurationsmöglichkeit der Applikation genauer beschrieben. Die Applikation kann an zwei verschiedenen Orten konfiguriert werden. Die verschiedenen Parameter, aus denen während des Trainings die besten bestimmt werden, werden im *configuration.xml* definiert. Alle anderen Optionen sind über das User Interface einstellbar.

Da für eine sinnvolle Parameterwahl, Kenntnisse über die unterliegenden Algorithmen sowie mathematisches Verständnis braucht, wurde entschieden, dass die Konfiguration statt im User-Interface in einem Konfigurations-XML vorgenommen werden soll. Der Anwender der Software soll nicht mit der Wahl von Parametern überfordert werden, die er ohne sorgfältiges Einarbeiten in den Algorithmus, nicht versteht. Stattdessen wird ein Set von sinnvollen Parametern mit der Applikation im *configuration.xml* mitgeliefert und in diesem Teil Empfehlungen gemacht. Falls zu einem späteren Zeitpunkt dieses Parameterset angepasst werden sollen, kann dies über eben jenes XML vorgenommen werden. Das XML kann während dem Betrieb verändert werden.

### 7.6.1 Trainingsparameter - configuration.xml

Wenn ein Trainingsvorgang gestartet wird, kann eine Vielzahl von Parametern mitgegeben werden. Der Trainingsalgorithmus ermittelt aus allen mitgegeben Parametern die optimale Übereinstimmung, um so die Erfolgsrate des Klassifizierers zu maximieren. Je mehr Variationen von Parametern vorhanden sind desto länger wird der Trainingsvorgang gehen. Die Parameter werden im *configuration.xml* definiert und vor jedem Trainingsprozess ausgelesen.

Beispiel des Konfiguration-XML:

```

1 <?xml version="1.0" encoding="utf-8"?>
3 <XmlClassifierEvaluationConfiguration>
4   <CacheSize>200</CacheSize>
5   <WeightRatios>
6     <double>0.8</double>
7     <double>0.9</double>
8     <double>1</double>
9   </WeightRatios>
10  <Complexities>
11    <double>0.01</double>
12    <double>0.02</double>
13  </Complexities>
14  <MinimalOccurrence>2</MinimalOccurrence>
15  <NFrequentWordsToDelete>100</NFrequentWordsToDelete>
16  <KCrossValidationType>NSets</KCrossValidationType>
17  <NSets>10</NSets>
18  <SetSize>100</SetSize>
19  <EmailFieldConfigurationSetups>
20    <EmailFieldConfigurationSetup>
21      <EmailFieldConfigurations>
22        <EmailFieldConfiguration>
23          <FieldType>Body</FieldType>
24          <Weight>1</Weight>
25          <TokenizationType>StemmingTokenization</TokenizationType>
26        </EmailFieldConfiguration>

```

```

27     <EmailFieldConfiguration>
        <FieldType>Subject</FieldType>
29     <Weight>1</Weight>
        <TokenizationType>StemmingTokenization</TokenizationType>
31     </EmailFieldConfiguration>
    </EmailFieldConfigurations>
33 </EmailFieldConfigurationSetup>
</EmailFieldConfigurationSetups>
35 <LinearKernels>
    <LinearKernel>
37     <Constant>1</Constant>
    </LinearKernel>
39     <LinearKernel>
        <Constant>2</Constant>
41     </LinearKernel>
    </LinearKernels>
43 <GaussianKernels>
    <GaussianKernel>
45     <Sigma>1</Sigma>
    </GaussianKernel>
47     <GaussianKernel>
        <Sigma>1.5</Sigma>
49     </GaussianKernel>
    <GaussianKernel>
51     <Sigma>2</Sigma>
    </GaussianKernel>
53 </GaussianKernels>
    <ChiSquareKernels>
55     <ChiSquareKernel>
    </ChiSquareKernel>
57 </ChiSquareKernels>
    <PolynomialKernels>
59     <PolynomialKernel>
        <Constant>1</Constant>
61     <Degree>1</Degree>
    </PolynomialKernel>
63     <PolynomialKernel>
        <Constant>1</Constant>
65     <Degree>2</Degree>
    </PolynomialKernel>
67     <PolynomialKernel>
        <Constant>2</Constant>
69     <Degree>2</Degree>
    </PolynomialKernel>
71 </PolynomialKernels>
</XmlClassifierEvaluationConfiguration>

```

### XmlClassifierEvaluationConfiguration

Auftreten: 1

Beschreibung: Root Element.

Kindelemente:

- CacheSize
- WeightRatios
- Complexities
- EmailFieldConfigurationSetups
- LinearKernels
- GaussianKernels
- ChiSquareKernels
- PolynomialKernels

### CacheSize

Auftreten: 1

Beschreibung: Die Cache-Grösse ist eines der bestimmenden Elemente, wenn es um die Geschwindigkeit des Trainingsvorgang geht. Jede Support-Vektor Maschine besitzt einen solchen Cache. In diesem werden die berechneten Werte der Kernel-Funktion zwischen zwei Punkten in einem Dictionary abgelegt. Berechnungen, die oft durchgeführt werden müssen, können so schnell wieder gefunden werden. Negativ wirkt sich der Cache auf die Speichernutzung aus, da er für jede Maschine angelegt werden muss. Die Anzahl Maschinen wird durch die Anzahl Kategorien bestimmt:

$$N \text{ machines} = \frac{N \text{ categories} * (N \text{ categories} - 1)}{2}$$

Dessen Grösse ist von der Anzahl Trainingsdaten abhängig. Weiterhin spielt es eine Rolle, ob die Features der Feature-Vektoren überhaupt definiert sind (Bei der E-Mail Klassifikation wird nur ein kleiner Teil der Features gesetzt, da in einem Mail nicht alle Worte aus dem Korpus verwendet werden)

Im worst case ist die Cache-Grösse einer einzelnen SVM  $\mathcal{O}(N \text{ trainingdata}^2)$

Das bedeutet für eine Multiclass SVM:

$$\mathcal{O}\left(\frac{N \text{ categories} * (N \text{ categories} - 1)}{2} * \left(\frac{N \text{ trainingdata}}{N \text{ categories}}\right)^2\right)$$

Bei einer Multiclass SVM mit 4 Klassen und 1000 Dokumenten und ausgeglichener Verteilung der Trainingsdaten wird somit im Extremfall für den Trainingsvorgang  $\frac{4 * (4 - 1)}{2} * \left(\frac{1000}{N \text{ categories}}\right)^2 * 8 \text{ Byte} = 3 \text{ MByte}$  Speicher benötigt.

### Empfehlung

Die Grösse des Caches kann auch manuell eingestellt werden. Auf diese Weise kann die automatische Allokation verhindert und die Speicherauslastung kontrolliert werden. Angegeben wird die Cache-Grösse über den Konfigurationsparameter "CacheSize".

Der benötigte Speicher einer SVM so auf  $\mathcal{O}(\text{CacheSize}^2)$  heruntergesetzt werden.

Bei normalen Rechnern wird üblicherweise eine CacheSize von 200 empfohlen. Bei Maschinen mit grossen RAM-Speichern kann er aber auch auf 1000 und mehr gesetzt werden.



**WeightRatios**

Auftreten: 1

Beschreibung: Die Gewichtsverteilung ist ein Parameter für den Optimierungsalgorithmus, welcher verwendet wird um die SVM zu trainieren. Die Gewichtsverteilung wirkt sich direkt auf den Komplexität Parameter  $C$  aus.

Eine Gewichtung kleiner als eins, wie zum Beispiel 0.1, bedeutet, dass nur 10% von  $C$  auf die positive Klasse angewendet wird, gleichzeitig aber 100% von  $C$  auf die negative Klasse.

Eine Gewichtung grösser als eins, wie zum Beispiel 10, bedeutet, dass 100% von  $C$  auf die positive Klasse angewendet wird, gleichzeitig aber nur 10% von  $C$  auf die negative Klasse.

Kindelemente:

- double

**Kindelement: double**

Auftreten: 0 oder beliebig viele

Beschreibung: Das Kindelement double steht für einen Gewichtungswert.

Erlaubte Werte für das Element:

- ein ganzzahliger Wert grösser 0
- eine Gleitkommazahl

Beispiel:

**<WeightRatios>**

2     <double>0.5</double>

      <double>1</double>

4     <double>5</double>

**</WeightRatios>****Empfehlung**

Es wird vorgeschlagen, [0.25, 0.5, 1, 2, 4] als Gewichtsverteilung einzustellen. Sind die Anzahl Trainingsdaten der einzelnen Kategorien ausgeglichen, kann auch nur der Wert 1 eingestellt werden.

**Complexities**

Auftreten: 1

Beschreibung: Die Komplexität  $C$  ist der Parameter, der beim Berechnen der Hyperebene massgebend für erlaubte Falschklassifikationen ist. Er erlaubt es das Erstellen einer *weichen Grenze*, indem die Ebene nicht für alle Trainingsdaten stimmen muss. Ein hoher Wert für  $C$  bedeutet, dass die Hyperebene möglichst genau sein muss und lässt so möglichst wenig Falschklassifikationen zu. Wenn die Ebene genauer wird, wird auch das Model besser auf die Trainingsdaten angepasst sein. Das kann aber den Nachteil mit sich ziehen, dass das trainierte Model auf neue Daten nicht gut anwendbar ist und somit Overfitting herrscht.

Ein geringer Wert für  $C$  erlaubt mehr Falschklassifikationen beim Berechnen der Hyperebene. Dies erzeugt ein verstärkt generalisiertes Model.

Kindelemente:

- double

### Kindelement: double

Auftreten: 1 oder beliebig viele

Beschreibung: Das Kindelement double steht für ein Complexity-Wert.

Erlaubte Werte für das Element:

- ein ganzzahliger Wert
- eine Gleitkommazahl

Beispiel:

```

1 <Complexities>
  <double>0.01</double>
3  <double>0.5</double>
  <double>1</double>
5 </Complexities>

```

### Empfehlung

Erfahrungswerte aus verschiedenen Papers zeigen, dass bei der Klassifizierung von Texten 0.01 ein guter Wert für  $C$  ist. Zusätzlich wird aber auch empfohlen, einen weiteren Wert auf 0.0001 oder tiefer einzustellen. Dieser Wert hilft dabei, wenn das Training mit dem ersten Wert nicht konvergiert und vermeidet so, dass das Training fehlschlägt.

### MinimalOccurrence

Auftreten: 1

Beschreibung: Das Element MinimalOccurrence dient als Parameter bei der Feature Extraction. Der Wert des Elements gibt an, wie oft ein Feature mindestens vorhanden sein muss, damit er berücksichtigt wird. Gerade bei Textklassifizierung ist dieser Wert wichtig, da eine grosse Anzahl an Worten möglich sind und nicht selten vorkommt, dass ein Wort nur ein Mal auftaucht. Der Parameter hilft so, diese Features zu entfernen und reduziert somit die Grösse der Vektoren, sowie auch die Laufzeit.

Erlaubte Werte für das Element:

- ein ganzzahliger Wert

Beispiel:

```

1 <MinimalOccurrence>2</MinimalOccurrence>

```

### Empfehlung

Ein Wert von 2 wird empfohlen. Bei dieser Einstellung werden alle Worte, die nur ein Mal auftauchen entfernt. Ist mit sehr vielen Trainingsdaten oder mit langen Texten zu rechnen, kann dieser Wert erhöht werden.

**NFrequentWordsToDelete**

Auftreten: 1

Beschreibung: Das Element NFrequentWordsToDelete dient als Parameter bei der Feature Extraction. Der Wert des Elements gibt an, wie viele der häufigsten Features entfernt werden sollen. Bei der Textklassifizierung macht dieser Parameter gerade deshalb Sinn, weil viele Verbundwörter existieren, die nichts dem Thema beisteuern. Weiterhin spielt auch die Problemdomäne eine Rolle. So ist das Wort "Bildschirm" bei Texten über "OLED" und "LCD" ein vielgebrauchter Begriff. Bei der Entscheidung des Klassenproblems hilft er aber nicht weiter

Erlaubte Werte für das Element:

- ein ganzzahliger Wert

Beispiel:

```
1 <NFrequentWordsToDelete>100</NFrequentWordsToDelete>
```

**7.6.1.0.1.1 Empfehlung** Ein Wert von 100 wird empfohlen. Bei dieser Einstellung werden 100 der häufigsten Worte entfernt. Ist mit sehr vielen Trainingsdaten oder mit langen Texten zu rechnen, kann dieser Wert erhöht werden.

**KCrossValidationType**

Auftreten: 1

Beschreibung: Das KCrossValidationType Element definiert, wie viele K-Cross Folds für die Validierung der Parameteroptimierung genutzt werden. Es wird dabei unter Sets fixer Grösse und eine Festlegung auf die Anzahl Sets unterschieden

Erlaubte Werte für das Element:

- NSets
- FixedSize

Beispiel:

```
1 <KCrossValidationType>NSets</KCrossValidationType>
```

**Empfehlung**

Es wird vorgeschlagen die Einstellung auf NSets vorzunehmen. Nachfolgend soll die Anzahl an Sets auf 10 eingestellt werden. 5 und 20 sind weiterhin gute Einstellungsmöglichkeiten.

Soll nicht auf die Laufzeit geachtet werden, kann auch FixedSize genommen werden und die Set-Grösse auf 1 eingestellt werden. Dies bedeutet aber, dass mit jedem TrainingsdatenElement validiert werden soll, was die Laufzeit extrem steigert.

**NSets**

Auftreten: 1

Beschreibung: Das NSets Element definiert wie viele Sets aus den Trainingsdaten für die K Cross Validation gebildet werden. Voraussetzung dass dieser Parameter beim Training berücksichtigt ist, ist dass das *KCrossValidationType*-Element den Wert *NSets* hat.

Erlaubte Werte für das Element:

- ein ganzzahliger Wert

Beispiel:

```
1 <NSets>10</NSets>
```

**Empfehlung** Es wird ein Wert von 10 empfohlen. 5 und 20 sind weitere gute Einstellungsmöglichkeiten.

**SetSize**

Auftreten: 1

Beschreibung: Das SetSize Element definiert wie gross die Sets für die K Cross Validation gebildet sind. Voraussetzung dass dieser Parameter beim Training berücksichtigt ist, ist dass das *KCrossValidationType*-Element den Wert *SetSize* hat.

Erlaubte Werte für das Element:

- ein ganzzahliger Wert

Beispiel:

```
1 <SetSize>1</SetSize>
```

**Empfehlung**

SetSize macht nur Sinn, wenn der Wert tief eingestellt wird. 1 und 2 sind dabei Werte, die optimal für die Validierung sind, aber die Laufzeit extrem ansteigen lassen. Andernfalls wird empfohlen auf NSets auszuweichen, da dafür nicht auf die Anzahl Trainingsdaten geachtet werden muss.

**EmailFieldConfigurationSetups**

Auftreten: 1

Beschreibung: Das Element EmailFieldConfiguraiontSetups kann mehrere EmailFieldConfigurationSetup enthalten. Das Element definiert die verschiedenen Möglichkeiten wie die verschiedenen Felder eines Emails Tokenisiert und Features gewichtet werden.

Kindelemente:

- EmailFieldConfigurationSetup

**Kindelement: EmailFieldConfigurationSetup**

Auftreten: 1 oder beliebig viele

Beschreibung: Das Kindelement EmailFieldConfigurationSetup steht für verschiedene Variationen der Gewichtung und der Tokenization einzelner E-Mail Felder.

Kindelemente:

- EmailFieldConfigurations

**Kindelement: EmailFieldConfigurations**

Auftreten: 1 oder beliebig viele

Beschreibung: Das Kindelement EmailFieldConfigurations kann aus mehreren EmailFieldConfiguration-Elementen bestehen.

Kindelemente:

- EmailFieldConfiguration

**Kindelement: EmailFieldConfiguration**

Auftreten: 1 oder beliebig viele

Beschreibung: Ein EmailFieldConfiguration-Element sagt aus, wie die Gewichtung der E-Mail Felder sein soll und wie deren Wörter extrahiert werden sollen.

Kindelemente:

- FieldType
- Weight
- TokenizationType

**Kindelement: FieldType**

Auftreten: 1

Beschreibung: Das FieldType-Element definiert, um was für eine Art von Feld es sich handelt.

Erlaubte Werte für das Element:

- Body: Steht für den eigentlichen Text des E-Mails
- Subject: Steht für die Betreffzeile des Mails
- From: Absender
- To: Empfängerliste

**Empfehlung**

Es lohnt sich, auf jeden Fall die Feldtypen "Body" und "Subject" einzustellen. Handelt es sich um eine öffentliche Adresse, macht es wenig Sinn, die Empfängerliste und den Absender einzustellen, da grundsätzlich immer die öffentliche Adresse als Empfänger angegeben wird und da die Absender von Mail zu Mail variieren.

**Kindelement: Weight**

Auftreten: 1

Beschreibung: Das Weight Element definiert, wie stark die Features der zugehörigen EmailFieldConfiguration gewichtet werden. Wird der Wert 0 angegeben, wird das E-Mail Feld nicht betrachtet.

Erlaubte Werte für das Element:

- ein ganzzahliger Wert grösser 0

**Empfehlung**

“Body“ und “Subject“ sollen ähnliche Gewichtung aufweisen. Von diesem Stand kann abgewiesen werden, wenn beispielsweise schon durch den E-Mail-Header erkannt werden kann, zu welcher Kategorie das Mail gehört.

Bei einer privaten Adresse, kann es von Vorteil sein, den Absender höher zu gewichten, wenn beispielsweise E-Mails der Kategorien meistens von denselben Absender stammen.

**Kindelement: TokenizationType**

Auftreten: 1

Beschreibung: Das TokenizationType Element sagt aus, wie das zugehörige Feld tokenisiert werden soll. Bei WhitespaceTokenization wird der Text anhand der Satzzeichen und der Whitespaces in Tokens aufgetrennt. Der StemmingTokenizer arbeitet mit einer Stopp-Liste und einem Stemming-Algorithmus. Erstere wird eingesetzt, um Wörter, die oft im normalen Sprachgebrauch verwendet werden, herauszufiltern. Beim Stemming wird zusätzlich versucht, die Wörter in eine Stammform zu bringen. Beide Vorgänge reduzieren die Anzahl Features eines Textes, setzen aber voraus, dass die Sprache erkannt werden muss.

Erlaubte Werte für das Element:

- StemmingTokenization
- WhitespaceTokenization

**Empfehlung** Für “Body“ und “Subject“ soll der StemmingTokenization Typ verwendet werden. Dies reduziert die Anzahl Features beträchtlich.

Bei den E-Mail Adresslisten macht es keinen Sinn ein Stemming zu betreiben, deshalb wird der WhitespaceTokenization Typ empfohlen.

Beispiel:

```

1 <EmailFieldConfigurationSetups>
  <EmailFieldConfigurationSetup>
3    <EmailFieldConfigurations>
      <EmailFieldConfiguration>
5        <FieldType>Body</FieldType>
          <Weight>1</Weight>
7        <TokenizationType>StemmingTokenization</TokenizationType>
      </EmailFieldConfiguration>
9    <EmailFieldConfiguration>
      <FieldType>Subject</FieldType>
11     <Weight>1</Weight>

```

```

13         <TokenizationType>StemmingTokenization</TokenizationType>
14     </EmailFieldConfiguration>
15 </EmailFieldConfigurations>
16 </EmailFieldConfigurationSetup>
17 <EmailFieldConfigurationSetup>
18     <EmailFieldConfigurations>
19         <EmailFieldConfiguration>
20             <FieldType>Body</FieldType>
21             <Weight>1</Weight>
22             <TokenizationType>StemmingTokenization</TokenizationType>
23         </EmailFieldConfiguration>
24     </EmailFieldConfigurations>
25 </EmailFieldConfigurationSetup>

```

### LinearKernels

Auftreten: 1

Beschreibung: Das LinearKernels-Element ist eine Auflistung von mehreren linearen Kernen. Jeder Kernel sollte unterschiedliche Parameter gesetzt haben. So kann überprüft werden, welcher lineare Kernel der beste für die Aufgabenstellung ist.

Kindelemente:

- LinearKernel

**Empfehlung** Es wird empfohlen nur einen Linearen Kernel mit einer Constant von 0 einzustellen. Weitere Kernel dieses Typs machen wenig Sinn, da die Constant keinen grossen Einfluss hat.

#### Kindelement: LinearKernel

Auftreten: 0 oder beliebig viele

Beschreibung: Das Kindelement LinearKernel steht für eine Instanz des Linearen Kernel.

Kindelemente:

- Constant

#### Kindelement: Constant

Auftreten: 1

Beschreibung: Das Constant-Element definiert den Constant Parameter für den Linearen Kernel.

Erlaubte Werte für das Element:

- ein ganzzahliger Wert
- eine Gleitkommazahl

Beispiel:

```

1 <LinearKernels>
  <LinearKernel>
3     <Constant>0</Constant>
  </LinearKernel>
5  <LinearKernel>
     <Constant>1</Constant>
7  </LinearKernel>
</LinearKernels>

```

### GaussianKernels

Auftreten: 1

Beschreibung: Das GaussianKernels-Element ist eine Auflistung von mehreren Gaussian Kerneln. Jeder Kernel sollte unterschiedliche Parameter gesetzt haben. So kann überprüft werden welcher Gaussian Kernel der beste für die Aufgabenstellung ist.

Kindelemente:

- GaussianKernel

#### Kindelement: GaussianKernel

Auftreten: 0 oder beliebig viele

Beschreibung: Das Kindelement GaussianKernel steht für eine Instanz des Gaussian Kernel.

Kindelemente:

- Sigma

#### Kindelement: Sigma

Auftreten: 1

Beschreibung: Das Sigma-Element definiert den Sigma Parameter für den Gaussian Kernel.

Erlaubte Werte für das Element:

- ein ganzzahliger Wert
- eine Gleitkommazahl

Beispiel:

```

<GaussianKernels>
2  <GaussianKernel>
     <Sigma>1</Sigma>
4  </GaussianKernel>
   <GaussianKernel>
6     <Sigma>2</Sigma>
   </GaussianKernel>
8 </GaussianKernels>

```



**ChiSquareKernels**

Auftreten: 1

Beschreibung: Das ChiSquareKernels-Element ist eine Auflistung von ChiSquare Kerneln.

Kindelemente:

- ChiSquareKernel

**Kindelement: ChiSquareKernel**

Auftreten: 0 oder 1

Beschreibung: Der ChiSquare Kernel besitzt keine eigenen Parameter. Soll der ChiSquare Kernel beim Training berücksichtigt werden so muss das Element ChiSquareKernel vorhanden sein. Soll der ChiSquare Kernel nicht verwendet werden muss das Element weggelassen werden.

Beispiel:

**<ChiSquareKernels>**

<sup>2</sup> **<ChiSquareKernel></ChiSquareKernel>**

**</ChiSquareKernels>**

**PolynomialKernels**

Auftreten: 1

Beschreibung: Das PolynomialKernels-Element ist eine Auflistung von Polynomial Kerneln.

Kindelemente:

- PolynomialKernel

**Empfehlung** Es wird empfohlen mehrere Polynomial Kernels einzusetzen. Deren Degree Werte sollen auf [1,2,3] eingestellt werden.

Wird zusätzlich auch ein linearer Kernel ausprobiert, kann der polynomiale Kernel mit Degree von 1 weggelassen werden, da dies zu demselben Resultat führt.

Der Constant Wert lohnt sich nicht, einzustellen.

**Kindelement: PolynomialKernel**

Auftreten: 0 oder beliebig viele

Beschreibung: Das PolynomialKernel Element definiert eine Instanz des PolynomialKernel mit entsprechend gesetzten Parameter.

Kindelemente:

- Constant
- Degree

**Kindelement: Constant**

Auftreten: 1

Beschreibung: Das Constant-Element definiert den Constant Parameter für den Polynomial Kernel.

Erlaubte Werte für das Element:

- ein ganzzahliger Wert
- eine Gleitkommazahl

**Kindelement: Degree**

Auftreten: 1

Beschreibung: Das Degree-Element definiert den Grad des Polynoms, das als Trennlinie zwischen den Klassen verwendet wird.

Erlaubte Werte für das Element:

- ein ganzzahliger Wert

Beispiel:

```

1 <PolynomialKernels>
  <PolynomialKernel>
3     <Constant>1</Constant>
     <Degree>1</Degree>
5 </PolynomialKernel>
  <PolynomialKernel>
7     <Constant>1</Constant>
     <Degree>2</Degree>
9 </PolynomialKernel>
</PolynomialKernels>

```

**7.6.2 Language Identifier**

Der Language Identifier ist so konfigurierbar, dass für jede Sprache, die er laut Anforderungen unterstützen kann, ein Wörterbuch (Dictionary) hinterlegt werden kann. Dies geschieht über das User Interface. Wird kein Dictionary für die Sprache hinterlegt, so ist es dem Language Identifier auch nicht möglich die Sprache zu erkennen. Auch hängt die Erfolgsrate für eine korrekte Spracherkennung vom verwendeten Dictionary ab. Wird ein Dictionary hinzugefügt, so wird, falls für die Sprache schon eins vorhanden ist, das alte gelöscht und durch das neue ersetzt. Dabei wird es in der Datenbank abgespeichert.

Für die Implementierung des Language Identifier wird die Library *NTextCat* verwendet. Die Library liefert für jede Sprache mehrere verschiedene Dictionaries mit, die verwendet werden können. Werden die Dictionaries für die Sprachen durch neuere ersetzt, so beeinflusst dies die Erfolgsrate der schon trainierten Classifiers. Es wird deshalb empfohlen, einen neuen Trainingsvorgang einzuleiten.

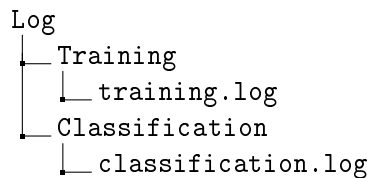
## 7.7 Logging

Abgeschlossene Trainings- und Klassifizierungsvorgänge werden in der Datenbank persistent gespeichert. Dabei wird in der Datenbank nur so viele Informationen gespeichert, dass alle Entscheidungsschritte der Applikation nachvollzogen werden können. Somit wird auch eine allfällige Auswertung vereinfacht.

Gleichzeitig werden alle Aktionen der Applikation in Logfiles abgelegt. Im Ausführungsordner befindet sich ein Unterordner mit dem Namen *Log*.

Detaillierte Informationen zum Trainingsvorgang werden unter *Log/Training/training.txt* gespeichert. Zu den Informationen zählen zum Beispiel die Fehlerraten der einzelnen Setups, sowie die Resultate der K-Fold Cross Validierung

Wenn ein Email klassifiziert werden soll, werden die Informationen zum Vorgang in der Datei *Log/Classification/classification.txt* festgehalten.

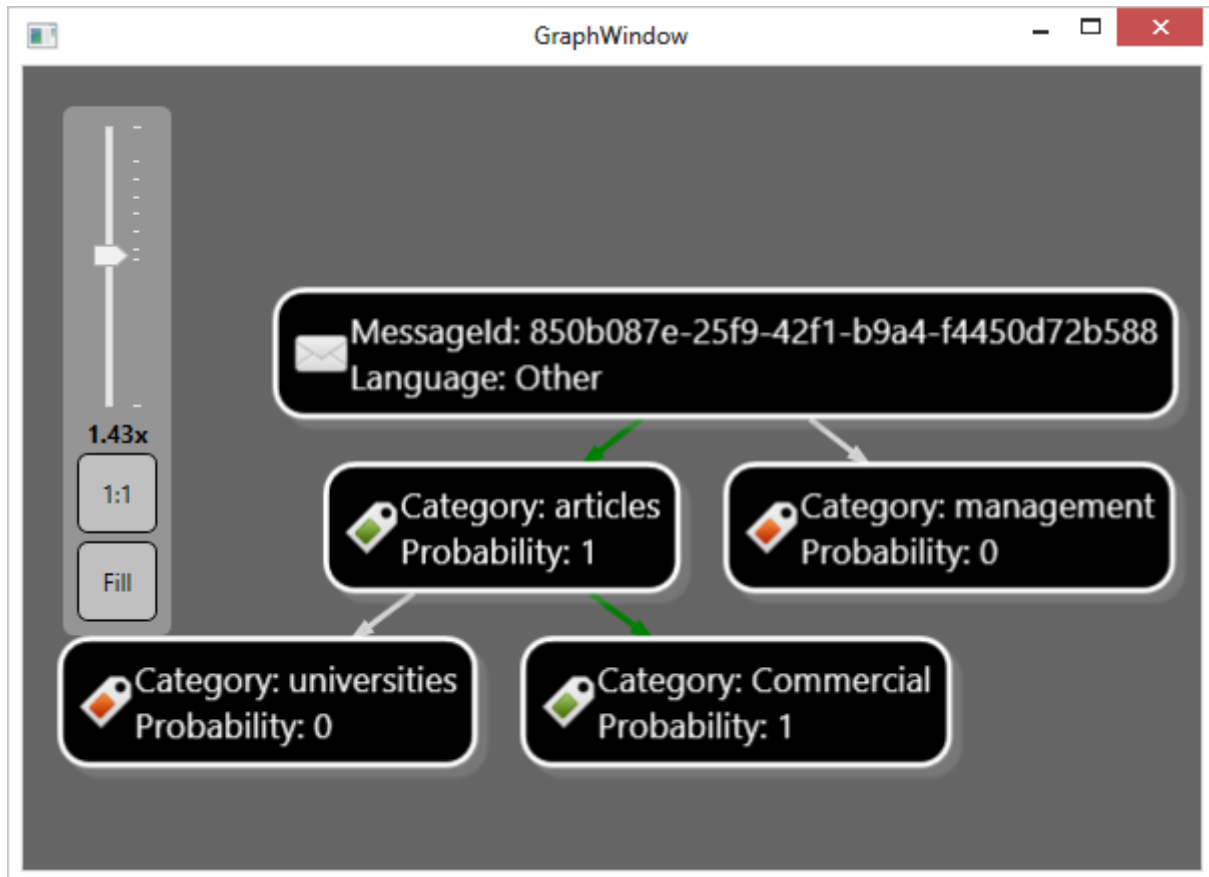


**Abbildung 7.55:** Logging File Struktur

Die Konfiguration des Loggings kann im XML-File *Log4NetConfig.xml* angepasst werden. Dieses File befindet sich in der Root des Projektes *EmailClassifier.Logging*.

## 7.8 Visualisierung des Resultats

Durch das Classifier Chaining können mehrere Klassifizierer für das Endresultat verantwortlich sein. Damit nachvollzogen werden kann, wie ein Klassifizierungsergebnis zustande kam, wird das Resultat als Baumstruktur visualisiert.



**Abbildung 7.56:** Visualisierung des Resultats

Der oberste Knoten ist immer das E-Mail mit dessen Messageld. Zusätzlich wird auch die erkannte Sprache, in der das Mail erfasst wurde, angezeigt.

Die unteren Knoten stellen die Kategorien dar, für die sich der Klassifizierer während des Vorgangs entscheiden musste. Die Kategorien mit einem grünen Icon sind diejenigen, die die grösste Wahrscheinlichkeit hatten und somit dem E-Mail zugeteilt wurden. Alle anderen Kategorien haben ein rotes Icon. Der Entscheidungsweg kann über die grünen Kanten nachvollzogen werden.

Der unterste Knoten im Baum mit der grössten Wahrscheinlichkeit, stellt die finale Entscheidung dar.

## 7.9 Testdokumentation

In diesem Teil der Dokumentation werden alle Tests sowie Resultate festgehalten.

### 7.9.1 Testumgebung

#### 7.9.1.1 Trainingsdaten

Damit die Testergebnisse gegeneinander verglichen werden können, müssen immer dieselben Trainingsdaten verwendet werden. Ausserdem soll die Aufteilung der Trainingsdaten in ein Training- und ein Testset immer gleich erfolgen.

##### 7.9.1.1.1 Datenset

Das Enron Datenset ist eine öffentliche Sammlung von E-Mails, die von 158 Personen aus dem Management der Firma Enron empfangen und gesendet wurden. Sie entstand aufgrund einer publiken Investigation eines Skandals, in der die Firma verwickelt war und der die Firma in den Ruin getrieben hat.

Die Sammlung dient heute vor allem für wissenschaftliche Untersuchung von E-Mail Klassifikatoren. Aus diesem Grund haben viele Anpassungen an ihr stattgefunden, um den Umgang mit ihr zu erleichtern. Für diese Arbeit wird die Sammlung von EnronData.org verwendet.

- Ungefähr 500 000 Mails
- Unterteilt auf 148 personelle E-Mail Accounts
- Eigens vorgenommene Klassifizierungen durch die Personen. Dazu wurden die E-Mails in unterschiedliche Verzeichnisse abgelegt
- Keine Attachments
- Anonymisierte Personen

##### 7.9.1.1.2 Wahl der Trainingsdaten

Vom Enron Datenset werden die Ordner der 7 grössten Mail-Accounts ausgewählt und für Testzwecke verwendet. Als Ausgangslage dient das Sub-Set der Sammlung von Ron Bekkerman [2], welches bereits folgendermassen bearbeitet wurde.

- Die Mail-Ordner werden in eine flache Hierarchie gebracht. Das heisst, dass Unterordner in einem Verzeichnis in die oberste Hierarchiestufe platziert werden
- Entfernung von Ordner, die ungeordnete Mails enthalten. Hierzu gehören beispielsweise der Posteingang oder der Papierkorb
- Entfernung von Ordnern, die weniger als 3 Mails enthalten. Diese bieten zu wenige Trainings-mails, um die Kategorisierung effektiv zu lernen.

Die Accounts haben folgende Eigenschaften:

**Tabelle 7.3:** Trainingsdaten Statistiken

Account	Anzahl Ordner	Anzahl E-Mails	Kleinste Anzahl Mails in einem Ordner	Grösste Anzahl Mails in einem Ordner
beck-s	101	1971	3	166
farmer-d	25	3672	5	1192
kaminski-v	41	4477	3	547
kitchen-l	47	4015	5	715
lokay-m	11	2489	6	1159
sanders-r	30	1188	4	420
williams-w3	18	2769	3	1398

### 7.9.1.1.3 Aufteilung der Trainingsdaten

Normalerweise werden beim Trainieren des Klassifizierers die Trainingsdaten zufällig in Training- und Testsets aufgeteilt. Dies scheint aber für das Testen problematisch, da der Zeitfaktor nicht einbezogen wird. So kann sich über den Lauf der Zeit der Fokus eines E-Mail Accounts verschieben, wenn beispielsweise die Person eine neue Arbeit annimmt. Um also eine möglichst authentische Testumgebung zu gestalten, werden die Mails nach Zeitpunkte des Empfangs sortiert.

Anschliessend wird ein inkrementelles Aufteilen der Trainingsdaten in Test- und Trainingssets durchgeführt. Dieses Verfahren wurde von Bekkerman et al. vorgeschlagen [2]. Aus dem Datenset werden die ersten N Daten für das Training verwendet und mit den nächsten N Daten getestet. Ist die Fehlerrate berechnet, werden 2N Daten für das Training verwendet und gegen die nächsten N Daten verglichen. Auf diese Weise wird fortgefahren bis man alle Mails des Datensets aufgeteilt hat. Enthalten die Testdaten eine Kategorie, welche beim Training noch nicht bekannt ist, dann werden jene Mails nicht zum Testen verwendet. N wird auf 100 Mails gesetzt. Zum Schluss werden die Fehlerraten der Testsplits zusammengerechnet und gemittelt.

### 7.9.1.1.4 Erläuterungen zu den Trainingsdaten

Da die verwendeten Trainingsdaten nur aus Mails von persönlichen E-Mail Accounts bestehen, ist es schwierig Rückschlüsse auf die Effizienz des Klassifizierers im Einsatz zu schliessen. Zum Einen unterscheiden sich die Mails, die an öffentlichen Adressen empfangen werden, von denen persönlicher Accounts. Zum Anderen fehlt der spezifische Kontext, in der der Klassifizierer zum Einsatz kommen soll. So sind die Mails fast ausschliesslich in Englisch geschrieben, weshalb die Sprachidentifizierung nicht getestet werden kann.

In folgenden Punkten könnte der Enron Datensatz besser an den Kontext eines öffentlichen Accounts herangeführt werden:

- Entfernung von Mails, die zu einer Diskussion gehören, da davon ausgegangen werden kann, dass der Mailverkehr nur zwischen zwei Parteien stattfindet. Ausserdem werden reply-to Mails schon von der Mailbox des Exchange Servers an den richtigen Empfänger weitergeleitet.
- Entfernung von Mails, die vom Besitzer des Mail-Accounts verfasst wurden. Auch hier spielt der zukünftige Verwendungszweck der Applikation eine Rolle, denn es sollen nur die Mails, die an die öffentliche Mailadresse gesendet wurden, sortiert werden.

## 7.9.2 Test Ende C1

### 7.9.2.1 Beschreibung

In diesem Test wird die erste Implementation des adaptiven Klassifizierers getestet. Es wird das zuvor erwähnte Verfahren angewendet, um die Trainingsdaten in Test- und Trainingssets aufzuteilen. Als Tokenizer wird ein einfacher Algorithmus eingesetzt, der Texte bei auftretenden Whitespaces trennt. Ausserdem werden die Wörter in Lowercase gespeichert, damit die Anzahl Features reduziert werden können. Der Complexity Parameter wird auf den Erfahrungswert von Bekkerman [2] gesetzt. Die WeightRatio wird zunächst auf 1 gesetzt, da noch in Erfahrung gebracht werden muss, welche Auswirkungen dieser Parameter auf die Testergebnisse hat.

<b>Testnummer</b>	1
<b>Preprocessing</b>	<ul style="list-style-type: none"> <li>• Whitespace Split</li> <li>• Kleinschreibung</li> <li>• Minimal Occurrence = 2</li> <li>• Nbr. of Most Frequent Words Deletions = 100</li> </ul>
<b>Klassifizierungsmechanismus</b>	SVM mit linearem Kernel, Complexity $c = 0.01$ , WeightRatio $j = 1$

### 7.9.2.2 Erwartung

Wandelt man die Wörter in Lowercase um, kann man die Anzahl Features zumindest verringern. Wie sich zeigen wird, wird jedoch immernoch eine Vielzahl von Features existieren. Dies wird sich auf die Geschwindigkeit des Trainingsvorgang auswirken, da die Komplexität mit der Anzahl an Features gekoppelt ist. Die Erfolgsraten bei den einzelnen Trainingsdatensätze werden sehr unterschiedlich ausfallen, da nicht alle Accounts die gleichen Qualitäten aufweisen.

### 7.9.2.3 Ergebnis

**Tabelle 7.4:** Testresultat: Test Ende C1

User Account	Dauer des Trainingsvorgang	Anzahl Features	Erfolgsrate
beck-s	-	9602	0.472
farmer-d	ConvergenceException bei Split 29 (Trainingsmails: 2900, TestMails: 100, Features: 10168)		
kaminski-v	ConvergenceException bei Split 27 (Trainingsmails: 2700, TestMails: 99, Features: 13592)		
kitchen-l	ConvergenceException bei Split 25 (Trainingsmails: 2500, TestMails: 100, Features: 13961)		
lokay-m	723	12882	0.762
sanders-r	89	10763	0.678
williams-w3	ConvergenceException bei Split 19 (Trainingsmails: 1900, TestMails: 100, Features: 7527)		

Dieser Test wurde am 01.04.2013 durchgeführt.

### 7.9.2.4 Interpretation

Bei vielen Accounts wird die Convergence Exception geworfen. Dies bedeutet, dass beim Trainieren zu viele Durchgänge für das Lösen des Optimierungsproblems benötigt werden und die Lösung somit nicht konvergiert. Ursache ist meistens, wenn eine zu hohe Komplexität verwendet wird oder wenn die Trainingsdaten nicht eindeutig den Kategorien zugewiesen wurden.

In einem nächsten Schritt sollte das Stemming ausprobiert werden. Dies sollte die Verbesserung bringen, dass die Feature Vektoren minimiert und somit Komplexität aus dem Klassifizierungsproblem genommen wird.

## 7.9.3 Test Ende C2

### 7.9.3.1 Beschreibung

Der Test, welcher Ende C1 durchgeführt wurde, wird nochmals wiederholt, um zu sehen ob sich Verbesserungen eingestellt haben.

In diesem Durchgang kommen das Stemming und die Stopp-Wörter Elimination zum Einsatz. Dies soll die Anzahl Features minimieren.



<b>Testnummer</b>	2
<b>Preprocessing</b>	<ul style="list-style-type: none"> <li>• Whitespace Split</li> <li>• Stop-Word Removal (Standard Liste von Lucene)</li> <li>• English Word Stemming</li> <li>• Kleinschreibung</li> <li>• Minimal Occurrence = 2</li> <li>• Nbr. of Most Frequent Words Deletions = 100</li> </ul>
<b>Klassifizierungsmechanismus</b>	SVM mit linearem Kernel, Complexity $c = 0.01$ , WeightRatio $j = 1$

### 7.9.3.2 Erwartung

Durch die Minimierung der Features, sollte die Komplexität des Mehrklassenproblems heruntersetzt werden können. Folglich wird erwartet, dass nun alle Lösungen konvergieren und keine Exception mehr geworfen wird.

### 7.9.3.3 Ergebnis

**Tabelle 7.5:** Testresultat: Test Ende C2

User Account	Dauer des Trainingsvorgang	Anzahl Features	Erfolgsrate
beck-s	384	7196	0.478
farmer-d	ConvergenceException bei Split 32 (Trainingsmails: 3200, TestMails: 97, Features: 9010)		
kaminski-v	ConvergenceException bei Split 29 (Trainingsmails: 2900, TestMails: 100, Features: 11048)		
kitchen-l	ConvergenceException bei Split 30 (Trainingsmails: 3000, TestMails: 100, Features: 11834)		
lokay-m	ConvergenceException bei Split 17 (Trainingsmails: 1700, TestMails: 100, Features: 8561)		
sanders-r	64	8072	0.681
williams-w3	ConvergenceException bei Split 24 (Trainingsmails: 2400, TestMails: 100, Features: 5957)		

Dieser Test wurde am 16.04.2013 durchgeführt.

### 7.9.3.4 Interpretation

Mithilfe des Stemming, konnte das Problem nicht behoben werden. Dennoch ist eine Verbesserung zu sehen, so dass nun die Exception zu einem späteren Zeitpunkt im Training geworfen wird.

Weiterhin ist ersichtlich, dass die Anzahl Features beispielsweise beim Account *beck\_s* von vorhergehenden 9602 auf 7196 minimiert werden konnte. Dies führte auch zu einer minimalen Verbesserung der Erfolgsrate.

## 7.9.4 Test Parameteroptimierung

### 7.9.4.1 Beschreibung

Um das Konvergierungsproblem in den Griff zu bekommen, wurde in einem nächsten Schritt Wert auf die Parameteroptimierung gelegt. Dazu werden mehrere Werte für die Parameter Weighratio und Kernel definiert. Anschliessend entstehen daraus mehrere Setups, von denen jedes ein Mal für das Training der Multiclass SVM verwendet wird.

Im Folgenden wird das Training so festgelegt

<b>Testnummer</b>	2
<b>Preprocessing</b>	<ul style="list-style-type: none"> <li>• Whitespace Split</li> <li>• Stop-Word Removal (Standard Liste von Lucene)</li> <li>• English Word Stemming</li> <li>• Kleinschreibung</li> <li>• Minimal Occurrence = 2</li> <li>• Nbr. of Most Frequent Words Deletions = 100</li> </ul>
<b>Klassifizierungsmechanismus</b>	<ul style="list-style-type: none"> <li>• Kernels <ul style="list-style-type: none"> <li>– Linear</li> <li>– Gaussian</li> </ul> </li> <li>• WeightRatio: 0.5, 1, 2</li> <li>• Complexity: 0.01</li> </ul>

### 7.9.4.2 Erwartung

Mit der Angabe von zusätzlichen Werten für die einzelnen Parameter, besteht die Möglichkeit, dass Setups existieren, die selbst die komplexen Systemen zum Konvergieren bringen. Es wird also erwartet, dass die Convergence Exception verhindert wird.

### 7.9.4.3 Ergebnis

**Tabelle 7.6:** Testresultat: Test Parameteroptimierung

User Account	Dauer des Trainingsvorgang [sek]	Bestes Setup	Erfolgsrate
beck-s	2467	Weightratio 1, Linear-Kernel	0.482
farmer-d	ConvergenceException bei allen Setups		
kaminski-v	ConvergenceException bei allen Setups		
kitchen-l	ConvergenceException bei allen Setups		
lokay-m	ConvergenceException bei allen Setups		
sanders-r	398	Weightratio 1, Linear-Kernel	0.678
williams-w3	ConvergenceException bei allen Setups		

Dieser Test wurde am 14.05.2013 durchgeführt.

### 7.9.4.4 Interpretation

Die Parameteroptimierung hat keine Verbesserung gebracht. Weiterhin wird die Convergence Exception bei den gleichen Accounts geworfen.

Bei den bisher funktionierenden Accounts *beck\_s* und *sanders\_s* wird in beiden Fällen die Weightratio von 1 und der lineare Kernel angewendet. Dies ist nicht das erwartete Ergebnis, weil erhofft wurde, dass unterschiedliche Setups verwendet werden.

## 7.9.5 Test Neuimplementierung der Parameteroptimierung

### 7.9.5.1 Beschreibung

Es wurde herausgefunden, dass die bisherige Implementierung der Parameteroptimierung nicht optimal umgesetzt wurde. Der Grund dafür ist, dass das ideale Setup für die gesamte Multiclass SVM ermittelt wird. Stattdessen ist es sinnvoller, dass die Parameteroptimierung auf jede Submaschine angewendet werden soll.

Aus diesem Grund ist auch erklärbar, warum im vorhergehenden Test immer dasselbe Setup gefunden wurde. So kann beispielsweise die Weightratio nicht für die gesamte Multiclass SVM definiert werden, da sie individuell für ein binäres Problem relevant ist.

<b>Testnummer</b>	4
<b>Preprocessing</b>	<ul style="list-style-type: none"> <li>• Whitespace Split</li> <li>• Stop-Word Removal (Standard Liste von Lucene)</li> <li>• English Word Stemming</li> <li>• Kleinschreibung</li> <li>• Minimal Occurrence = 2</li> <li>• Nbr. of Most Frequent Words Deletions = 100</li> </ul>
<b>Klassifizierungsmechanismus</b>	<ul style="list-style-type: none"> <li>• Kernels <ul style="list-style-type: none"> <li>– Linear</li> <li>– Gaussian</li> </ul> </li> <li>• WeightRatio: 0.5, 1, 2</li> <li>• Complexity: 0.01</li> </ul>

### 7.9.5.2 Erwartung

Das Optimieren von einzelnen Sub Maschinen soll zu besseren Ergebnissen führen und schlussendlich das Problem des Nicht-Konvergierens beheben.

### 7.9.5.3 Ergebnis

**Tabelle 7.7:** Testresultat: Test Neuimplementierung der Parameteroptimierung

User Account	Erfolgsrate
beck-s	OutOfMemoryException
farmer-d	OutOfMemoryException
kaminski-v	OutOfMemoryException
kitchen-l	OutOfMemoryException
lokay-m	0.71
sanders-r	OutOfMemoryException
williams-w3	OutOfMemoryException

Dieser Test wurde am 21.05.2013 durchgeführt.

### 7.9.5.4 Interpretation

Mit der Implementierung der neuen Form der Parameteroptimierung hat der Bedarf an Speicher zugenommen. Dies führt sogar so weit, dass bei fast allen Testaccounts eine OutOfMemory-Exception

geworfen wird.

Der Grund dafür ist der, dass nun die Parameteroptimierung der verschiedenen Sub SVMs parallel ablaufen. Im Gegensatz dazu, wurde in der vorhergehenden Implementierung jedes Setup seriell für eine Multiclass SVM ausprobiert.

Da für jede parallele Ausführung eines Sub Trainings ein Cache angelegt wird, um bisherige Berechnungen festzuhalten, steigt der Speicherbedarf gerade bei Mehrklassenproblemen mit vielen Klassen frappant an.

Als Ausnahme ist hier der *lokey - m* Account anzumerken, welcher mit 11 verschiedenen Kategorien weniger Sub Maschinen benötigt als die anderen Accounts.

Um dieses Problem zu umgehen, muss der Cache Size Parameter beim Training auf einen tieferen Wert gesetzt werden. Da dies im Moment noch nicht möglich ist, muss die Konfiguration umgebaut werden.

## 7.9.6 Test Neuimplementierung der Parameteroptimierung 2

### 7.9.6.1 Beschreibung

Es wurde die CacheSize Angabe in die Konfiguration genommen. Mit diesem Feature soll die OutOfMemory Exception verhindert werden. Somit wird erhofft, dass die Ausführung des Trainings diesmal erfolgreich sein wird.

<b>Testnummer</b>	4
<b>Preprocessing</b>	<ul style="list-style-type: none"> <li>• Whitespace Split</li> <li>• Stop-Word Removal (Standard Liste von Lucene)</li> <li>• English Word Stemming</li> <li>• Kleinschreibung</li> <li>• Minimal Occurrence = 2</li> <li>• Nbr. of Most Frequent Words Deletions = 100</li> </ul>
<b>Klassifizierungsmechanismus</b>	<ul style="list-style-type: none"> <li>• Kernels <ul style="list-style-type: none"> <li>– Linear</li> <li>– Gaussian</li> </ul> </li> <li>• WeightRatio: 0.5, 1, 2</li> <li>• Complexity: 0.01</li> </ul>

### 7.9.6.2 Erwartung

Mit der einstellbaren Cache Size kann der Speicherverbrauch kontrolliert und individuell für die Testumgebung angepasst werden. Ein tiefer Wert für die Cache-Grösse wird dazu führen, dass keine Speicherknappheit mehr möglich wird.

### 7.9.6.3 Ergebnis

**Tabelle 7.8:** Testresultat: Test Neuimplementierung der Parameteroptimierung 2

User Account	Erfolgsrate
beck-s	0.53
farmer-d	0.75
kaminski-v	0.54
kitchen-l	0.48
lokay-m	0.72
sanders-r	0.71
williams-w3	0.88

Dieser Test wurde am 24.05.2013 durchgeführt.

### 7.9.6.4 Interpretation

Mit der Neuimplementierung der Parameteroptimierung ist es gelungen, das Problem mit der Nicht-Konvergenz zu einem grossen Teil in den Griff zu bekommen.

Auch die Speichernutzung ist kein Thema mehr. So konnte die Speicherknappheit verhindert werden.

## 7.9.7 Systemtests

### 7.9.7.1 Konfigurationseinstellungen

In diesem Abschnitt werden für die einzelnen Trainingskonfigurationen verschiedene Werte ausprobiert, die innerhalb und ausserhalb des Einstellungsbereich liegen. Da die Konfiguration über ein XML-gemacht wird, kann ein Nutzer nicht daran gehindert werden, Falscheinstellungen vorzunehmen. Deshalb ist ein ExceptionHandling notwendig.

Folgende Default Einstellung wurde gewählt. Anschliessend werden einzelne Parameter von dieser Einstellung abweichend festgelegt.

Einstellung	Wert
Trainingsdaten	Set 1: farmer-d, 106 enron-news Mails, Set 2: farmer-d 102 entex Mails
MinimalOccurrence	2
NFrequentWordsToDelete	100
WeightRatios	0.5, 1, 2
Complexities	0.01
KCrossValidationType	NSets
NSets	5
EmailFieldConfigurationSetups	[(Feld: Body, Gewicht: 1)], [(Feld: Subject, Gewicht: 1)]
Kernel	[Linear(Constant: 1), Polynomial(Constant: 1, Degree: 2)]
CachSize	200

### 7.9.7.2 Stand: 20.05.13

#### Test 1: Standardkonfiguration

Testspezifikation	Erwartung		Ergebnis
Training mit Standard Einstellung	Das Training ist erfolgreich	OK	2607 Words
Konfiguration wird während dem Training geändert	Das Training läuft weiter. Neue Konfiguration wird auf das nächste Training angewendet	OK	-

#### Test 2: Negative Minimal Occurrence

Testspezifikation	Erwartung		Ergebnis
Minimal Occurrence auf -1	Der Eingabewert wird abgefangen und der Nutzer wird darauf hingewiesen	Nicht OK	Das Training wird trotz Fehleinstellung ausgeführt und beendet erfolgreich (Successrate 90.44%, 5502 Wörter in Bag of Words)

**Interpretation** Bei der Erstellung des Bag Of Words, wird geprüft, ob ein Wort weniger als Minimal Occurrence vorkommt. Bei einem negativen Wert wird also jedes Wort in den Bag of Words aufgenommen, deshalb funktioniert auch das Training. Mit einem Wert von 5502 Wörtern ist es wesentlich grösser als im ursprünglichen Training 2607

#### Test 3: Minimal Occurrence höher als Wörter in Trainingsdaten

Testspezifikation	Erwartung		Ergebnis
Minimal Occurrence auf 50000	Der Bag of Words enthält keine Wörter. Das Training kann mit leeren Featurevektoren nicht durchgeführt werden. Das Training schlägt fehl und wird im GUI als Failed markiert	Nicht OK	Das Training wird trotz Fehleinstellung ausgeführt und beendet erfolgreich (Successrate 55.33%, 0 Wörter in Bag of Words)

**Interpretation** Obwohl keine Wörter bekannt sind, funktioniert der Trainingsvorgang. Dieses Resultat überrascht, da ohne Features gar nicht unter den Dokumenten entschieden werden kann. Wie an der Erfolgsrate ersichtlich scheint es jedoch so, dass willkürlich versucht wird, die Trainingsdaten zu trennen. Da beide Trainingssets ähnlich gross sind wird somit immer ungefähr eine Erfolgsrate von 50% erreicht

#### Test 4: Negative NFrequentWordsToDelete

Testspezifikation	Erwartung		Ergebnis
NFrequent WordsToDelete auf -1	Der Eingabewert wird abgefangen und der Nutzer wird darauf hingewiesen	Nicht OK	Das Training wird trotz Fehleinstellung ausgeführt und beendet erfolgreich (Successrate 97.61%, 2707 Wörter in Bag of Words)

**Interpretation** Bei der Erstellung des Bag Of Words, werden die Wörter nach ihrer Häufigkeit sortiert und die ersten Wörter bis zum NFrequentWordsToDelete Limit entfernt. Ist der Wert somit negativ, bricht die Löschung schon zu Beginn ab, weshalb kein Wort entfernt wird. Dies ist auch daran ersichtlich, dass im Vergleich zum ersten Training, bei dem NFrequentWordsToDelete auf 100 gesetzt wurde, 100 Wörter mehr im Dictionary vorhanden sind.

#### Test 5: NFrequentWordsToDelete höher als Anzahl Worte in Trainingsdaten

Testspezifikation	Erwartung		Ergebnis
NFrequent WordsToDelete auf 50000	Der Eingabewert wird abgefangen und der Nutzer wird darauf hingewiesen	Nicht OK	Das Training wird trotz Fehleinstellung ausgeführt und beendet erfolgreich (Successrate 51.44%, 0 Wörter in Bag of Words)

**Interpretation** Wie schon bei der Minimal Occurrence werden 0 Wörter in den Bag of Words eingefügt. Da somit keine Feature Selection betrieben werden kann. Ist die Erfolgsrate wiederum nahe 50%.



**Test 6: Cross Validation mit negativer Anzahl Sets**

Testspezifikation	Erwartung		Ergebnis
NSets auf -1	Eine negative Cross Validation Grösse, führt dazu, dass keine Cross Validation durchgeführt werden kann. Das Training soll gar nicht gestartet werden sondern vorher den Benutzer auf die Fehleingabe hinweisen	Nicht OK	IndexOutOfRangeException. Message: Index was outside the bounds of the array, Source: CrossValidation Constructor

**Interpretation** Die Falscheingabe wird nicht abgefangen. Das Programm übergibt den Wert dem CrossValidation Konstruktor, der anschliessend die Eingabe prüft und eine Exception wirft.

**Test 7: Cross Validation mit 1 Fold**

Testspezifikation	Erwartung		Ergebnis
NSets auf 1	Eine 1-Fold Validation führt dazu, dass gar kein Validationsset erstellt werden kann. Deshalb ist es nicht möglich den Klassifizierer zu testen. Der Trainingsvorgang soll deshalb gar nicht gestartet werden	Nicht OK	IndexOutOfRangeException. Message: Index was outside the bounds of the array, Source: CrossValidation Constructor

**Interpretation** Die Falscheingabe wird nicht abgefangen. Das Programm übergibt den Wert dem CrossValidation Konstruktor, der anschliessend die Eingabe prüft und eine Exception wirft.

**Test 8: Cross Validation mit mehr Folds als Trainingsdaten**

Testspezifikation	Erwartung		Ergebnis
NSets auf 300	Ist die Fold Anzahl grösser als die Anzahl Trainingsdaten, ist es nicht möglich die Folds zu erstellen, da zu wenig Daten zur Verfügung stehen. Der Trainingsvorgang soll deshalb gar nicht gestartet werden	Nicht OK	ArgumentException. Message: The number of folds can not exceed the total number of samples in the data set, Source: CrossValidation Constructor

**Interpretation** Die Falscheingabe wird nicht abgefangen. Das Programm übergibt den Wert dem CrossValidation Konstruktor, der anschliessend die Eingabe prüft und eine Exception wirft.

**Fazit zum Test** Die Konfiguration wird im Moment nicht auf ihre Richtigkeit geprüft. Ausschlaggebend dafür ist das nicht Vorhandensein eines Konfigurations-UI. Stattdessen wird im Moment die Einstellungen über ein XML angeboten. Dies führt dazu, dass Fehler nicht schon bei der Eingabe abgefangen werden können. Wird wiederum ein UI angeboten, wäre es möglich Validierungen einzubauen und so nicht zulässige Werte zu unterbinden.

### 7.9.7.3 Stand: 31.05.2013

Es wurde eine Validierung eingebaut, die die Konfigurationsfehler abfangen kann.

Testspezifikation	Erwartung		Ergebnis
NSets 208	Bei 208 Trainingsdaten soll die Eingabe akzeptiert werden	OK	-
NSets 209	Bei 208 Trainingsdaten können nicht mehr Folds erstellt werden. Es wird eine Fehlermeldung ausgegeben	OK	-
NSets 1	Ein 1-Fold führt dazu, dass keine Validationsdaten zur Verfügung steht. Deshalb soll eine Fehlermeldung ausgegeben werden	OK	-
NSets 0	Es müssen mindestens 2 Sets fesgelegt werden. Deshalb soll eine Fehlermeldung ausgegeben werden	OK	-
NSets -1	Es kann nicht eine negative Anzahl Sets angegeben werden. Deshalb soll eine Fehlermeldung ausgegeben werden	OK	-
SetSize 208	Bei 208 Trainingsdaten entspricht eine SetSize von 208 einer Fold-Grösse, die keine Validationsdaten zulässt. Es soll eine Fehlermeldung ausgegeben werden	OK	-
SetSize 209	Bei 208 Trainingsdaten kann die SetSize nicht grösser als 208 sein. Es wird eine Fehlermeldung ausgegeben	OK	-

SetSize 1	Eine Grösse von 1 entspricht exakt einem 208-Fold für die 208 Datenelemente. Die Eingabe wird akzeptiert	OK	-
SetSize 0	Es kann nicht eine Set Grösse von kleiner gleich 0 eingestellt werden. Es wird eine Fehlermeldung ausgegeben	OK	-
SetSize -1	Es kann nicht eine Set Grösse von kleiner gleich 0 eingestellt werden. Es wird eine Fehlermeldung ausgegeben	OK	-
WeightRatio: [0.1, 0.01]	Der Werte werden akzeptiert und ein Training kann gestartet werden	OK	-
WeightRatio: [0.1, 0.0]	Der Wert 0 wird nicht akzeptiert, da die WeightRatio nicht kleiner oder gleich 0 sein darf. Die Falschein-gabe wird dem Benutzer gemeldet	OK	-
WeightRatio: []	Der Vorgang wird abgebrochen und es wird gemeldet, dass keine WeightRatio angegeben wurde	OK	-
Complexity: [0.1, 0.01]	Der Werte werden akzeptiert und ein Training kann gestartet werden	OK	-
Complexity: [0.1, 0.0]	Der Wert 0 wird nicht akzeptiert, da die Complexity nicht kleiner oder gleich 0 sein darf. Die Falschein-gabe wird dem Benutzer gemeldet	OK	-
WeightRatio: []	Der Vorgang wird abgebrochen und es wird gemeldet, dass keine Complexity angegeben wurde	OK	-
CacheSize 100	Der Werte werden akzeptiert und ein Training kann gestartet werden	OK	-
CacheSize 0	Der Wert 0 wird nicht akzeptiert, da die CacheSize immer über 0 sein muss. Die Falschein-gabe wird erkannt und der Benutzer darauf hingewiesen	OK	-

CacheSize wird nicht angegeben	Es wird erkannt, dass die CacheSize nicht deklariert wurde. Dies wird am Benutzer mitgeteilt	Nicht OK	Es wird gemeldet, dass die CacheSize nicht 0 sein darf. Dies hat zur Bedeutung, dass das XML-File richtig geladen werden konnte, auch wenn ein Property nicht festgelegt wurde. Gleichzeitig wird aber auch die CacheSize auf den Default-Wert von 0 gesetzt, der nicht akzeptiert wird
Kernels: [Polynomial(Degree: -1 Constant: nicht angegeben)]	Der Polynomiale Kernel wird geladen, obwohl der Constant Wert nicht angegeben wurde	OK	-
Kernels: [Polynomial(Degree: nicht angegeben Constant: 1)]	Der Kernel kann nicht initialisiert werden, da kein Degree Wert angegeben wurde. Dies wird an den Benutzer gemeldet	Nicht OK	Das Training startet, auch wenn der Wert nicht angegeben wurde. Ohne Angabe des Degree Wertes, wird der Default-Wert von 0 verwendet. Dieser Wert ist unerwünscht, da er zu sehr schlechten Resultaten führt.
Das Tag "Polynomial-Kernels" wird nicht deklariert	Solange weitere Kernel deklariert wurde, wird das Auslassen des Polynomialen Kernel nicht abgefangen und das Training kann gestartet werden	OK	-
Es wird kein Kernel angegeben	Die Applikation meldet, dass keine Kernel angegeben wurden und das Training wird nicht gestartet	OK	-
EmailFieldConfiguration-Setups: Der Tag wird weggelassen	Die Applikation meldet, dass keine Einstellung vorhanden ist und der Trainingsvorgang wird nicht gestartet	OK	-
EmailFieldConfiguration-Setups: Es wird keine Configuration angegeben	Die Applikation meldet, dass keine Einstellung vorhanden ist und der Trainingsvorgang wird nicht gestartet	OK	-
EmailFieldConfiguration-			

Setups: Zweimalige Gewichtung von Field "Body"		Da die Einträge in ein Dictionary gespeichert werden, mit Field als Schlüssel, wird der Wert abgefangen, da ansonsten mehrere Schlüssel präsent wären. Der Trainingsvorgang wird nicht gestartet.	OK	-
Allgemein: CacheSize-Tag "CachSize" genannt	Das wird	Das XML kann nicht geparkt werden. Dieser Umstand wird an den Nutzer gemeldet und die Zeilennummer angegeben	Nicht OK	Das XML wird Falschein-gabe geparkt. Es wird gemeldet, dass die CacheSize nicht 0 sein darf. Es scheint also so, dass Schreibfehler überlesen werden. Die Fehlermeldung ist aber fehlerhaft.
Allgemein: CacheSize-Tag kein End-Tag	Das besitzt	Das XML kann nicht geparkt werden. Dieser Umstand wird an den Nutzer gemeldet und die Zeilennummer angegeben	Nicht OK	Exception: InvalidOperationException, Message: There is an Error in document (5,4). Der Programmabsturz konnte nicht verhindert werden. Es muss also zusätzlich ein Exception-Handling für XML Parsefehler realisiert werden

**Fazit zum Test** Viele Konfigurationsfehler konnten abgefangen werden. Ausserdem scheint der Parser sehr tolerant zu sein, wenn Tags nicht angegeben wurden. In diesen Fällen wird ein Default-Value festgelegt. Dieser Umstand wird aber zum Nachteil, weil der Wert einer Überprüfung unterzogen wird. Da der Default-Value meistens keine erlaubte Eingabe ist, wird eine unbrauchbare Fehlermeldung ausgegeben.

Allgemein ist es fraglich, ob nicht mit Standard-Werten gearbeitet werden sollte. So könnte zum Beispiel die Cache-Grösse immer auf 200 gesetzt werden, wenn keine Angaben gemacht wurden. Dies gilt als reine Vorsichtsmassnahme, damit keine OutOfMemoryException provoziert wird.

Gleichzeitig scheint die Nutzung von Standard-Eingaben jedoch sinnlos, wenn mit einem Configuration-File gearbeitet wird. Dieses wird zu Beginn einmal definiert und verändert sich wohl über eine längere Zeit nicht. Es ist also wichtig, ein vorgefertigtes Config-File auszuliefern, das ausbalancierte Einstellungen trifft und überall eingesetzt werden kann.

## Test 9: Trainingsdaten Import

Testspezifikation	Erwartung		Ergebnis
Trainingsdaten ins Konfigurationsfenster laden	Anzahl neuer Trainingsdaten wird angezeigt	OK	-
Beim laden neuer Trainingsdaten Prozess abbrechen	Vorgang wird abgebrochen	OK	-
Trainingsdaten in Datenbank speichern	Anzahl gespeicherter Trainingsdaten wird angezeigt	OK	-
Beim Speichern neuer Trainingsdaten in die Datenbank Prozess abbrechen	Vorgang wird abgebrochen	OK	-
Trainingsdaten löschen	Anzahl neuer Trainingsdaten wird angezeigt	OK	-

### Test 10: Kategorie

Testspezifikation	Erwartung		Ergebnis
Kategorie eines bereits trainierten Klassifizierers umbenennen	Bei erneuter Verwendung des Klassifizierers wird die umbenannte Kategorie angezeigt	OK	-
Kategorie eines bereits trainierten Klassifizierers löschen	Bei erneuter Verwendung des Klassifizierers wird die umbenannte Kategorie angezeigt	OK	-

### 7.9.8 Language Identifier Performancetest

Um den Language Identifier zu testen, wurde ein mehrsprachiges Datenset bestehend aus Tweets verwendet. Da ein Tweet aus maximal 160 Zeichen bestehen darf, wird es schwieriger für den Language Identifier auf gute Resultate zu kommen als wenn man Emails zum Testen verwenden würde. Der Hauptgrund für die Verwendung von Tweets anstatt Emails für das Testing ist der, dass kein mehrsprachiges Emaildatenset zur Verfügung stand, welches hätte verwendet werden können. Konkret wurde ein Teil des LIGA Benelearn 11 Datasets verwendet. In diesem Dataset sind die Tweets schon nach Sprache aussortiert. Für die Tests wurden ca. 1500 Tweets für die unterstützten Sprachen Deutsch, Englisch, Französisch und Italienisch verwendet sowie auch rund 1500 Spanische Tweets. Als erfolgreich Klassifiziert gilt wenn der Language Identifier die Sprache korrekt zuordnen kann oder wenn er sagt dass er die Sprache nicht erkennen konnte.

**Tabelle 7.10:** Testresultat: Language Identifier Performancetest

Sprache	Anzahl Tweets	Anzahl Richtige Zuweisungen	Anzahl Falschzuweisungen
Englisch	1505	1505	0
Deutsch	1479	1471	8
Französisch	1551	1448	3
Italienisch	1539	1539	0
Spanisch	1562	1367	195

### 7.9.9 Performance Test

Ziel des Performance Tests ist es, unter möglichst realen Bedingungen die Performance, sprich CPU sowie auch RAM Auslastung zu beobachten und festzuhalten. Folgende Konfiguration wurde verwendet:

- Whitespace Split
- Gross- Kleinschreibung
- Stop-Word Removal (Standard Liste von Lucene)
- English Word Stemming
- Kleinschreibung
- Minimal Occurrence = 2
- Nbr. of Most Frequent Words Deletions = 100

Dabei wurden neun Kategorien Vorgegeben mit insgesamt 5100 Mails.

#### 7.9.9.1 Resultat

Die Anzahl Features beträgt 17357. Die CPU Auslastung bewegte sich während des gesamten Vorgangs zwischen 50% - 100% und die Applikation benötigte dabei ca. 1,5GB RAM. Der hohe Memory verbrauch kommt vor allem durch die hohe Anzahl an Features und Mails zustande. Ein Feature Vektor wird in der Applikation über ein Double Array repräsentiert, wobei jedes Feature eine Wert im Array einnimmt, deshalb ist der Speicherverbrauch direkt Abhängig von der Anzahl an extrahierten Features sowie auch der Anzahl Mails. Damit die SVM ihre Hyperebene berechnen kann muss sie alle Vektoren ins RAM laden.

### 7.9.10 Unit Tests

Für das Projekt wurden total 123 Unit tests geschrieben. Die Testabdeckung sieht wie folgt aus:

Projekt	Testabdeckung
EmailClassifier.FeatureExtraction	83.44%
EmailClassifier.Core	81%
EmailClassifier.MachineLearning	80%
EmailClassifier.MailImport	63.5%
EmailClassifier.Logging	60.42%
EmailClassifier.LanguagelIdentifier	48.8%

**Tabelle 7.11:** Unit Test Abdeckung



## 8 Weiterführende Arbeiten

Obwohl das Projekt zum jetzigen Zeitpunkt auf einem guten Stand ist, sowie eine solide Implementation eines Textklassifizierers erreicht wurde, gibt es Verbesserungen, die noch angebracht werden könnten.

Nachfolgend sind potentielle weiterführende Arbeiten dokumentiert. Diese weiterführende Arbeiten haben das Ziel, die Software weiter zu verbessern und mit neuen Features anzureichern.

### 8.1 Regelbasierter Klassifizierer

Die Klassifikation kann zusätzlich mit einem regelbasierten Klassifizierer ergänzt werden. Die Idee dabei ist, dass manuell Regeln definiert werden können, die auf bestimmte Absender oder Schlüsselwörter greifen. Beispielsweise wäre es möglich, eine Regel festzulegen, die E-Mails mit einem bestimmten Absender direkt einer vordefinierten Kategorie zugewiesen werden sollen.

Falls eine Art von E-Mail immer wieder falsch klassifiziert wird, könnte über ein solches Feature Falschklassifizierungen kurzzeitig verhindert werden. Dies bietet sich gerade an, da für ein erneutes Training zuerst Daten vorhanden sein müssten, die den Fall abdecken. Weiterhin erlaubt dies mehr Kontrolle bei der Bestimmung der Kategoriezugehörigkeit, wenn der Anwender der Software beispielsweise bereits eigene Schlüsselwörter herausgesucht hat.

### 8.2 Anbindung an ein Mailsystem

Es müsste eine Anbindung an ein Mailsystem entwickelt werden, damit eingehende Nachrichten in Echtzeit klassifiziert werden können. Klassifizierte Mails könnten dann basierend auf der zugewiesenen Kategorie und ermittelten Sprachen in einen entsprechenden Ordner verschoben werden.

Dabei spielt es keine Rolle, um was für ein Mailsystem es sich dabei handelt. Einzig müsste ein Parser geschrieben werden, der den E-Mail Typ des Mailsystems in ein Dto-Objekt wandeln könnte. Falls es jedoch im MIME-Format aufliegt, könnte auch die Umsetzung im Projekt wiederverwendet werden.

### 8.3 Genauere Informationen zum trainierten Klassifizierer

Im Moment zeigt ein Klassifizierer nach seinem Training nur seine Erfolgsrate an. Es wäre aber auch möglich zusätzliche Informationen zum Training herauszugeben. Zu diesen Informationen gehören beispielsweise die ermittelten Parameter jeder Sub SVM und deren Validierungsschritte.

## 8.4 Neutrainieren eines Klassifizierers

Wenn, wie oben beschrieben, die ermittelten Parameter aus der Multiclass Support Vector Machine ausgelesen werden, könnten diese auch für einen erneuten Trainingsvorgang verwendet werden. Auf diese Weise kann die Parameteroptimierung umgangen werden, da das beste Setup schon zuvor ermittelt wurde. Somit würde die Dauer des Trainings um ein Vielfaches verkürzt werden. Dies bietet sich in folgenden Situationen an, bei denen nur wenige neue Trainingsdaten zu den Kategorien hinzugefügt wurden und ein Update des Klassifizierers gefordert wird.

Kommen jedoch viele neue Trainingsdaten dazu, wäre es besser, ein komplett neues Training zu starten.

## 8.5 User Interface

Beim Entwickeln des User Interface wurde viel Wert auf Funktionalität gelegt. Es wäre sinnvoll das User Interface auf Benutzerfreundlichkeit zu testen und eventuell zu überarbeiten. Alle Beschreibungen im User Interface sind auf Englisch und die Zeit wird in UTC angezeigt. Eine Lokalisierung des User Interfaces wäre sicher sinnvoll.

## 8.6 Konfigurationsfenster für das `configuration.xml`

Aktuell werden Parameter, die für das Training verwendet werden, im `configuration.xml` definiert. Hierfür könnte man ein separates User Interface entwickeln, um diese Parameter festlegen zu können. Dieses User Interface sollte jedoch nicht direkt dem Benutzer angezeigt werden sondern beispielsweise als erweiterte Optionen für fortgeschrittene Benutzer.

## 8.7 Zusätzliche Trainingsparameter

Für den Trainingsvorgang könnten noch weitere Parameter unterstützt werden. Für die Parameter *Epsilon* und *Tolerance* werden Standardwerte verwendet. Diese Parameter sind nicht essentiell für das Trainieren eines erfolgreichen Klassifizierers, jedoch könnte deren Anpassungen eine Leistungssteigerung erbringen.

## 8.8 Generalisierung des adaptiven Klassifizierers

Die Architektur der Applikation ist sehr modular aufgebaut. So werden beispielsweise Abhängigkeiten auf externe Libraries über Interfaces und eigene Projekte gekapselt. Im Moment ist die Architektur aber noch sehr auf die Verwendung von SVMs als adaptiven Klassifizierer ausgelegt. Dies könnte in einer Weiterentwicklung generalisiert werden, so dass auch andere Implementation wie zum Beispiel ein neuronales Netz oder naiver Bayes zugelassen werden.

## 8.9 Logging und Unit Testing

Das Logging müsste besser für das Unit Testing designed werden. Im Moment wird im Logmanager per Präprozessor Anweisung überprüft, ob es sich um ein Unit Test handelt oder nicht. Falls es sich um kein Unit Test handelt, werden die Einstellung aus dem *Log4NetConfig.xml* gelesen. Andernfalls werden Standard-Einstellungen verwendet, die kein Log-File erstellen.

# 9 Projektmanagement

Dieses Kapitel beschreibt, nach welchem Arbeitsprozess vorgegangen und welche Verwaltungstools verwendet wurden

## 9.1 Zweck und Ziel des Projektes

Im Rahmen der Bachelorarbeit soll in Zusammenarbeit mit der Luware AG ein E-Mail Klassifizierer entwickelt werden.

Üblicherweise geben Firmen der Öffentlichkeit nur eine E-Mail Adresse wie zum Beispiel *info@XYZ.ch* bekannt. Kunden der Firma schreiben deshalb bei verschiedensten Anliegen an dieselbe Adresse. So kann es sein, dass ein Kunde nach Support erfragt, während ein anderer sich über ein Produkt beschweren möchte. Dieser Umstand erschwert die Bearbeitung der Mails, da das Anliegen des Kunden den Kompetenzbereich des E-Mail Empfängers überschreiten kann und deshalb an eine andere Stelle weitergeleitet werden muss. Damit dieses Problem entschärft wird, soll in dieser Arbeit eine Lösung gefunden werden, wie E-Mails in erster Instanz automatisch vorsortiert werden können. Dies soll auf Basis eines Klassifizierers geschehen, der den Text eines Mails analysieren und einer Kategorie von ähnlichen Nachrichten zuordnen kann.

Unterstützte Sprachen sind in erster Linie Deutsch und Englisch. Danach auch Italienisch und Französisch. Die Software muss völlig autonom funktionieren und darf keine online Services zum Verrichten der Arbeit verwenden, da es sich bei E-Mails um vertrauliche Daten handelt.

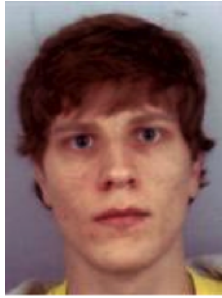
## 9.2 Lieferumfang

Es wird ein Prototyp eines Textklassifizierers entwickelt, der über ein UI bedient werden kann.

## 9.3 Projektorganisation

Das Projektteam besteht aus zwei Mitgliedern und einem externen Partner.

### 9.3.1 Organisationsstruktur



**Fabian Senn**



**Cyrill Lam**

Beide Teammitglieder sind gleichwertig an diesem Projekt beteiligt und tragen die gleiche Verantwortung. Die Tätigkeiten im Projekt werden nach Absprache untereinander aufgeteilt.

### 9.3.2 Externe Schnittstellen

Betreuer dieser Arbeit ist Rolf Schärer von der HSR.

Externer Partner ist die Luware AG. Ansprechperson und gleichzeitig auch Experte ist Michael Jakob.

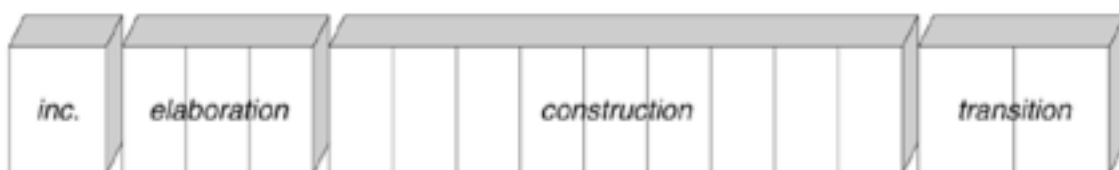
## 9.4 Planung

### 9.4.1 Zeitbudget

Die Bachelorarbeit wird mit 12 ECTS Credits gewichtet wobei ein ECTS Credit 30 Stunden Aufwand im Semester entspricht. Das entspricht einem Aufwand von 360 Stunden pro Person und somit 720 Stunden für das gesamte Projekt. Das Projekt startet am 18. Februar 2013 und endet am 14. Juni 2013.

### 9.4.2 Zeitliche Planung

In diesem Projekt wird nach dem Rational Unified Process (RUP) vorgegangen. Das Projekt wird dabei in vier Phasen geteilt: Inception, Elaboration, Construction und Transition.



**Abbildung 9.1:** Phasen nach RUP

Iteration	Milestone	Dauer
Inception	MS1: Inception	18.02.2013 - 26.02.2013
Elaboration E1		27.02.2013 - 05.03.2013
Elaboration E2	MS2: Elaboration	06.03.2013 - 19.03.2013
Construction C1	MS3: Construction1	20.03.2013 - 02.04.2013
Construction C2		03.04.2013 - 16.04.2013
Construction C3	MS4: Construction2	17.04.2013 - 30.04.2013
Construction C4	MS5: Construction3	01.05.2013 - 14.05.2013
Construction C5	MS6: Construction4	25.05.2013 - 21.05.2013
Transition T1		22.05.2013 - 31.05.2013
Reserve	MS7: Projektabschluss	01.06.2013 - 14.06.2013

### 9.4.2.1 Phasen / Iterationen

Die genaue Planung der Arbeitspakete wird in Redmine erstellt und verwaltet. Alle detailliertere Informationen sind dort ersichtlich.

#### 9.4.2.1.1 Inception

Dauer	1 Woche
Beschreibung	Es wird eine erste Fassung des Projektplans erstellt, um den groben Ablauf des Projektes zu planen. Zweite Aufgabe ist es, sich einen Überblick über das maschinelle Lernen und die Textklassifizierung zu verschaffen. Die Ergebnisse, die während dem Einarbeiten in die Thematik gewonnen werden, sind in einem Dokument festzuhalten, welches die Entscheidungsfindung aufzeichnen sollte.

#### 9.4.2.1.2 Elaboration

Dauer	3 Wochen
Beschreibung	Aufgrund der Ergebnisse der Inception Phase, wird entschieden, welche Algorithmen und Methoden verwendet werden. Folgend werden diejenige in einer weiteren Phase genauer betrachtet, um zusätzliche Variationen und Verbesserungsvorschläge zu finden. Gleichzeitig sollen auch Frameworks evaluiert werden, welche für die Umsetzung der Lösung herbeigezogen werden können. In einem weiteren Schritt wird die Software Architektur, die internen Abläufe so wie die Anforderungen an das Programms definiert und in einem Dokument festgehalten.

#### 9.4.2.1.3 Construction

Dauer	9 Wochen
Beschreibung	Die Software wird in dieser Phase implementiert und umgesetzt. Der Fokus der Entwicklung wird auf Modularität und Wartbarkeit gelegt. Falls genügend Zeit vorhanden ist, werden auch optionale Features implementiert. Parallel zur Implementierung werden immer Unit Tests geschrieben. Am Ende der Construction wird der Code überarbeitet und Refactored.

#### 9.4.2.1.4 Transition

Dauer	1.5 Wochen
Beschreibung	Abschlusspräsentation wird vorbereitet und die formellen Dokumente, die für die Bachelorarbeit erforderlich sind erstellt. Die gesamte Projektdokumentation wird nochmals überarbeitet und für die Abgabe vorbereitet.

#### 9.4.2.1.5 Reserve

Dauer	2 Wochen
Beschreibung	Alle Arbeiten werden so geplant, dass das Projekt am 31.05.2013 abgeschlossen werden kann. Die letzten zwei Wochen dienen als Reserve, falls unvorhergesehene Ereignisse eintreten.

### 9.4.2.2 Meilensteine

#### 9.4.2.2.1 MS1: Inception

Datum	26.02.2013
Iteration Beschreibung	Inception Überblick über die verschiedenen Methodiken zur Klassifizierung erarbeiten und in einem Dokument festhalten. Gemeinsame Planung des Projektes und Dokumentation des Vorgehens.
Artefakte	<ul style="list-style-type: none"> <li>• Projektplan</li> <li>• Risikoanalyse</li> <li>• Beschreibung der verschiedenen Methoden und Algorithmen</li> </ul>

#### 9.4.2.2.2 MS2: Elaboration

Datum	19.03.2013
Iteration	Elaboration E1, Elaboration E2
Beschreibung	Vertiefung in die Methodiken der Klassifizierung. Einrichten der Infrastruktur und der Projekttools. Entscheid, welche Algorithmen und Frameworks verwendet werden. Anforderungen sind ausgearbeitet und beschrieben. Architektur der Software ist definiert.
Artefakte	<ul style="list-style-type: none"> <li>• Problemanalyse</li> <li>• Entscheidung begründet und dokumentiert</li> <li>• Framework Beschreibungen</li> <li>• Anforderungsspezifikation</li> <li>• Softwarearchitektur</li> <li>• Eingerichtete Infrastruktur</li> </ul>

#### 9.4.2.2.3 MS3: Construction C1

Datum	02.04.2013
Iteration	Construction C1
Beschreibung	In dieser Phase wird ein erster vertikaler Prototyp der Funktionalität erstellt und getestet. Dazu wird eine vereinfachte Variante des gewählten Klassifizieralgorithmus umgesetzt. Anschliessend wird in einem Test festgehalten, wo Verbesserungsmöglichkeiten bestehen und wie die umgesetzt werden können.
Artefakte	<ul style="list-style-type: none"> <li>• Prototyp des Klassifizieralgorithmus</li> <li>• Validierung des Prototyps</li> </ul>

#### 9.4.2.2.4 MS4: Construction C2/C3



Datum	30.04.2013
Iteration Beschreibung	Construction C2, Construction C3 Der Prototyp soll ausgebaut und verbessert werden, damit er automatisch die optimale Konfiguration findet. Zudem werden die E-Mails sauber gefiltert und eine Spracherkennung umgesetzt, mit dem Ziel, dass noch bessere Ergebnisse erreicht werden können. Auch hier wird ein Test gemacht, der aufzeigen soll, was für Verbesserungen erzielt worden sind. Ausserdem soll definiert werden, wie softwaretechnisch ein Logging umzusetzen wäre, das die Schritte des Algorithmus festhält.
Artefakte	<ul style="list-style-type: none"> <li>• Spracherkennung für verbessertes Filtering</li> <li>• Automatische Optimierung durch den Algorithmus</li> <li>• Validierung des erweiterten Prototypen inklusive Vergleich mit Vorgänger</li> <li>• Datenmodell für Logging</li> <li>• Entwurf des Konfiguration UI</li> </ul>

#### 9.4.2.2.5 MS5: Construction C4

Datum	14.05.2013
Iteration Beschreibung	Construction C4 Im Vordergrund dieser Iteration steht die Implementation eines konfigurierbaren Loggers um die Prozesse und Entscheidungen in einem Klassifikationsvorgang festzuhalten. Des weiteren wird die K Cross Validation Methode für das Training umgesetzt. Diese Methode soll helfen optimale Parameter für das Training zu finden. Aufbauend auf dem Entwurf der vorherigen Iteration wird ein Konfigurations UI realisiert. In diesem werden Einstellungen vorgenommen und der Trainingsvorgang in Gang gesetzt.
Artefakte	<ul style="list-style-type: none"> <li>• Cross Validation für das Training</li> <li>• Konfigurations UI Umsetzung</li> <li>• Logging Implementation</li> </ul>

#### 9.4.2.2.6 MS6: Construction C4

Datum	21.05.2013
Iteration Beschreibung	Construction C5 Dies ist die letzte Iteration der Construction. Ziel ist es, den Programmcode sowie die Dokumentation für die Abgabe vorzubereiten. Ausserdem wird ein Test der Software durchgeführt, der zeigt, wie gut die Klassifizierung funktioniert und welche Features umgesetzt wurden. Ist noch genügend Zeit vorhanden, können auch optionale Features umgesetzt werden.
Artefakte	<ul style="list-style-type: none"> <li>• Abgabebereiter Quellcode</li> <li>• Kerndokumente sind abgeschlossen</li> </ul>

#### 9.4.2.2.7 MS7: Projektabschluss

Datum	14.06.2013
Iteration Beschreibung Artefakte	Transition T1, Reserve Schlussabgabe wird fertiggestellt, formale Dokumente erstellt. <ul style="list-style-type: none"> <li>• Gesamte Projektdokumentation fertiggestellt</li> </ul>

### 9.4.3 Besprechungen

Besprechungen finden wöchentlich am Mittwoch Morgen um 8:00 Uhr statt bei der Luware AG, Zürich Technopark statt. Ausnahmen oder Verschiebungen werden bilateral geregelt.

Ziele der wöchentlichen Meetings sind

- Was wurde seit dem letzten Meeting erreicht?
- Wie sehen die nächsten Arbeitsschritte aus?
- Sind Probleme aufgetaucht?

## 9.5 Risikomanagement

### Risikomanagement: Selbstlernende Software zur Analyse von Texten

Gewichteter Schaden: 21.3

Nr	Titel	Beschreibung	max. Schaden [h]	Eintrittswahrscheinlichkeit	Gewichteter Schaden	Vorbeugung	Verhalten beim Eintreten
R1	Lokaler Datenverlust	Verlust der Daten einer ganzen Woche, eines einzelnen Teammitgliedes.	8	5%	0.4	Häufige Synchronisation auf den Server. Alle Projektrelevanten Daten in der Dropbox speichern	Verlorene Arbeit wiederholen
R2	Ausfall durch Krankheit oder Unfall	Ausfall einer Person für je eine Woche	24	20%	4.8	Keine.	Kompensation durch das andere Teammitglied soweit
R3	Server-Datenverlust	Ausfall des virtuellen Servers mit Datenverlust auf dem Server.	12	5%	0.6	Daten sind lokal sowie auf dem Server vorhanden.	Wiederherstellen des Servers mit lokalen Daten sowie Rekonfiguration des Servers
R4	Totalausfall eines Teammitglieds	Ein Teammitglied hört auf an dem Projekt mitzuarbeiten.	360	0.5%	1.8	Keine.	Teilweise Kompensation durch andere Teammitglieder.
R5	Falsche Algorithmenwahl	Die ausgewählte Algorithmen eignen sich schlecht dazu Emails zu klassifizieren	50	5%	2.5	Gründliche Evaluation der möglichen Algorithmen und Gegenüberstellung ihrer Vor- und Nachteile	Algorithmus muss durch einen leistungsfähigeren ausgetauscht werden um bessere Resultate zu erzielen. Verantwortliche Komponente
R6	Keine Trainingsdaten	Es wird für das Projekt keine Trainingsdaten zur Verfügung gestellt	32	20%	6.4	Regelmäßiges erkundigen wie der Status der Trainingsdaten aussieht	Es müssen zuerst Trainingsdaten erzeugt werden. Erfolgsrate der Arbeit ist nicht repräsentativ da qualitativ
R7	Optimierungsverfahren für Klassifizierungsalgorithmus schlecht	Das ausgewählte Optimierungsverfahren was für das verbessern des Klassifizierungsalgorithmus im produktiven Betrieb verwendet wird	32	15%	4.8	Gründliches Informieren über vorhandene Optimierungsverfahren	Optimierungsverfahren muss durch ein anderes ersetzt werden. Verantwortliche Komponente muss neu
<b>Summe</b>			<b>518</b>		<b>21.3</b>		

Tabelle 9.1: Risikomanagement

## 9.6 Infrastruktur

### 9.6.1 Arbeitswerkzeuge

Die Teammitglieder verwenden Ihre persönlichen Laptops für die Arbeit. Zusätzlich ist von der HSR ein Desktop Computer und Arbeitsplatz zur Verfügung gestellt worden, der für die Arbeit verwendet werden kann.

### 9.6.2 Zentraler Server

Dem Team steht ein Ubuntu Server zur Verfügung. Auf dem Server läuft Redmine und das Git.

### 9.6.3 Entwicklungsumgebung

Für die Entwicklung wird mit Visual Studio 2012 Ultimate gearbeitet. Visual Studio ist bei allen Teammitgliedern gleich eingerichtet, so dass überall die gleichen Formatierungsregeln etc. verwendet werden.

Zusätzlich wird das Visual Studio Plugin ReSharper eingesetzt. ReSharper erweitert das Visual Studio unter anderem um wertvolle Refactoring Methoden die während dem Entwickeln eingesetzt werden.

### 9.6.4 Betriebssystem

Die Teammitglieder verwenden Windows 8 Professional.

### 9.6.5 Dokumentationswerkzeug

Alle Dokumente werden mit  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  verfasst. Grafiken werden mit dem Visio 2013 oder Astah Community erstellt.

## 9.7 Qualitätsmassnahmen

**Tabelle 9.2:** Qualitätsmassnahmen

Massnahme	Ziel
Dropbox Ordner	Fachliteratur in digitaler Form werden in einem gemeinsamen Dropbox Ordner abgelegt, damit alle Teammitglieder immer Zugriff darauf haben.
Redmine	Redmine wird für das Projektmanagement verwendet. Dort werden alle Tasks und die benötigte Zeit dafür erfasst. Damit ist es jederzeit möglich eine Projektauswertung zu machen.
Git als Versionierungssystem	Git wird als Versionierungssystem für den Code und die Dokumentation verwendet. Somit kann jederzeit auf frühere Versionen zurückgegriffen werden. Das Master Repository wird von der HSR zur Verfügung gestellt und gehostet.
Entwicklungswerkzeuge definiert	Keine Komplikation beim Zusammenarbeiten.
Pair Programming	Bei kritischen Codeabschnitten wird Pairprogramming verwendet. Dies führt zu einheitlicherem Code Design und Fehlerreduktion.
Code Reviews	Fehlerfindung und Informationsaustausch
Unit Tests parallel zur Entwicklung	Möglichst grosse Testabdeckung in allen Stadien des Projektes zur Fehlerminimierung
Systemtests	Nach jeder Iteration der Construction Phase werden Systemtests durchgeführt, um den produzierten Code zu verifizieren und allfälligen Breaking Changes zu identifizieren und zu beheben.

### 9.7.1 Dokumentation

Die Dokumentation wird in  $\text{\LaTeX}$  verfasst und wird auch im Git Repository abgespeichert. Damit kann jederzeit auf eine frühere Version zurückgegriffen werden. Auch das parallele Arbeiten am selben Dokument wird damit ermöglicht.

### 9.7.2 Projektmanagement

Für das Projektmanagement wird Redmine verwendet. Die Redmineseite des Projektes ist unter <http://tcba.no-ip.org/redmine/projects/email-classification> erreichbar.

### 9.7.3 Entwicklung

Für die Code Versionierung wird Git verwendet. Das Master Git Repository wird an der HSR gehostet.

### 9.7.3.1 Vorgehen

Bei diesem Projekt wird nach RUP gearbeitet.

### 9.7.3.2 Unit Testing

Für das Unit Testing werden die von Visual Studio zur Verfügung gestellten Mittel benutzt.

### 9.7.3.3 Code Reviews

Code Reviews werden gemacht falls sie nötig sind.

### 9.7.3.4 Code Style Guidelines

- Comments, Code und Commits in Englisch
- Todos:  
    TODO: [Kürzel] tt-mm-jjjj Text
- Namenskonvention:  
    Private Member:  
        private int counter  
    Properties:  
        public int Counter {get; set;}  
    Methoden: Beginnen mit Grossbuchstaben und enthalten keine Underlines  
        public int GetValue(){}  
    EventHandler: Beginnen mit On und enden mit dem Verb+ed  
        private void OnCallStateChanged(object sender, CallStateChangedEventArgs args)  
        public static String ToFullExceptionStringCust(this Exception source, bool printStack-  
Trace = false)
- If Statements müssen mit {} geschrieben werden.

## 9.7.4 Testen

### 9.7.4.1 Modultest

Kernkomponenten sollen mit Unit-Tests abgedeckt werden. GUI sowie nicht-kritische Komponenten benötigen keine Unit-Tests. Beim Testen soll ein Mocking Framework verwendet werden, um das Testing zu vereinfachen.

### 9.7.4.2 Integrationstest

Integrationstests testen die gesamte Architektur. Dazu werden einzelne Komponenten als Fake-Objekte implementiert, damit das System ohne deren Existenz getestet werden kann.

### 9.7.4.3 Systemtest

Am Ende der Construction wird ein Systemtest durchgeführt um das reibungslose Zusammenspiel der einzelnen Komponenten zu verifizieren.

## 9.8 Projektauswertung

### 9.8.1 Zeit

Das Zeitbudget für dieses Projekt beträgt 720 Stunden. Effektiv Aufgewendet wurden 865 Stunden. Dies ergibt einen Mehraufwand von ca. 20%.

Der Grund für diesen Mehraufwand lässt sich durch den Mangel an Erfahrung mit intelligenten Systemen erklären. Dies resultierte automatisch in einer höheren Einarbeitungszeit. Ein anderer Grund ist auch die Motivation und Bereitschaft der Teammitglieder, mehr als das Minimum zu leisten.

Die detaillierte Stundenaufteilung sieht wie folgt aus:

Name	Aufwand
Fabian Senn	430 Stunden
Cyrill Lam	435 Stunden

**Tabelle 9.3:** Zeitauswertung

### 9.8.2 Code Statistiken

Die Abgabe des Codes beinhaltet 3079 Zeilen produktiven Code. Nachfolgend kommt eine Übersicht über die einzelnen Visual Studio Projekte mit dem Wartbarkeitsindex und deren zyklomatischen Komplexität. Alle Angaben wurden direkt mit den integrierten Auswertungsmöglichkeiten aus dem Visual Studio erstellt. Die Testprojekte wurden bei dieser Statistik nicht berücksichtigt.

Projektname	Wartbarkeit	zyklomatische Komplexität	Lines Of Code
EmailClassifier.MailImport	80	46	86
EmailClassifier.Util	77	22	36
EmailClassifier.Ui.Demo	95	200	286
EmailClassifier.Ui	92	536	886
EmailClassifier.MachineLearning.Interfaces	97	74	61
EmailClassifier.MachineLearning	78	85	197
EmailClassifier.Logging	86	44	52
EmailClassifier.LanguagIdentifier	77	33	59
EmailClassifier.Interfaces	98	137	106
EmailClassifier.FeatureExtraction	73	52	108
EmailClassifier.Data	92	292	495
EmailClassifier.Core	75	219	544
EmailClassifier.Console	76	17	40
EmailClassifier.Configuration	81	78	123

**Tabelle 9.4:** Codestatistiken



## 10 User Manual

In diesem Kapitel wird kurz die Inbetriebnahme der Software und die Möglichkeiten der einzelnen User Interface Screens erläutert.

Detailliertere Informationen zu den dahinter liegenden Konzepten werden hier nicht noch einmal erklärt. Diese können bei Bedarf den entsprechenden vorherigen Kapiteln entnommen werden.

### 10.1 Systemanforderungen

Für die Inbetriebnahme der Software sind folgende Systemanforderungen nötig:

- Betriebssystem: Windows 7 und höher
- .Net-Version: 4.5
- SQL-Server: MS-SQL Express 2008
- Mindestens 4GB Arbeitsspeicher.

### 10.2 Installation

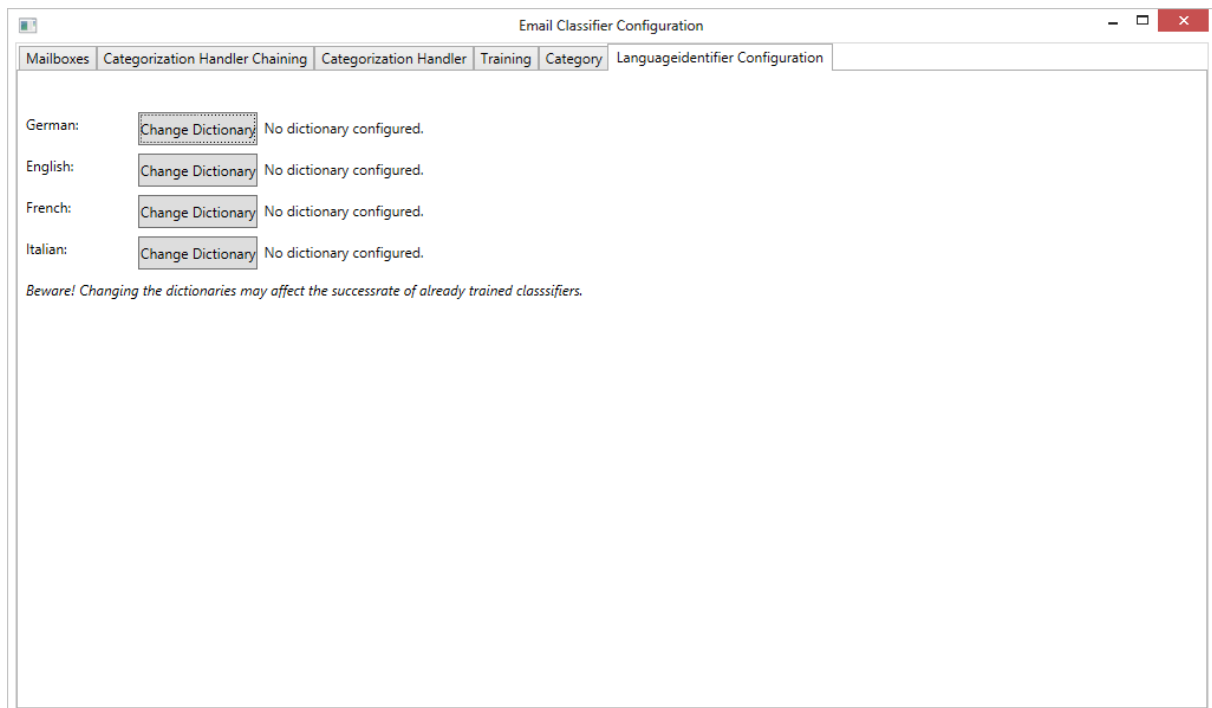
Die Applikation kann als Exe-Datei ausgeliefert werden. Einzig die Datenbank muss zusätzlich angelegt werden. Die Datenbank muss den Namen *EmailClassifier* Tragen. Ein SQL-Script, welches die notwendigen Tabellen und Relationen erstellt befindet sich im Projekt *EmailClassifier.Data*.

Der Zugriff auf den Datenbankservers kann über das File *cstrings.config* geregelt werden.

### 10.3 User Interface

Nachfolgend werden die einzelnen Screens des User Interfaces kurz beschrieben. Das User Interface ist momentan sehr funktional orientiert.

### 10.3.1 Languageidentifizier Configuration



**Abbildung 10.1:** User Interface: Languageidentifizier Configuration

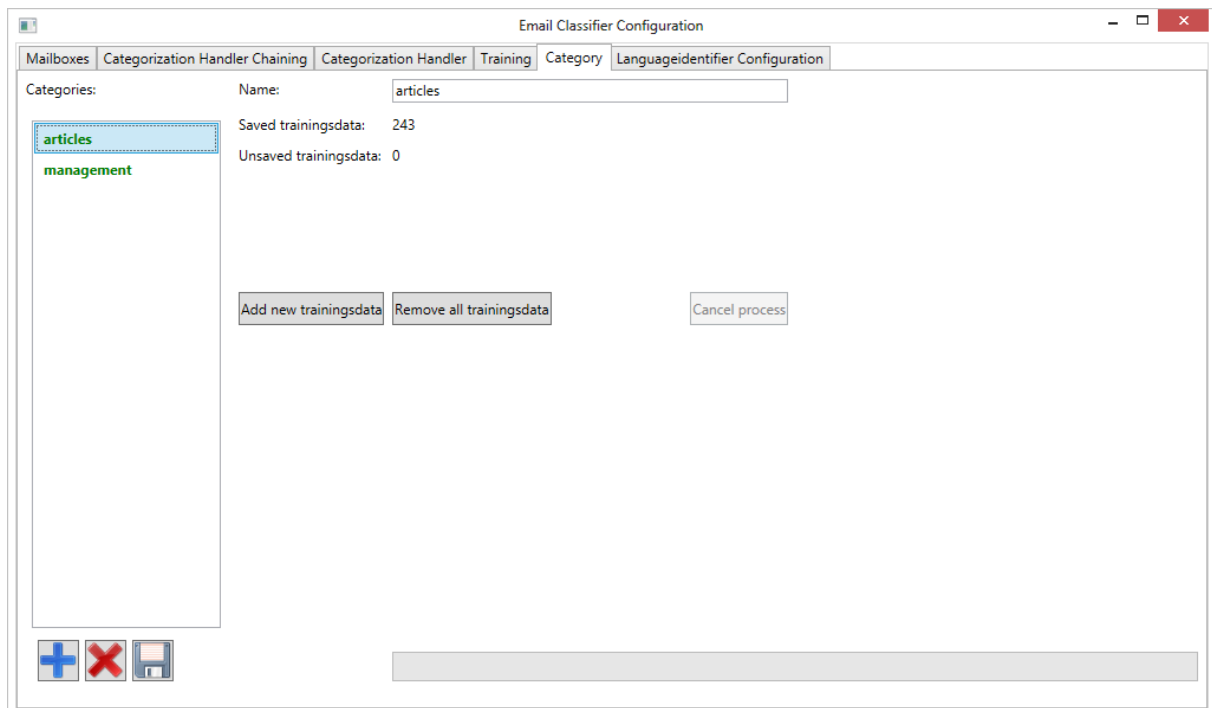
In diesem Dialog werden die Wörterbücher einzelner Sprachen für die Spracherkennung festgelegt.

Verschiedene Wörterbücher sind im Open Source Projekt *NTextCat* erhältlich, welches unter <http://ntextcat.codeplex.com/releases/> verfügbar ist.

Im UI ist ersichtlich wann das letzte mal ein Wörterbuch aktualisiert wurde.

Wenn ein Wörterbuch ändert, kann dies negative Einwirkungen auf die Qualität eines bereits trainierten Klassifizierers haben, da die Spracherkennung auch beim Training des Klassifizierers zum Einsatz kommt. Es ist deshalb empfohlen, nach dem Festlegen eines neuen Wörterbuches, ein erneutes Training zu starten.

### 10.3.2 Category



**Abbildung 10.2:** User Interface: Category

Im Category-Screen werden die Kategorien verwaltet. Hier können Kategorien neu erstellt, gelöscht und umbenannt werden. Des Weiteren werden über dieses Dialog-Fenster Trainingsdaten den einzelnen Kategorien hinzugefügt.

Wenn eine Kategorie ausgewählt ist, können über einen Klick auf *Add new trainingsdata* neue Trainingsdaten ins Programm geladen werden. Die Trainingsdaten müssen dabei im MIME Format sein. Mit Betätigung des Speichern-Buttons unten links werden die Trainingsdaten in die Datenbank gespeichert.

Mit dem Löschen-Button können Kategorien gelöscht werden. Sind noch Klassifizierer mit einer solchen Kategorie im Einsatz, werden diese weiterhin funktionieren. Neue Klassifizierer hingegen können mit dieser Kategorie nicht mehr trainiert werden.

### 10.3.3 Training

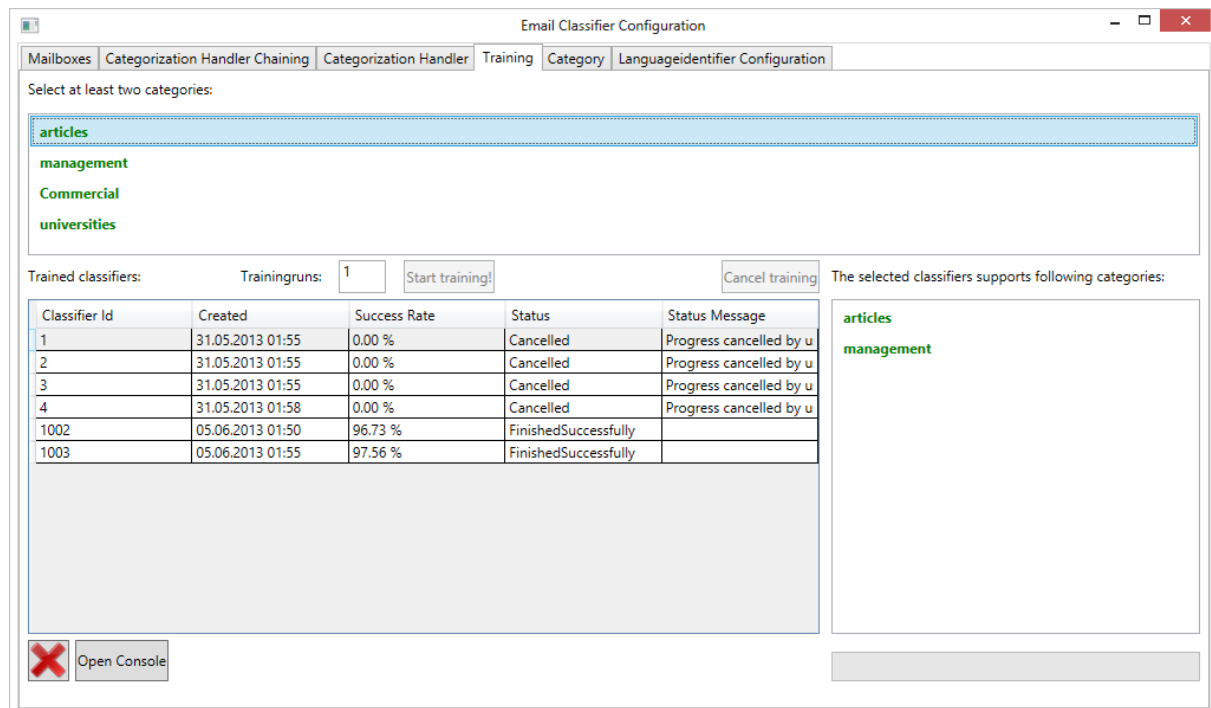


Abbildung 10.3: User Interface: Training

Im Training-Fenster werden neue Klassifizierer trainiert.

In der oberen Hälfte sind alle Kategorien ersichtlich. Davon müssen mindestens zwei ausgewählt werden, um ein Training zu starten. Eine Kategorie kann nur für ein Training verwendet werden, wenn sie mindestens 20 Trainingsdaten enthält. Bei weniger als 20 Trainingsdaten würde der Klassifizierer beinahe unbrauchbar werden.

Im Allgemeinen gilt, je mehr Trainingsdaten desto besser!

Im Eingabefeld *Trainingsruns* kann die Anzahl an Trainings konfiguriert werden. Mit Klick auf den Button *Start training!* wird das Training in Gang gesetzt. Je nach Anzahl und Qualität der Trainingsdaten kann der Vorgang sehr lange dauern. Um detaillierte Informationen zum Trainingsprozess zu erhalten, kann mit Klick auf den Button *Open Console* ein Konsolenfenster geöffnet werden. Dort erscheinen in Echtzeit Statusmeldungen zum Trainingsvorgang.

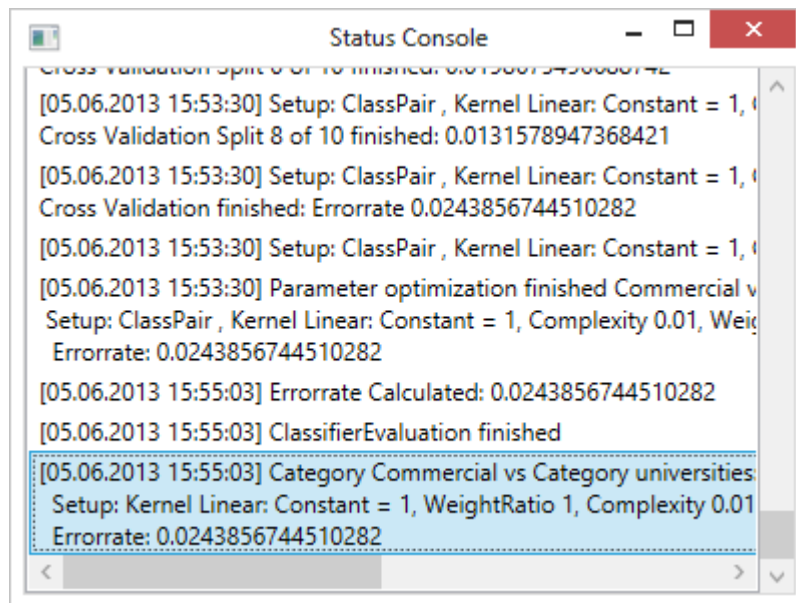


Abbildung 10.4: User Interface: Trainings Konsolenfenster

Bereits trainierte Klassifizierer werden in der unteren Hälfte aufgelistet. Wird einer ausgewählt, sind die Kategorien, die er unterstützt, ersichtlich.

Mit einem Klick auf den Löschen-Button wird der Klassifizierer gelöscht.

### 10.3.4 Categorization Handler

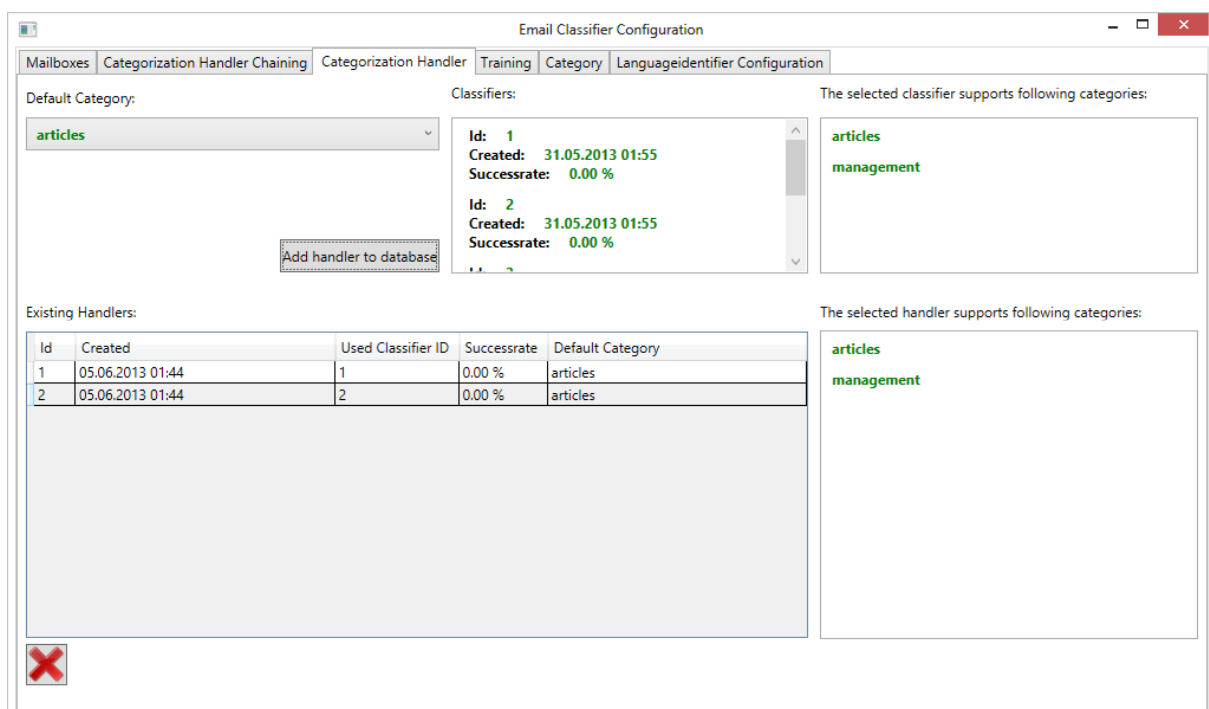
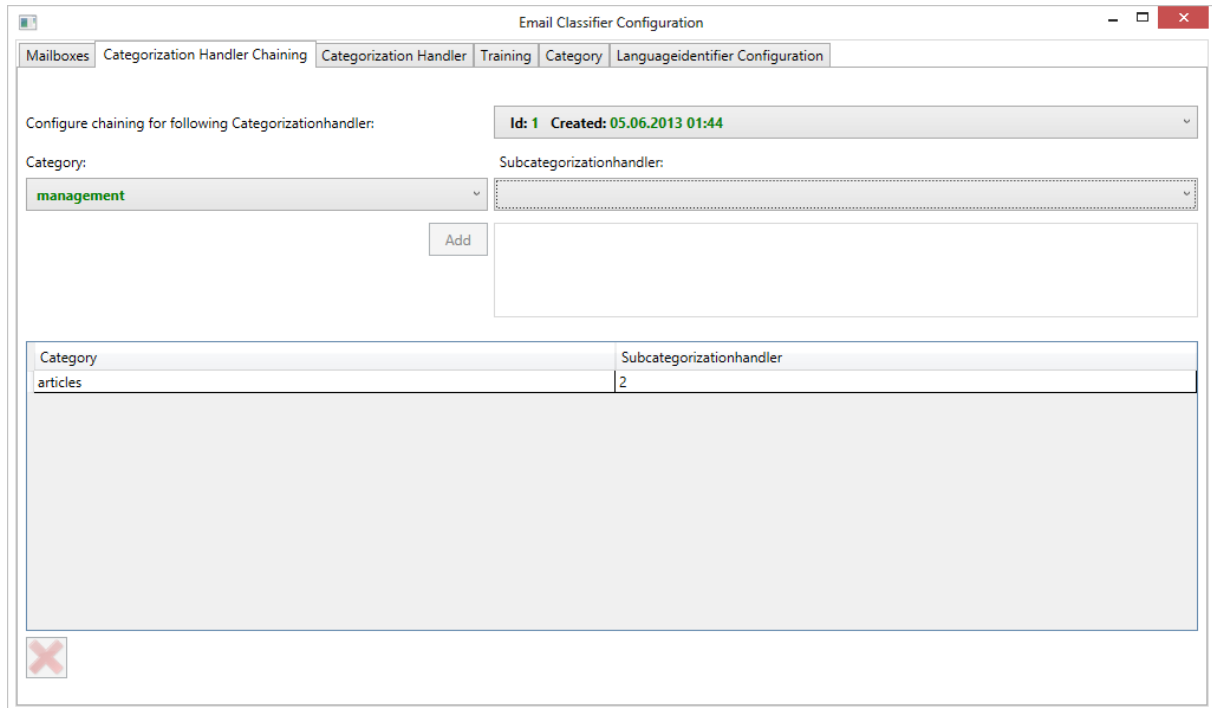


Abbildung 10.5: User Interface: Categorization Handler

In der Categorization Handler Ansicht können Categorization Handler erstellt werden. Dazu wird ein Klassifizierer und eine gewünschte Standardkategorie ausgewählt. Mit Klick auf den Button *Add handler to database* werden die Einstellungen in die Datenbank gespeichert.

Ausgewählte Categorization Handler können auch wieder gelöscht werden.

### 10.3.5 Categorization Handler Chaining



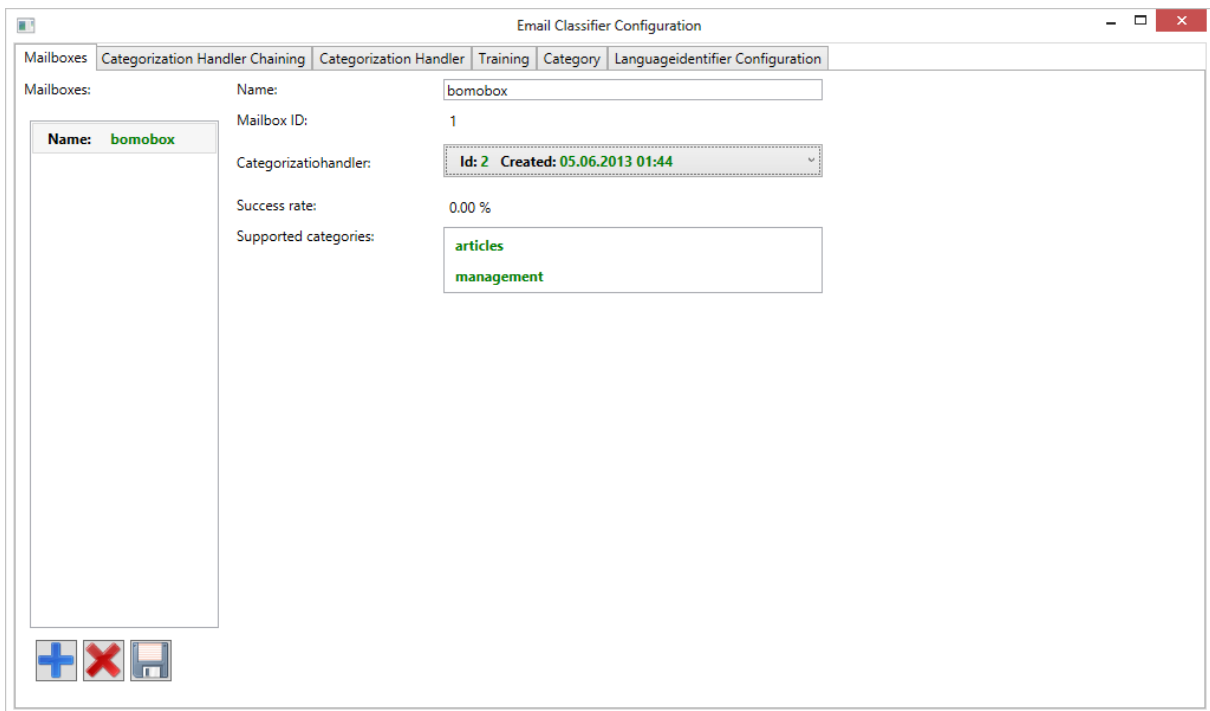
**Abbildung 10.6:** User Interface: Categorization Handler Chaining

In dieser Ansicht kann das Categorization Handler Chaining konfiguriert werden. Im obersten Dropdown-Feld wird derjenige Handler ausgewählt, für den das Chaining konfiguriert werden soll.

Danach kann eine seiner Kategorien ausgewählt und ein Handler, der die weiterführende Klassifizierung vornimmt, festgelegt werden.

Ein Categorization Handler, kann immer nur einmalig in einer Categorization Handler Chain definiert werden.

### 10.3.6 Mailbox



**Abbildung 10.7:** User Interface: Mailbox

In der Mailbox View wird das Mailbox Objekt verwaltet. Hier können Mailboxen erstellt, umkonfiguriert oder gelöscht werden. Einer Mailbox wird immer ein Categorization Handler zugewiesen. Besitzt der Categorization Handler ein Chaining, so werden im Infobereich alle Möglichen Kategorien angezeigt.

### 10.3.7 Mail Emulation

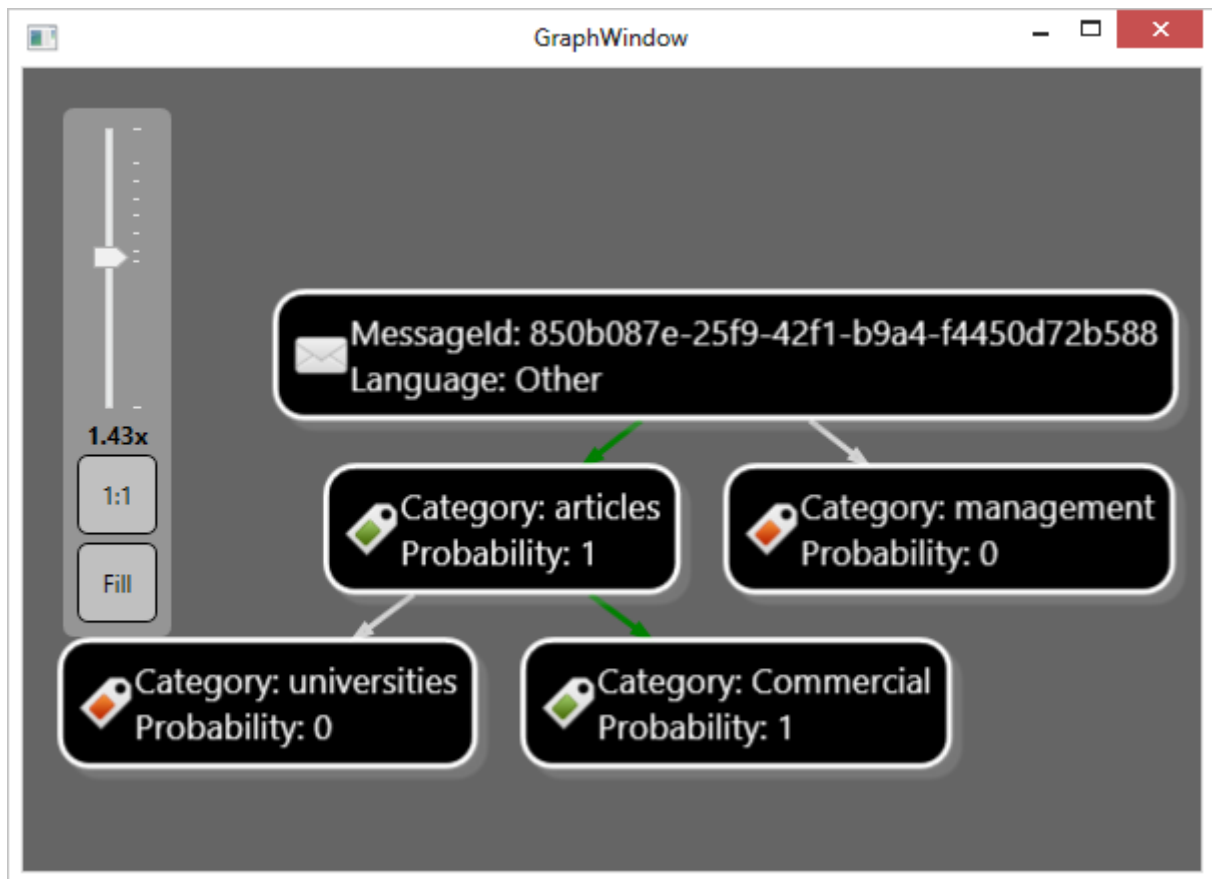
**Abbildung 10.8:** User Interface: Mail Emulation

In der Mail Emulation View, kann das Kategorisieren eines Mails ausgetestet werden. Dabei kann ein "Email" selbständig erfasst oder als MIME File eingelesen.

Im Feld *Mailbox Id* wird die Id der Mailbox angegeben, die angesprochen werden soll. Mit Klick auf den Button *Classify Mail* wird die Klassifizierung gestartet. Das Resultat der Klassifizierung wird in der unteren Hälfte angezeigt.

Mit einem Klick auf den *Showgraph* Button wird der Entscheidungsweg aufgezeigt.





**Abbildung 10.9:** User Interface: Visualisierung

Der Entscheidungsweg wird als Baum gezeichnet. Der oberste Knoten symbolisiert das E-Mail. Alle anderen Knoten sind die möglichen Kategorien. Die grünen Kanten zeigen auf, für welche Kategorien sich der Klassifizierer entschieden hat. Der unterste Knoten, zu dem eine grüne Kante führt, ist somit die finale Kategoriezuweisung.

## 11 Eigenständigkeitserklärung

Ich erkläre hiermit,

- dass ich die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt habe, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt ist oder mit dem Betreuer schriftlich vereinbart wurde,
- dass ich sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben habe.
- dass ich keine durch Copyright geschützten Materialien (z.B. Bilder) in dieser Arbeit in unerlaubter Weise genutzt habe.

Ort, Datum:

Rapperswil, 05.06.2013

Name, Unterschrift:

C. Lam

F. Senn

Cyrill Lam

Fabian Senn

## 12 Vom Betreuer unterschriebene Aufgabenstellung

### Ausgangslage:

Gerade in der Zeit der heutigen Informationsflut ist die automatische und intelligente Analyse von Texten sehr wichtig. Dabei müssen schon kleinste Texte klassifiziert und einer Kategorie zugeordnet werden können. Dies hilft, die Menge an Informationen besser zu bewältigen. Da sich der Inhalt, Sprache und die Kategorien über die Zeit schnell ändern können, müssen solche Systeme die Fähigkeit besitzen, die neuen Begebenheiten selbstständig zu lernen.


Im Bereich der Kundendienste wird eine solche Analyse von Texten in der Kategorisierung von Emails benötigt. Diese soll dienen, die Kundenanfragen an die dafür zuständigen Agenten zu leiten. Je besser und genauer die Klassifizierung ist, desto kürzer sind die Antwortzeiten und letztendlich desto geringer sind die entstehenden Kosten. Die Firma Luware AG entwickelt zurzeit eine E-Mail Integration für ihr auf Microsoft Lync basiertes Contact Center. Im Zuge dieser Neuentwicklung wird eine Machbarkeitsanalyse / ProofOfConcept eines Email Klassifizierungsprogramms benötigt. Dieses soll die Kategorisierung von Email selbstständig lernen und somit einen wichtigen Beitrag zur optimalen Verteilung leisten.

### Aufgabe:

In dieser Bachelor Arbeit soll eine Applikation auf der Basis von Microsoft .net C# entwickelt werden, welche Texte nach vorgegebenen Kategorien optimal klassifiziert. Dabei soll die Applikation anhand von bereits kategorisierten Daten trainieren können und die richtige Klassifizierung lernen. Die Applikation soll Texte vom Umfang einer Email verarbeiten können. Neben der Typisierung in Kategorien soll vorgängig die Sprache im Text bestimmt und Optional eine Regelbasierte Klassifizierung ermöglicht werden.

Rapperswil, den

*18 Februar 2013*

  
.....

Der Betreuer

## 13 Glossar

Terminus	Beschreibung
API	Application Programming Interface. Schnittstelle von Softwaresystemen, die eine Anbindung durch andere Programme ermöglicht.
Interface	Ein Interface ist eine Schnittstelle, die definiert, wie mit einem Objekt kommuniziert werden kann.
Library	Ist eine Ansammlung an Methoden, die für gewisse Problemstellungen implementiert worden sind und so den Programmieraufwand minimieren, da sie immer wieder benutzt werden können.
RDP	Abkürzung von Remote Desktop Protokoll. Ist ein Protokoll, das dazu verwendet wird Inhalte von entfernten Rechnern darzustellen oder zu steuern.
.Net	Ist die Common Language Runtime von Microsoft für Software Entwicklung. Sie bietet eine Grosszahl an Bibliotheken und Schnittstellen an, die es erlauben, komplette Programmiersprachen darauf aufbauen zu lassen.
XML	Extensible Markup Language. Ist eine Sprache, die zur Darstellung hierarchisch strukturierter Daten geeignet ist. Wird heutzutage oftmals verwendet, um beispielsweise User Interfaces zu erstellen.
UTC	Abkürzung für Coordinated Universal Time. Ist heutzutage quasi das Standard-Zeitformat in Anwendungen.
GIT	Dezentralisiertes Software Versionierungssystem.
Online Services	Dienste, wie zum Beispiel das Erkennen von Sprache, die über das Internet angeboten werden.
OS	Operating System - Betriebssystem.
Vektor	Eine Mathematisches Objekt definiert durch ein $N$ -Tupel von reellen Zahlen.
MS	Microsoft
SQL	Standard Query Language - Standard Abfragesprache für Datenbanken
RUP	Rational Unified Process – Ist ein Vorgehensmodell für die Softwareentwicklung.
XML	Extensible Markup Language - Eine Auszeichnungssprache zur Darstellung hierarchisch strukturierter Daten in Form von Textdateien.
IDE	Integrated Development Requirement - Entwicklungsumgebung.
Visual Studio	IDE von Microsoft für .Net entwicklung.
Visual Studio Solution	Repräsentieren ein Softwareprojekt im Visual Studio.
DLL	Dynamic Link Library - Typische Programmbibliothek, die von einer Windows Applikation verwendet werden kann.
MIME	Multipurpose Internet Mail Extensions - Standardformat für Emails.
Refactoring	Änderung am Code um die Struktur zu verbessern.
Textkorpus	Ansammlung von schriftlichen Texten.
WPF	.Net GUI Technologie

**Tabelle 13.1:** Glossar

# Abbildungsverzeichnis

3.1	Übersicht gesamter Klassifikationsvorgang	5
4.1	Use Case Diagramm	9
5.1	Lebenszyklus eines Klassifizierers	24
5.2	Support Vector	27
5.3	Separierbarkeit	28
5.4	Binäre Klassifikationsprobleme	29
5.5	Künstliches neuronales Netzwerk	30
6.1	Domainmodell	38
6.2	Domainmodell	39
7.1	3-Fold Cross Validation	44
7.2	Ablauf: Trainingsvorgang	45
7.3	Ablauf: Trainingsvorgang	47
7.4	Domainmodell: Trainingsvorgang	48
7.5	Ablauf: Trainingsvorgang Neuimplementation	50
7.6	Ablauf: Parameteroptimierung	52
7.7	Modell: Klassifizierung	56
7.8	Ablauf: Klassifizierungsvorgang Activity	57
7.9	Ablauf: Klassifizierungsvorgang mit Chaining	60
7.10	Screenshot: Demo Applikation	61
7.11	Layerübersicht der Architektur	64
7.12	Abhängigkeiten: EmailClassifier.Core	68
7.13	Abhängigkeiten: EmailClassifier.Ui.Demo	69
7.14	Abhängigkeiten: EmailClassifier.FeatureExtraction	69
7.15	Abhängigkeiten: EmailClassifier.Logging	70
7.16	Abhängigkeiten: EmailClassifier.MailImport	70
7.17	Abhängigkeiten: EmailClassifier.Languageldentifier	71
7.18	Abhängigkeiten: EmailClassifier.UnitTest	71
7.19	Übersicht des gesamten Datenmodells	73
7.20	Datenbankklasse: AdaptiveClassifier	74
7.21	Datenbankklasse: EmailFieldConfiguration	75
7.22	Datenbankklasse: MailBox	76
7.23	Datenbankklasse: CategorizationHandler	77
7.24	Datenbankklasse: LanguageldentifierConfiguration	78
7.25	Datenbankklasse: SubCategorizationHandlerAssignment	79
7.26	Datenbankklasse: Category	80
7.27	Datenbankklasse: Trainingdata	81
7.28	Datenbankklasse: Field	82
7.29	Datenbankklasse: ProbabilisticCategorizationResult	82
7.30	Datenbankklasse: CategorizationTask	83
7.31	Interface: BagOfWords	87

7.32 Interface: CategorizationHandlerService . . . . .	89
7.33 Interface: ICategoryDecider . . . . .	90
7.34 Interface: IConcreteCategorizationHandler . . . . .	91
7.35 Interface: IFeatureExtractor . . . . .	91
7.36 Interface: ILanguageIdentifier . . . . .	92
7.37 DTO: MailDto . . . . .	93
7.38 DTO: CategorizationRequest . . . . .	94
7.39 DTO: CategorizationResponse . . . . .	94
7.40 Interface: IMulticlassSupportVectorMachine . . . . .	95
7.41 Interface: IKernel . . . . .	96
7.42 Interface: IMulticlassSvmTraining . . . . .	96
7.43 Interface: ISequentialMinimalOptimization . . . . .	97
7.44 Interface: IGridSearch . . . . .	97
7.45 Interface: ICrossValidation . . . . .	98
7.46 Interface: IFoldTraining . . . . .	99
7.47 Delegate: ParameterOptimizingFunction . . . . .	100
7.48 Delegate: SetupTrainingFunction . . . . .	100
7.49 Delegate: FoldTrainingFunction . . . . .	101
7.50 DTO: ClassPair . . . . .	101
7.51 DTO: Setup . . . . .	102
7.52 DTO: Fold . . . . .	102
7.53 DTO: GridSearchResult . . . . .	103
7.54 DTO: CrossValidationResult . . . . .	104
7.55 Logging File Struktur . . . . .	118
7.56 Visualisierung des Resultats . . . . .	119
9.1 Phasen nach RUP . . . . .	144
10.1 User Interface: Languageidentifier Configuration . . . . .	157
10.2 User Interface: Category . . . . .	158
10.3 User Interface: Training . . . . .	159
10.4 User Interface: Trainings Konsolenfenster . . . . .	160
10.5 User Interface: Categorization Handler . . . . .	160
10.6 User Interface: Categorization Handler Chaining . . . . .	161
10.7 User Interface: Mailbox . . . . .	162
10.8 User Interface: Mail Emulation . . . . .	163
10.9 User Interface: Visualisierung . . . . .	164
Abbildungsverzeichnis . . . . .	168

# Tabellenverzeichnis

6.1	Vergleich Klassifikationsalgorithmen . . . . .	36
7.1	Trainingsdaten Übersicht . . . . .	40
7.3	Trainingsdaten Statistiken . . . . .	121
7.4	Testresultat: Test Ende C1 . . . . .	123
7.5	Testresultat: Test Ende C2 . . . . .	124
7.6	Testresultat: Test Parameteroptimierung . . . . .	126
7.7	Testresultat: Test Neuimplementierung der Parameteroptimierung . . . . .	127
7.8	Testresultat: Test Neuimplementierung der Parameteroptimierung 2 . . . . .	129
7.10	Testresultat: Language Identifier Performancetest . . . . .	138
7.11	Unit Test Abdeckung . . . . .	139
9.1	Risikomanagement . . . . .	150
9.2	Qualitätsmassnahmen . . . . .	152
9.3	Zeitauswertung . . . . .	154
9.4	Codestatistiken . . . . .	155
13.1	Glossar . . . . .	167
	Tabellenverzeichnis . . . . .	169

## Literaturverzeichnis

- [1] Dmitry Babenko Haralambos Marmanis. *Algorithms of the Intelligent Web*. Manning, 2009.
- [2] Gary Huang Ron Bekkerman, Andrew McCallum. Automatic categorization of email into folders: Benchmark experiments on enron and sri corpora. Technical report, Center of Intelligent Information Retrieval, CIIR.
- [3] Toby Segaran. *Kollektive Intelligenz*. O'Reilly, 2008.
- [4] Tobias Kaufmann Beat Pfister. *Sprachverarbeitung, Grundlagen und Methoden der Sprachsynthese und Spracherkennung*. Springer, 2008.
- [5] Peter Norvig Stuart J. Russel. *Artificial Intelligence, A Modern Approach*. Prentice-Hall, 1995.
- [6] Wolfgang Ertel. *Grundkurs Künstliche Intelligenz*. Vieweg+Teubner, 2009.
- [7] Anil Ted Dunning Ellen Friedman Sean Owen, Robin. *Mahout in Action*. Manning, 2012.
- [8] Gabriele Kern-Isberner Christoph Beierle. *Methoden Wissensbasierter Systeme*. Vieweg, 2003.
- [9] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.
- [10] Wen-Chyi Lin. A case study on support vector machines versus artificial neural networks. These, University of Pittsburgh, 1998.
- [11] Waleed Zaghloil, Sang M. Lee, Silvana Trimi. Text classification: neural networks vs supportvector machines. Studie, Emerald Insight, 2009.
- [12] Reinhard Rapp. The automatic generation of thesauri of related words for english, french, german, and russian. Technical report, 2009.
- [13] Ram Dayal Goyal. Knowledge based neural network for text classification. Technical report, 2007.
- [14] Taeho Jo. Ntc(neural text categorizer): Neural network for text categorization. Technical report, School of Information Technology and Engineering, Ottawa University, 2010.
- [15] Dominic Savio Lam Lay Yin. Learned text categorization by backpropagation neural network. Technical report, Hong Kon University of Science and Technology, 1996.
- [16] Xiaojin Zhu. Semi-supervised learning literature survey. Technical report, University of Wisconsin – Madison, 2008.
- [17] Satarupa Banerjee Vikramjit Mitra, Chia-Jiu Wang. Text classification: A least square support vector machine approach. Technical report, 2006.
- [18] Manabu Sassano. Virtual examples for text classification with support vector machines. Technical report, Fujitsu Laboratories Ltd.
- [19] David D. Lewis. Applying support vector machines to the trec-2001 batch filtering and routing tasks. Technical report.



- [20] N. E. Ayat, Mohamed Cheriet, and Ching Y. Suen. *Optimization of the SVM Kernels using an Empirical Error Minimization Scheme*, volume 2388 of *Lecture Notes in Computer Science*, pages 354–, 369. 2002.
- [21] John C. Platt. *Sequential minimal optimization: A fast algorithm for training support vector machines*, 1998.
- [22] Hendrik Blom. *Entwicklung von optimierungsverfahren für das lösen verschiedener lernaufgaben mit derstutzvektormethode*, 2011.
- [23] Media Wiki. Künstliches neuronales netz. [http://de.wikipedia.org/wiki/K%C3%BCnstliches\\_neuronales\\_Netz](http://de.wikipedia.org/wiki/K%C3%BCnstliches_neuronales_Netz), 2 2013.
- [24] David Kriesel. Ein kleiner Überblick über neuronale netze. [http://www.dkriesel.com/science/neural\\_networks](http://www.dkriesel.com/science/neural_networks), 2 2013.
- [25] Ivan Akcheurov. Ntextcat. <http://ntextcat.codeplex.com/>, 3 2013.
- [26] Andrew Kirillov. Aforge.net. <http://www.aforgenet.com/framework/>, 3 2013.
- [27] Steffen Nissen. Fann. <http://leenissen.dk/fann/wp/>, 3 2013.
- [28] Chih-Jen Lin Chih-Chung Chang. Libsvm. <http://www.csie.ntu.edu.tw/~cjlin/libsvm/>, 3 2013.
- [29] Dr. Tie-Yan Liu. Svm and its applications to text classification. [research.microsoft.com/en-us/people/tyliu/atc-lecture--tyliu.ppt](http://research.microsoft.com/en-us/people/tyliu/atc-lecture--tyliu.ppt), 3 2013.