TECHNISCHE HOCHSCHULE LÜBECK

Bachelorarbeit

Dokumentenerkennung für Rechnungsbelege am Beispiel der riscLOG Solutions GmbH

abgegeben von: Christoph Thomas

Abteilung: Elektrotechnik und Informatik

Studiengang: Informatik/Softwaretechnologie

Erster Gutachter: Prof. Dr. Ehlers

Zweiter Gutachter: Prof. Dr. Krause

Anfangsdatum: 15.07.2019

Abgabedatum: 15.10.2019

Arbeitsauftrag

Um die Automatisierung firmeninterner Geschäftsprozesse zu ermöglichen, soll anhand von Machine Learning eine Klassifikation von Rechnungsbelegen stattfinden. Eine zuverlässige Dokumentenerkennung soll letztendlich einem Sachbearbeiter das Ausfüllen eines Formulars abnehmen. Im Laufe dieser Arbeit sollen Probleme analysiert und geeignete Muster zur Problemlösung gefunden werden. Anhand von Trainingsdaten wird ein für dieses Problem geeigneter Klassifikator trainiert, wissenschaftlich evaluiert und für die Verwendung im Betrieb vorbereitet. Der Erkennungsprozess wird in drei Phasen untergliedert: Der Erkennungsprozess wird in drei Phasen untergliedert:

- Mustererkennung: Ein Klassifikator ist in der Lage Datenmuster in den Rechnungsbelegen zu erkennen und einem Belegtyp zuzuordnen.
- Belegextraktion: Auf Basis der Mustererkennung kann eine Extraktion der Informationen vom Beleg durchgeführt werden. Die Positionen der Daten werden durch die Musterkennung ermittelt.
- Evaluation: Der Automatisierungprozess wird evaluiert. Durch Quantifizierung werden fehlerhafte Erkennungen und die durchschnittliche Zeitersparnis ermittelt. Die Ergebnisse sollen visuell dargestellt werden.

TECHNISCHE
HOCHSCHULE
LÜBECK

Department of
Electrical Engineering and Computer Science
Dean's Office - Board of Examiners

Erklärung zur Bachelorarbeit

Ich versichere, dass ich die Arbeit selbst¨andig, ohne fremde Hilfe verfasst habe.

Bei der Abfassung der Arbeit sind nur die angegebenen Quellen benutzt worden.Wörtliche oder dem Sinne nach entnommene Stellen sind als solche gekennzeichnet.

Ich bin damit einverstanden, dass meine Arbeit veröffentlicht wird, insbesondere dass die Arbeit Dritten zur Einsichtnahme vorgelegt oder Kopien der Arbeit zur-Weitergabe an Dritte angefertigt werden.

Lübeck, den October 14, 2019

. . . . . .

(signature)

Zusammenfassung der Arbeit / Abstract of Thesis

Fachbereich:	Elektrotechnik und Informatik
Studiengang:	Informatik/Softwaretechnologie B.Sc.
Thema:	Dokumenten und- Texterkennung von Dokumenten
Zusammenfassung:	Um die Automatisierung firmeninterner Prozesse zu ermöglichen soll anhand von Machine learning eine Klassifikation von Dokumenten stattfinden. Eine Texterkennung soll letzendlich den Nutzer das Ausfüllen eines Formulars abnehmen. Im Laufe dieser Arbeit sollen Probleme analysiert und geeignete Architekturmuster zur Problemlösung verwendet werden. Anhand von Trainingsdaten wird ein für dieses Problem geeigneter Klassifikator trainiert und im Betrieb verwendet. Letztendlich werden Präzisionsergebnisse, die über eine Schnittstelle gesammelt werden, evaluiert.
Autor	Christoph Thomas
Betreuender Professor:	Prof. Dr. Jens Ehlers
WS / SS:	WS 2019/20

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Bei firmeninternen Schadensabwicklungen werden für die Schadensprotokollierung eine Vielzahl von Dokumenten hochgeladen, die auf einem Server gespeichert werden. Die hochgeladenen Belegarten umfassen Rechnungsbelege, aus denen Informationen extrahiert werden können, damit die Daten für zu automatisierende Prozesse zur Verfügung stehen.

Machine Learning ist in vielen Gebieten nicht mehr wegzudenken und gewinnt immer mehr an Bedeutung. Aus Sicht des Kunden ist die Automatisierung der Schadensprotokollierung mit einem ökomischen Gewinn verbunden. Das Ausfüllen von Formularen ist mit einem großen Ressourcenverbrauch der Kunden unserer Plattforum verbunden. Eine automatisierte Belegerkennung mit anschließender Textextrahierung führt zu einer Vergrößerung der Personalkapazitäten externer Unternhmen. Der heutige Stand der Technik ermöglicht es anhand großer Datenmengen eine Lösung für das Problem realisieren. Das Anwenden von Machine Learning leistet einen beachtlichen Beitrag zur Ressourceneinsparung und Unternehmensstrategien, weshalb das Ziel der Unternehmung Gewinne zu maximieren effektiver verfolgt werden kann.

Durch einen großen Datenbestand kann Machine Learning realisiert werden und dafür eingesetzt werden eigenständig Lösungen für Probleme zu finden. Die Nutzung der Technologie ist ein wichtiger Schritt sich gegen konkurriende Unternehmen durchzusetzen und die Kunden an das Unternehmen zu binden. Die Adaption des Unternehmens hinsichtlich auf Machine Learning schafft gegenüber nicht adaptierenden Unternehmen einen großen Wettbewerbsvorteil.

Die vorhandenen Datenbestände sind heutzutage als Wirtschaftsgut angesehen, die jedoch firmenintern nicht kapitalisiert werden. Im Rahmen dieser Arbeit sollen Unternehmen durch einen Anwendungsfall der Eindruck vermittelt werden, dass Datenbestände einen hohen Wert haben und sie genutzt werden sollten.

1.2 Problemstellung

Um die Datenbestände zu nutzen sind einige Vorbereitungen zu treffen um sie in eine geeignete Struktur zu bringen. Des Weiteren muss die richtige Wahl des Klassifikators abhängig von den potentiellen Erkennungsmerkmalen getroffen werden. Sobald der Klassifikator in der Lage ist Rechnungsbelege von anderen Belegen zu unterscheiden, ist eine Textextrahierung abhängig von festgelegten Koordinaten durchzuführen. Da die Koordinaten jedoch sich Unternehmensabhängig unterscheiden, müssen Rechnungsbelege auch untereinander klassifiziert werden. Die Ergebnisse der Klassifikation und Textextrahierung sollen mit quantitativen Gebrauch evaluiert werden um eine zielgenaue Bewertung zu berechnen.

Auf der sicht des Komplexitätsniveaus sollten universell einsetzbare Funktionalitäten gekapselt werden und Unternehmensübergreifend eingesetzt werden können. Es muss eine klare Infrastruktur geschafft werden, die Entwickler nutzen können um die entsprechende Funktionalität nutzen zu können.

1.3 Zielsetzung

Durch das Sortierten der Daten wird der Grundbaustein zum Erstellen eines Klassifkators geschaffen. Da die Daten höchst vertraulich sind werden sie mit höchster Sensibilität behandelt. Der Klassifikator wird unternehmensintern mit einer kleinen Teilmenge trainiert und wird einen prototypähnlichen Zustand haben. Er soll in der Lage sein zwei Belegarten voneinander trennen zu können. Die Klassifikation finden anhand des Textes statt und muss daher von den Bildern extrahiert werden, bevor die Textklassifikation erfolgt.

Die Koordinaten der zu extrahierenen Informationen werden für ein bestimmes Muster für Rechnungsbelege errechnet. Auf Basis der Koordinaten werden bestimmte Regionen aus den Bild rausgelesen und in ein strukturiertes Format gebracht.

Durch eine Kapselung von Klassifkator und Extrahierung können beide jeweile Funktionalitäten eindeutig über eine Schnittstelle angesprochen werden. Zudem sollen die Schnittstellen Ergebnisse liefern, die beispielsweise über das Dashboard aufrufbar sind. Das Dashboard wird sowohl als Schauplatz für den Klassifikations- und Extrahierungsprozess als auch für die visuelle Darstellung von Ergebnissen fungieren.

Die Arbeit soll dem Unternehmen die Welt des Machine Learnings näher bringen und aufrufen das Potential der Daten zu nutzen. Durch eine moderne Architektur können gekapselte und intelligente Komponenten nicht nur Unternehmensintern genutzt werden, sondern geschickt vermarktet werden.

1.4 Grundaufbau der Arbeit

In diesem Unterkapitel wird die grundlegende Struktur der Arbeit kurz zusammengefasst und erläutert, um den Leser einen Überblick über die Inhalte dieser Arbeit zu geben

Kapitel 1 - Grundlagen

Die Grundlagen sollen dem Leser die Kenntnisse vermittelt werden, die es braucht, um die Arbeit grundlegend zu verstehen. Durch das Kapitel wird Basiswissen über Machine Learning, OCR und über die serviceorientierten Architektur vermittelt.

Kapitel 2 - Anforderungen

Die Anforderungen beinhalten die Spezifikationen von sowohl funktionalen als auch technischen Produktanforderungen. Es werden die Anwendungsfälle definiert und anhand von Diagramen illustriert.

Kapitel 3 - Entwicklung des Textklassifikators

Dieser Kapitel befasst sich mit dem gesamten Entwicklungsprozess des Klassifkators einschließlich der Aufbereitung der Daten. Zudem wird die Wahl und Funktionsweise und die Implementierung des Klassifikators erläutert.

Kapitel 4 - Entwicklung der Textextrahierung

Kapitel 5 soll einen Eindruck geben wie die Koordinaten für die Extrahierung errechnet werden und wie daraus erfasste Informationen extrahiert und strukturiert werden. Es wird ein Überblick über die Vorangehensweise der Koordinatenbestimmung der interessanten Regionen und dessen Implementierung verschafft.

Kapitel 5 - Entwicklung des Servers

Die Entwicklung des Servers beschreibt die Architektur und Implementierung der serverseitigen Logik. Das Kapitel behandelt die Kommunikationsschnittstelle zur Textklassifizierung und der Textextrahierung sowie deren Integration.

Kapitel 6 - Entwicklung des Dashboards

In diesem Kapitel wird die Entwicklung des Dashboards dokumentiert. Die Implementierung und die Architektur der clientseitigen Logik werden in diesem Abschnitt erläutert.

Kapitel 7 - Evaluation

Dieser Abschnitt durchläuft die Ergebnissbestimmung der Textklassifikation sowie die durchschnittlichen Zeitersparniss durch die Prozessautomatisierung

Kapitel 8 - Zusammenfassung

In der Zusammenfassung wird dem Leser einen Rückblick der Arbeit verschafft und zukünftige Erweiterungen aufgelistet.

2 Grundlagen

In Diesem Kapitel werden die benötigten Grundlagen soweit erklärt, dass Basiskenntnisse zum Verstehen dieser Arbeit gegeben sind.

2.1 System

Grundlegend stellt das System eine Schnittstelle zwischen Versicherungsunternehmen und Logistikunternehmen dar. Es existiert als Webanwendung und setzt eine Registrierung des Benutzers voraus, um es zu gebrauchen. Benutzer verfügen im Kontext der Webanwendung über bestimmte administrative Rechte, die mittels einer Rollenvergabe organisiert werden.

Bei Erfassen eines Schadens seitens des Spediteurs wird für ihm eine Schadensakte angelegt. Eine Funktion in der Schadensakte beinhaltet das Hochladen von Nachweisen für die Schadensprotokollierung, darunter Rechnungsbelege, die es zu erkennen gilt.

2.2 Maschinelles Lernen

Maschinelles Lernen beruht auf den Gebiet der Mustererkennung, die sich mit der automatischen Erkennung von Regelmäßigkeiten in Daten unter Verwendung von Algorithmen befasst. Durch das Erkennen von Regelmäßigkeiten werden Maßnahmen zur Klassifizierung der Daten in verschiedene Kategorien ergriffen[Bis06].

Maschinelles Lernen ist die Wissenschaft Computer so zu programmieren, sodass sie in der Lage sind von Daten zu lernen und ist bereits eine weiterforschte Technologie, die Verwendung in hochtechnologischen Produkten findet. Sie ist verantwortlich für die Realisierung von unter anderem die Spracherkennung in Smartphones, das Empfehlen von Videos auf Videoportale und das Ranking von Suchergebnissen im Internet-Suchmaschinen[Ger17].

Eine der ersten populärsten Anwendungsfälle von Maschinelles Lernen ist der Filter um Spam-Mails zu erkennen. Der Spamfilter ist dazu in der Lage seriösen Mails von Spam-Mails zu unter-

scheiden und als solche zu kennzeichnen. Eine dementsprechende Intelligenz setzt voraus, dass der Software beigebracht werden muss wie es eine Spam-Mail als solche erkennt.

2.2.1 Überwachtes Lernen

Damit ein intelligenter Filter in der Lage ist Spam-Mails zu erkennen, muss er die korrespondierenden Regelmäßigkeiten kennen. Das notwendige Wissen eignet sich die Software über ein Verfahren im Maschinellen Lernen an: Es wird eine Vielzahl von Daten benötigt, die zum Trainieren der Software dient, damit sie Charakteristiken von Spam-Mails erkennen kann. Dieser Datensatz nennt sich Trainingssatz und wird zum Einstellen von Parametern eines adaptiven Modells benötigt. Im Fall des Spamfilters gibt es zwei Zielmerkmale:

- Es handelt sich um eine Spam-Mail
- Es handelt sich um keine Spam-Mail

Spam-Mails weisen Regelmäßigkeiten auf, die sie von seriösen Mails unterscheiden. Ein sehr prägnantes Merkmale einer Spammail ist der Inhalt der Nachricht. Wörter wie *kostenlos, schnelles Geld, reich werden, risikofrei* und *hier klicken* sind eine typische Charakteristiken einer Spam-Mail*[Spy03].*

Das Modell adaptiert Erkennungsmuster anhand von den Trainingssatz und kann bei neuen Daten anhand der Charakteristiken das Zielmerkmal prognostizieren.

2.3 Natural Language Processing

Das Initialwort NLP beschreibt die Maschinelle Verarbeitung natürlicher Sprache und ist eine Technologie die einen Bereich des Machine Leaning darstellt. Die Motivation hinter dieser intelligenten Technologie ist die Interaktion zwischen Mensch und Computer unter Verwendeung der natürlichen Sprache in sowohl schriftlicher als auch mündlicher Form. Die Kategorisierung von Text findet auf erlernbare syntaktische und semantische Erkennungsmuster statt [Ron11].

Die Wahl der Technik ist auf den Anwendungsfall abgezielt, so werden in einigen Fällen beispielsweise Wortfrequenzen gezählt, um verschiedene Schreibstile zu vergleichen wobei in einem anderem Kontext muss der vermeintliche Inhalt einer menschlichen Äußerung verstanden werden[Ste09]. Die Technologie wird sowohl im Bereich der Texterkennung als auch in der Spracherkennung verwendet. Die Entwicklung der Merkmale (*feature engineering*) findet kontextabhängig auf Syntax und Semantik der Sprache statt. NLP wird heute in in vielen Bereichen eingesetzt.

Die inhaltliche Deutung von NLP findet ihre Grenze bei Informationen die ein höheres Abstraktionsniveau haben, so kann beispielsweise sarkastischer Inhalt nicht richtig gedeutet werden.

Syntaktische Analyse

Die Syntax ist die Anordnung der Wörter, so dass sie grammatikalisch einen Sinn ergeben. Im Bereich des NLP kann eine syntaktische Analyse durchgeführt werden, die überprüft wie die natürliche Sprache mit den grammatikalischen Regeln überstimmt. Auf diese Weise kann festgestellt werden ob der eingegebene Text ein Satz in der natürlichen Sprache ist[Cen01].

Semantische Analyse

Die Semantik bezieht sich auf die Bedeutung von Texten. Mit einem Algorithmus werden grammatikalische Regeln auf einen großen Textkorpus[1] angewendet um daraus eine kontextbezogene Bedeutung von Wörtern abzuleiten.

Die Kernidee ist, dass bei der Betrachtung aller Wortkontexte nach dem Vorkommen oder Fehlen eines bestimmtes Wortes gesucht wird, um die Ähnlichkeit der Bedeutung von Wortgruppierungen untereinander zu bestimmen[Lan98]. Bestimmte Wörter treten mit anderen bestimmten Wörtern in einem Kontext auf, womit ein inhaltlicher Sinn hergeleitet werden kann.

2.4 Dokumentenanalyse

Der Bereich der Forschung, der sich mit der Erkennung von Text- und Grafikkomponenten in Bildern von Dokumenten beschäftigt, heißt *Document Image Analyse* (DIA). DIA bezeichnet eine große Gruppe von Techniken, die visuelle Informationen charakterisieren können und gehört zu den ältesten Bereichen der Informatik. Die Motivation in diesem Forschungsgebiet ist die automatische Extraktion von Informationen auf die Art und Weise wie ein Mensch die abgebildeten Informationen aufnehmen würde[Ran02].

Es gibt zwei Kategorien mit je zwei Komponenten im Gebiet der Dokumentenanalyse:

1. Texterkennung
    a) Seitenlayoutanalyse - Erkennen von Spalten, Paragraphen, Textlinien und Wörter
    b) Optische Zeichenerkennung - Erkennung von Zeichencharakteren
2. Grafikerkennung

[1] Sammlung von Texten

a) Linienverarbeitung - Erkennen von Polygonen wie Linien, Ecken und Kurven

b) Region- und Symbolverarbeitung Erkennung von gefüllten Regionen

Im Rahmen dieser Arbeit ist die Motivation eine Texterkennung durchzuführen, weshalb sich ausschließlich auf diesen Zweig eingegangen wird.

2.4.1 Seitenlayoutanalyse

Die Analyse des Seitenlayouts ist der erste Schritt der Dokumentenerfassung, um *Regions Of Interests*[2] ausfindig zu machen. Dabei werden im Dokument bestimmte Bereiche erkannt und beschriftet, um eine kontrollierte Extraktion der darin enthaltenen Informationen durchzuführen. Die Bereiche variieren nach dem Aufbau des Dokumentenlayouts, weswegen die Unterteilung in Blöcke, Illustrationen, Symbole oder Tabellen durchgeführt wird[Bre03].

2.4.2 OCR

OCR (Optical Character Recognition) ist eine Technologie zur Erkennung von gedruckten oder handgeschriebenen Textzeichen in digitalen Bilder von physischen Dokument. Die Motivation von OCR ist die Identifizierung von spezielle Anordnungen von Pixeln als Buchstaben in einer Rastergrafik. Mittels dieser Technologie können von beispielsweise gescannten Dokumenten die Textzeichen mit Rücksicht auf die Kompositionen der Zeichen ausgelesen werden. Der Kernprozess ist die Umwandlung von erkannten Textzeichen in eine digitalisierte Rekonstruierung des Textes.

Durch eine Infrastruktur zwischen Hardware, wie beispielsweise einem Scanner und einer Texterkennungssoftware können Informationen digital verarbeitet werden. Eine populärer Anwendungsfall ist die Digitalisierung von historischen Dokumenten in physischer Form, da der abgebildete Text lesbar rekonstruiert werden kann und im Internet zur Verfügung gestellt wird.

Texterkennung ist eine Technologie die auf Maschinelles Lernen und Mustererkennung basiert. Einer OCR-Software werden mit Textbeispielen in verschiedenen Schriftarten und Formaten trainiert, so dass die Software in der Lage ist durch Merkmale, wie die Anzahl und Anordnungen der Linien und Kurven, Zeichen zu erkennen.

[2]Bereiche die Informationen beinhalten

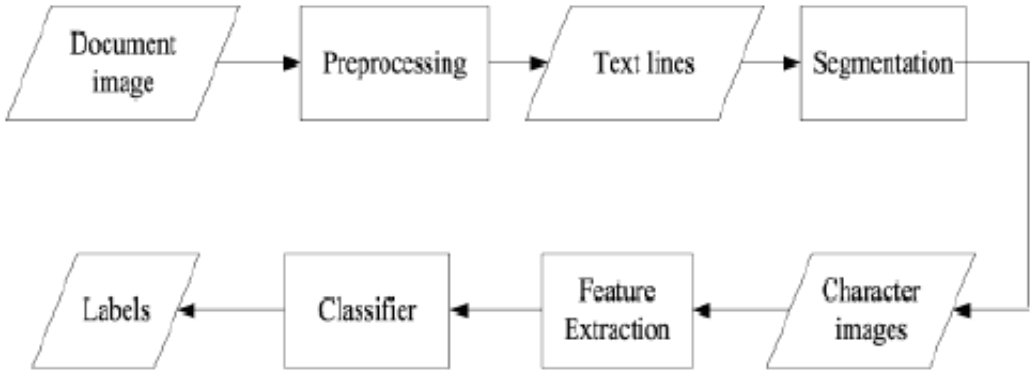

Abbildung 2.1: System von OCR (Quelle: [Sul10]

System von OCR

Das gescannte Dokument muss zunächst Vorverarbeitet werden, indem ein Schwellwert gesetzt wird, um bestmöglich die dunkleren Pixel von den helleren zu trennen. Das Binarisieren ist notwendig, da somit der Text sich klar abhebt und damit ein besserer Erkennungsprozess ermöglicht wird. Eine Texterkennungssoftware ist in der Lage Textzeichen zu erkennen und sie zu segmentieren, damit sie als Einzelne untersucht werden können. Eine Textzeile beinhaltet Buchstaben, die als Bilder extrahiert und klassifiziert werden. Dieser Fluss ist in Abbildung 2.1 veranschaulicht und zeigt insgesamt 8 Schritte im Stil eines Flussdiagramms.

2.4.3 Tesseract

Tesseract ist eine Texterkennungssoftware und wurde im Jahre 2007 in der *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)* von Ray Smith veröffentlicht[EE07]. Sie ist zwischen 1984 und 1994 von dem Unternehmen *Hewlett-Packard*[3] entwickelt wurden.

HP hatte den Fokus auf OCR und erzielten mit Tesseract sehr gute Zielgenauigkeiten bezüglich von textbasierten Erkennungsmustern, obwohl sie Tesseract nicht als Produkt vermarktet haben. 1995 erzielte Tesseract mit seinem Ergebnissen den dritten Platz auf der UNLV - Einem jährlichen Test zur Messung von Genauigkeiten im Bereich von OCR und seitdem nicht mehr weiterentwickelt wurden. Ray Smith, der bei HP an Tesseract mitentwickelte, arbeitete mittlerweile bei Google und brachte 2005 Tesseract, wessen Quelltext im selbigen Jahr öffentlich gemacht wurden ist, auf den neusten Stand der Technik[SN95].

Tesseract wird im Rahmen dieser Arbeit als Texterkennungssoftware benutzt, um Texte zu

[3]https://www8.hp.com

erkennen. Die neuste Version von Tesseract[4] arbeitet mit einem rekurrenten Neuronalen Netz zur Klassifizierung von Charakteren.

2.4.4 Rekurrente Neuronale Netze

Rekurrente Neuronale Netze eignen sich besonders gut für bildbasierte Sequenzerkennung. Anders wie bei normalen neuronalen Netzen, die nützlich bei der Erkennung von einzelnen Objekten, die sich nicht korrelativ zueinander verhalten, sind rekurrente neuronale Netze auf das Erkennen von sequenzähnlichen Objekten ausgelegt. Das rekurrente neuronale Netzwerk ist ein weiterer wichtiger Zweig der Familie von tiefen neuronalen Netzwerken und sind hauptsächlich für die Klassifizierung von Sequenzen konzipiert [Has14].

Das Klassifizieren von einer Reihe aus Objekten ist ein Problem, welches darauf basiert Bildsequenzen zu erkennen. Bei dem Erkennungsprozess können die Objekte von der Länge stark variieren. Im Bereich von Texterkennung gibt es Wörter die wie “Ja”, die lediglich 2 Zeichen aufweisen, während “Allgemeinmedizin” 16 Zeichen aufweist. Die Anzahl der Kombinationen aus Zeichen zu

Sequenzen von beispielsweise chinesischen Zeichen, Musiknoten und Wörter kann größer als 1 Million sein, weshalb ein normales neuronales Netz durch die hohe Anzahl der Klassen unbrauchbar wäre.

2.5 Serviceorientierte Architekturen

Serviceorientierte Architekturen oder abgekürzt SOA bezeichnet eine moderne Architektur, dessen Kerngedanke darin besteht komplexe Strukturen aufzulösen. SOA ermöglicht eine Modernisierung der IT-Struktur des Unternehmens, da Funktionalitäten in geschäftsorientierte Serviceblöcke gekapselt werden[Pat09].

Das Kernkonzept besteht aus dem Anbieten, Suchen und Nutzen von Diensten über ein Netzwerk. Diese Dienste werden plattformübergreifend von Applikationen genutzt und sind unabhängig von der jeweiligen Software[Mel10]. Die Serviceorientierte Architektur bietet daher universell anwendbare Dienste, die bei Bedarf von jeder Applikation angesprochen werden kann. Die Geschäftslogik der Dienste können zweckbedingt mit der Programmiersprache geschrieben werden, die eine optimale Lösung bietet.

Eine Anfrage auf die Kommunikationsschnittstelle des Dienstes wird von der Geschäftslogik verarbeitet und wieder als Antwort über die Schnittstelle zurückgegeben. Der Vorteil gegenüber

[4] https://github.com/tesseract-ocr/tesseract/wiki/4.0-with-LSTM

dem Konzept ist eine Betrachtungsweise auf den Dienst, der nicht nur auf Entwickler begrenzt ist. Die Simplizität die durch das Konzept der serviceorientierten Architektur ermöglicht wird erbringt eine Zusammenarbeit von Entwickeln und dem Management des Unternehmens hinsichtlich der Organisation von Diensten.

2.5.1 Webservices

Werbservices stellen die modularen, gekapselten Komponenten der SOA dar. Jede Webservice-Komponente beinhaltet nur die für ihren Zweck bestimmte Geschäftslogik und eine kontextorientierte Schnittstelle, die es bei Bedarf anzusprechen gilt. Im Verbund mit anderen Webservice-Komponenten werden im Kontext einer Applikation Funktionalitäten bereit gestellt.

Das Abstrahierungskonzept von SOA sieht vor, dass Die Schnittstelle der Webdienste wird von außen hin betrachtet wird. Es sind daher nur spezifizierte Dienstanbieter für Wartung, Betrieb und Infrastruktur des jeweiligen Dienst verantwortlich, weswegen eine Kapselung der Gesamtkomplexität unter den einzelnen Dienstanbietern möglich ist. Des Weiteren sind Dienstanbieter für die Authentisierung und Authentifizierung und der Aufrechterhaltung zuständig. Dienstnutzende müssen sich als solche identifizieren bevor sie aus den Webservice zugreifen können.

In einer Schnittstellenbeschreibung werden Servicebeschreibungen veröffentlicht, die im des Servicekontext die Kommunikationsmöglichkeit der Schnittstelle beschreibt. Konkret müssen dem Service nötige Informationen geliefert werden um die Anfrage in der Geschäftslogik zu bearbeiten. Die Schnittstellenbeschreibung listet die möglichen Antwortmöglichkeiten für eine an die Service gerichtete Anfrage auf. Der Dienstnutzende hat somit den Vorteil zu wissen wie der Webservice sich verhält.

2.5.2 REST

Der Begriff REST steht für *Representional State Transfer* und ist von Roy Fielding in seiner Dissertation *Architectural Styles and the Design of Network-based Software Architectures* geprägt wurden[Fie00]. Die REST-Schnittstelle ist ein weitverbreiteter Architekturansatz, der dazu verwendet wird um eine Infrastruktur zwischen Client und Server zu etablieren.

Die REST-Architektur ermöglicht, dass Ressourcen jeweils über *Webservices* mit *Anfragen* (requests) gefordert werden können. Jeder Webservice verfügt über eine *Uniform Ressourcer Locator* (URL) und lässt sich damit eindeutig identifizieren. Der Server schickt daraufhin eine *Antwort* (response), die die geforderten Ressourcen enthält. Da eine REST-Schnittstelle zustandslos ist und der Server keine gespeicherten Daten zurückgreifen kann, müssen bei einer Anfrage alle nötigen

Methode	Grundsatz
GET	Beschaffung von Informationen über eine Ressource
POST	Erstellen einer Resource
PUT	Aktualisieren einer Resource
DELETE	Löschen einer Resource

Tabelle 2.1: HTTP/s-Anfragemethoden

Informationen vom Client geliefert werden. Die Clientseitige Anfragen basieren auf dem *HTTP/s-Protokoll* und verwenden die damit zusammenhängenden *Anfragemethoden* (request methods), die in der Tabelle 2.5.2 aufgeführt sind.

Um beispielsweise eine *Ressource User* zu erstellen muss eine Anfrage an einen Endpunkt gerichtet werden, der die Funktionalität für die *POST-Anfrage* zur Verfügung stellt. Der Endpunkt würde über die URL *.../users* erreichbar sein. Die REST-Schnittstelle würde die Anfrage entgegennehmen und an die zuständige Geschäftslogik weiterreichen. Die entsprechende Schnittstellenarchitektur wird im REST-Schnittstelle im Kapitel 6 erläutert.

2.6 Entwicklungsumgebung

2.6.1 Jupyter Notebook

Jupyter[5] Notebook bietet die Möglichkeit in einer interaktiven Umgebung Programmcode parallel durchlaufen zu können. Es können Ergebnisse als Live-Berechnung oder im festen Format an die Zellen gespeichert und weitergereicht werden. Dieser Vorteil bietet eine gekapselte Ausführung von Code in den jeweiligen Zellen. Jupyter eignet sich gut Für Projekte in den Bereichen *Data Science*[6] und Maschinellem Lernen, da Ergebnisse durch die Zellenweise Ausführung gut nachverfolgen und visualisieren lassen.

[5] https://jupyter.org/
[6] Bereich der Datenwissenschaft

3 Konzept der Software

Dieses Kapitel erklärt das Konzept des Systems. Zunächst werden im Unterkapitel 3.1 Anforderungen anhand eines Use-Case-Diagrams definiert, die das Vorgehen in der Entwicklung erkennbar macht um einen abschließenden Lösungsansatz zu konzipieren.

3.1 Anwendungsfälle

In der Abbildung 3.1 wird das Use-Case-Diagram veranschaulicht, aus welchem die funktionalen Anforderungen abgeleitet werden sollen. Der Webclient wird dazu verwendet über das UI[1] eine Datei hochzuladen. Sollte diese Datei das Bild oder PDF-Format haben, wird das Bild von einem Service, der Funktionalität zum Erkennen eines Belegtyps kapselt, als entweder Schadensfoto oder Rechnungsbeleg klassifiziert. Nachdem festgestellt wurden ist, ob es sich bei der hoch geladenen Datei ein Rechnungsbeleg handelt, extrahiert ein Service, der darauf ausgelegt ist bestimmte Regionen auszulesen, Informationen aus dem Bild.

Anwendungsfälle bezüglich der Verwaltung und Authentifizierung von Nutzern werden im weiteren Verlauf über den Unternehmenseigenen IAM-Server abgewickelt und werden in diesem Kontext nicht berücksichtigt.

3.2 Funktionale Anforderungen

In diesen Unterabschnitt werden die funktionalen Anforderungen aus dem Anwendungsszenario geschlussfolgert und sind etappenweise in drei Bereiche gekapselt, um sie weiterhin zu modularisieren. Die funktionalen Anforderungen sind abhängig vom Szenario gegliedert und tabellarisch aufgeführt. Die Anforderungen werden aus der Sicht des Webclients definiert, um die Sicht des Benutzers zu veranschaulichen.

Rechnungsbeleg erkennen Ein Webclient soll in der Lage sein Dateien hochzuladen und sie an den Server zu senden, um die gewählte Datei zu klassifizieren. Der Webclient erhält daraufhin

[1] User Interface (Benutzeroberfläche)

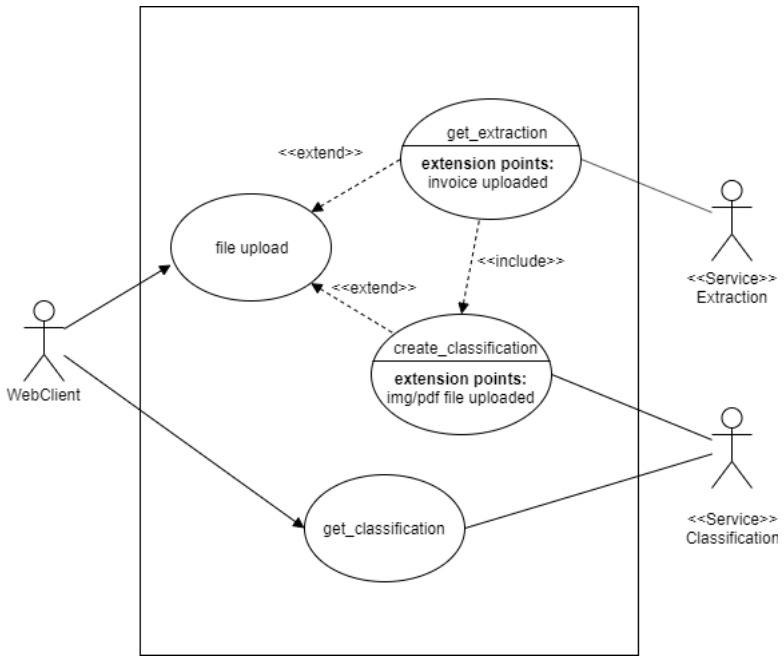

Abbildung 3.1: Use-Case-Diagram

eine entsprechende Antwort vom Server. Der Webclient reagiert auf die Antwort und stellt sie repräsentativ dar. Die Antwort beinhaltet vor allem das Klassifkationsergebnis. Die Tabelle 3.2 zeigt insgesamt 4 technische Anforderungen an den Klienten.

Wenn ein Rechnungsbeleg erkannt wurde, wird dem Nutzer eine Extraktion der Informationen angeboten. Die funktionalen Anforderungen zum Anwendungsszenario werden in Tabelle 3.2 aufgelistet und beschrieben..

Informationen extrahieren Der Webclient sendet das zuvor hochbeladene Bild, vorausgesetzt es ist ein Rechnungsbeleg, an den Server. Der Client erhält eine Antwort die die extrahierten Daten enthält und sie deklarativ dem Benutzer präsentiert.

Die ersten beiden Szenarien treten in diesem Kontext zusammenhängend auf, da eine Extraktion lediglich für Rechnungsbelege gedacht ist. Aufgrund von universeller Einsetzbarkeit und Modularität werden sie als zwei Webservices implementiert. Mit der Servicearchitektur können die beiden Komponenten serverseitig komplett voneinander gekapselt werden.

Evaluationsergebnisse im Dashboard präsentieren Beim Aufruf des Dashboards wird eine GET-Anfrage an eine URL gesendet, um die Evaluationsergebnisse abzufragen. Das Dashboard bekommt die Antwort als Datenformat und präsentiert sie auf der UI mittels Statistischen Modellen.

ID	Beschreibung
F-10010	Die UI kann eine Bilddatei auswählen
F-10020	Die UI kann die Bilddatei über POST-Anfrage senden
F-10030	Der Server kann Anfrage entgegennehmen
F-10040	Der Server kann Anfrage validieren
F-10050	Der Server kann Bild dekodieren
F-10060	Der Server kann Bild klassifizeren
F-10070	Der Server kann entsprechende Antwort schicken
F-10080	Die UI kann eine Antwort empfangen
F-10090	Die UI kann die Antwort präsentieren

Tabelle 3.1: Klassifikation von Dokumenten

ID	Beschreibung
F-20010	Die UI kann Beleg über POST-Anfrage senden
F-20020	Der Server kann Anfrage entgegennehmen
F-20030	Der Server kann Anfrage validieren
F-20040	Der Server kann Bild dekodieren
F-20050	Der Server kann Informationen extrahieren
F-20060	Der Server kann entsprechende Antwort schicken
F-20070	Die UI kann eine Antwort empfangen
F-20080	Die UI kann die Antwort tabellarisch präsentieren

Tabelle 3.2: Informationen aus Rechnungsbeleg extrahieren

ID	Beschreibung
F-30010	Die UI kann eine GET-Anfrage senden
F-30020	Die UI kann eine Antwort empfangen
F-30030	Die UI kann eine Antwort über Diagramme darstellen

Tabelle 3.3: Evaluationsergebnisse präsentieren

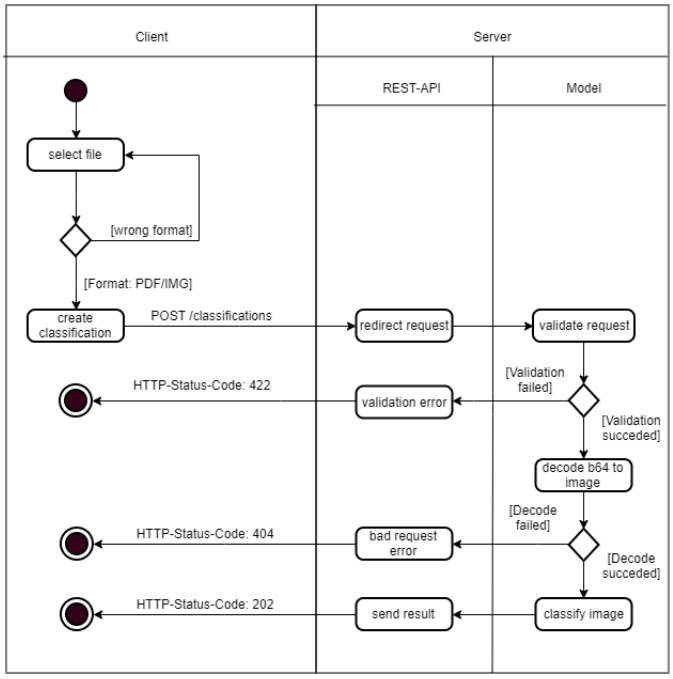

Abbildung 3.2: Aktivitätsdiagramm Klassifikation

3.2.1 Aktivitätsdiagramme

In diesen Unterabschnitt befinden sich die einzelnen Aktivitätsdiagramme. Die Aktivitätsdiagramme zeigen die aufgelisteten funktionalen Anforderungen in Ablaufsequenzen, wo Client und Server als Hauptakteure miteinander interagieren, wobei im Server untergeordnet die Kommunikationsschnittstelle mit dem Model interagiert. Das System ist nach dem *MVC-Ansatz*[2] aufgebaut[Ber09]. Die View ist der Client und ist zuständig für die Darstellung des Modells. Der Controller ist die REST-Schnittstelle, da sie die Interaktionen vom Benutzer über die View entgegennimmt und weiterleitet. Das Model enthält die Geschäftslogik, die die Funktionalitäten für die Bearbeitung der Interaktionen zuständig ist.

Klassifikation

Das Aktivitätsdiagramm 3.2.1 zeigt die funktionalen Anforderungen *F-10010* bis F-10090 in einer Ablaufsequenz.

Extraktion

Das Aktivitätsdiagramm 3.2.1 zeigt die funktionalen Anforderungen *F-20010* bis F-20080 in einer Ablaufsequenz.

[2] *Model-View-Controller*

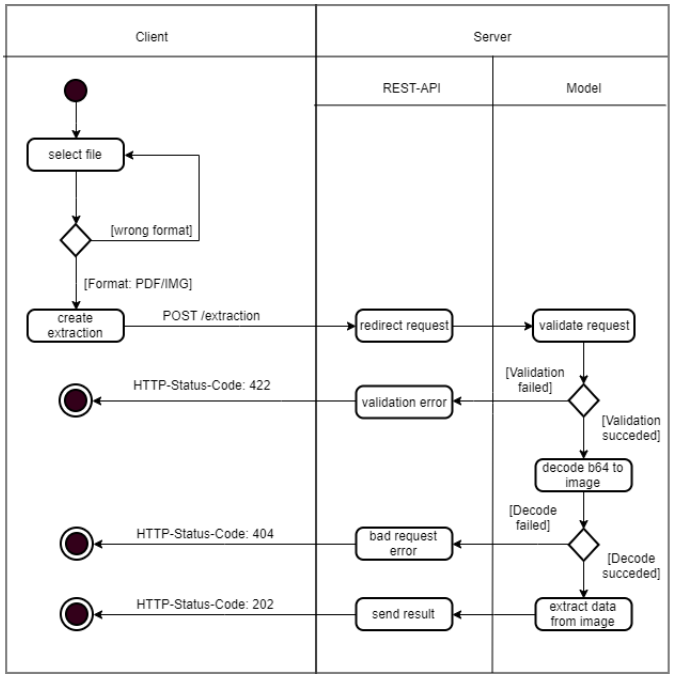

Abbildung 3.3: Aktivitätsdiagramm Extraktion

Evaluation

Das Aktivitätsdiagramm 3.2.1 zeigt die funktionalen Anforderungen *F-30010* bis F-30030 in einer Ablaufsequenz.

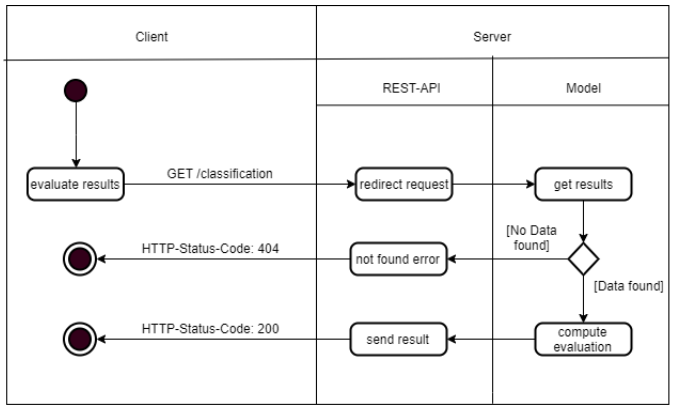

Abbildung 3.4: Aktivitätsdiagramm Evaluation

4 Entwicklung des Textklassifikatoren

Dieses Kapitel befasst sich mit der Entwicklung des Klassifikators und ist in den zwei Kapiteln Architektur und Implementierung gegliedert.

4.1 Datenanalyse

Unter der Organisation der Daten ist sowohl die Formatierung als auch die Strukturierung von den zur Verfügung stehenden Datensätzen gemeint. Um das Model mit Überwachtem Lernen zu trainieren müssen die Daten so strukturiert sein, dass die Kategorie von jeden Datenpunkt bekannt ist. Ursprünglich sind die Bilder im PDF-Format in einer Ordnerstruktur gegliedert gewesen, die abhängig vom Hochladedatum nach Jahr, Monat und Tag sortiert wurden sind. Die Zeitangabe für die Textextraktion ist als unrelevant zu werten, da hier nur auf der inhaltliche Aspekt eine Rolle spielt.

Kollektion

Mit dem Programmcode sind alle Dateien kollektiviert, indem rekursiv über jeden Ordner in einem bestimmten Ordner *input_dir* alle PDF-Dateien erfasst wurden und in den Ordner *output_dir* verlagert wurden sind. Die Generierung der Merkmale verläuft über das Extrahieren der Texte aus Bilddateien, weswegen jedes Bild mit der Funktion *convert_from_path()* der externen Bibliothek *pdf2image* zum JPG-Format konvertiert wurden ist.

```
1 def collect_data(input_dir, output_dir):
2     ext = 'jpg'
3     for pdf in glob.glob('{inp}/*/**/*.pdf'.format(inp=input_dir),
          recursive=True):
4         try:
5                 pages = convert_from_path(pdf)
6                 except (ValueError, Exception):
7                 pass
8
```

```
9     for page in pages:
10          page.save('./{output}/{name}.{ext}'.format(
11              output=output_dir, name=str(uuid.uuid4())),  ext=ext)
```

Listing 4.1: Sammeln der Daten

Die rekursive Suche verlief über glob (*Unix style pathname pattern expansion*), einer Bibliothek die alle Pfadnamen nach einem spezifischen Muster nach Unix-Regeln durchsucht. Die Funktion *convert_from_path()* gibt eine Liste Bild-Objekte zurück, da PDF-Dokumente eine Mehrzahl an Seiten aufweisen können.

Die Betrachung der Bilder zeigt, dass es sowohl digitalisierte Aufnahmen von Sowohl Schadensfotos als auch Belegen aller Art gibt. Es wurden zwei Belegarten selektiert, die es zu klassifizieren gilt:

- Rechnungen
- Regressforderungen

Aus Zeitgründen werden hier lediglich zwei Kategorien betrachtet, wobei die eigentliche Menge an Daten dafür ausreicht eine Vielzahl von Belegarten zu kategorisieren.

Sortierung

Eine interessante Möglichkeit die Bilder zu Sortieren ist nach Schlüsselwörtern in den Texten der Dokumente zu suchen und sie abhängig davon in bestimmte Ordner zu sortieren. Durch eine Textextraktion von jedem Bildes gelang es die Texte nach bestimmten Schlüsselwörtern zu durchsuchen und sie danach zu sortieren. Da eine große Variation an Daten vorhanden ist, sind mitunter auch Ausreißer unter den sortierten Daten dabei.

Ausreißer üben eine negative Korrelation auf die Zielsicherheit des Klassifikator aus. Durch stichprobenartiges herüberschauen ist aufgefallen, dass die Menge an Ausreißern überschaubar ist. Die Sortierung hat dazu geführt, dass insgesamt 700 Rechnungsbelege und 700 Regressforderungen identifiziert und kategorisiert wurden.

4.2 Vorbereitung der Daten

Um Die Daten für die Klassifizierung vorzubereiten mussten die Daten in einem bestimmtes Format gebracht werden. Im Programmcode wird vom Ordner, der durch den Parameter *input_dir* projiziert ist, von der Funktion *extract_text()* der Text extrahiert.

```
def extract_text(im):
        im = Image.open(im)
        extracted = pytesseract.image_to_string(im, output_type=Output.
            DICT, lang='deu')

        return extracted

def doc_to_list(input_dir, label):
        res = []
        ext = "jpg"
        images = len(glob.glob('{dir}/*.{ext}'.format(dir=input_dir, ext=
            ext))
        for im, i in zip(images, range(len(images))):
        extracted_text = extract_text(im)
                extracted_text['id'] = i + 1
                extracted_text['label'] = label
                res.append(extracted_text)

        return res
```

Listing 4.2: Formatieren der Daten

Die Methode *doc_to_list* fügt dem Dictionary, welches durch *image_to_string()* zurückgegeben wurden ist, eine ID und die Kategorie hinzu und gibt eine Liste von Dictionaries zurück.

Mit dem Aufruf *json.dump()* wurden die Listen in ein Json-Array überführt und in einer Json-Datei zwischengespeichert um sie zu persistieren. Mit *Pandas*[1] wurden die Json-Dateien wiederrum in einem Dataframe[2] überführt. Ein Dataframe ist eine einer Zweidimensionale tabellarischen Datenstruktur, die sich gut visualieren lässt und sich somit gut für Datenanalysen eignet.

```
with open('./data/rechnungsbeleg/invoice_data.json', 'w') as json_file:

        json.dump(list_rb, json_file)
with open('./data/regress/regress_data.json', 'w') as json_file:
        json.dump(list_rg, json_file)
```

Listing 4.3: Persistieren der Daten

[1] *https://pandas.pydata.org/*
[2] https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html

Die jeweiligen Dataframes werden konkateniert, was dazu führt, dass sich alle Texte mit ihrer zugehörigen Kategorie in einem Dataframe befinden.

```
1 df_inv = pd.read_json("./data/rechnungsbeleg/invoice_data.json")
2 df_re = pd.read_json("./data/regress/regress_data.json")
3 df = pd.concat([df_inv, df_re])
```

Listing 4.4: Lesen der Json-Dateien

Die Daten wurden mit diesem Vorgang in das Programm geladen und stehen für weitere Prozeduren zur Verfügung. Die Texte wurden durch die Vorbereitung in eine übersichtliche Struktur verlagert, die es erlaubt die Texte im Programm kontrolliert aufzubereiten.

4.3 Aufbereitung der Daten

Die Texte befinden sich momentan noch im Rohzustand, was bedeutet, dass sie für das Training ungeeignet sind. Im jetzigen Zustand der Daten befinden sich Unregelmäßigkeiten in den Daten, was zu einem deutlich Qualitätsverlust führt. Beispielsweise beinhalten die Texte willkürliche Zahlen, Zeilenumbrüche und unbrauchbare Werte wie Maßeinheiten. Die Unregelmäßigkeiten beschränken sich nicht auf diesen Anwendungsfall und treten in verschiedenster Form auf.

Der Bereinigungsprozess wird gut vom Machine Learning Team *Azure* von Microsoft[3] dokumentiert. In der Dokumentation wird beschrieben, dass es drei Kriterien von minderwertigen Daten gibt: Die Daten können unvollständig durch fehlende Attribute und Werte, überflüssig durch fehlerhafte Datensätze und Ausreißer und inkonsistent durch wiedersprüchliche Datensätze sein. Für die Bereinigung von Texten trifft folgendes Zitat zu:

[...] *Entfernen eingebetteter Zeichen, die zu einer falschen Datenausrichtung führen können, z. B. eingebetteter Tabstopps in tabstoppgetrennten Dateien oder eingebetteter Zeilenumbrüche, die Datensätze unterbrechen könnten* [...][4]

4.3.1 Bereinigung der Daten

Zum Bereinigen der Daten werden auf Text ausgerichtete Verfahren[V S10] angewandt:

[3]https://docs.microsoft.com/de-de/azure/machine-learning/team-data-science-process/prepare-data
[4]https://docs.microsoft.com/de-de/azure/

Stopwörter

Stopwörter sind Wörter die in einer Kollektion von Dokumenten so häufig vorkommen, dass es kein Sinn macht sie in das Training mit einzubinden, da sie keine Relevanz auf Dokumentinhalte enthalten. Typische Wörter sind Artikel, Konjunktionen und Präpositionen. Abhängig von den Inhalten der Dokumente können spezifische Stopwörter hinzugefügt werden.

Lemmatisierung

Es wird eine Reduktion der Wortformen auf ihre Grundform vorgenommen. Dieser Vorgang wird mithilfe eines digitalen Lexikons realisiert.

Stemming

Stemming ist ein Verfahren womit Suffixe von dem jeweiligen Wort zu entfernen. Suffixe sind generell auf die Englische Sprache reduziert, weshalb diese Algorithmen nur für englischsprachige Dokumente eingesetzt werden kann.

Daneben können verschiedene Algorithmen angewandt werden, wie das ersetzen der Großbuchstab durch Kleinbuchstaben oder das Entfernen von Sonderzeichen.

Die Funktion *normalize()* im Programmcode wendet verschiedene Algorithmen auf die Wörter des Textes an, um die Daten zu normalisieren.

```
1 def normalize(df):
2         # remove special characters
3         df['text'].apply(lambda x: re.sub("(\\W)+"," ",x))
4         # remove punctuation
5         df['text'].apply(lambda x: re.sub(r'[^\w\s]', '', x))
6         # tokenize
7         df['text'].apply(lambda x: nltk.word_tokenize(x))
8         # to lower case
9         df['text'].apply(lambda x: [word.lower() for word in x])
```

Listing 4.5: Bereinigung der Daten

Mit zwei Regex-Ausdrücken wird der Text auf Sonderzeichen und Zeichensetzung gefiltert. Mit der NLP-Bibliothek *nltk* wird der Text in eine Liste von Tokens konvertiert. Dadurch kann jedes Wort iteriert werden um Großbuchstaben in Kleinbuchstaben umzuwandeln. Lemmatisierung wird nicht eingesetzt, da es die Genauigkeit verringert.

4.3.2 Trainings- und Testdaten

Die Daten werden mit der Funktion *train_test_split()* der Bibliothek *sklearn*[5], die allgemein Funktionen für den Bereich des Data Science zur Verfügung stellt, in Trainings- und Testmengen unterteilt.

```
1 X_train, X_test, y_train, y_test = train_test_split(df['text'], df['label
    '], test_size=0.2)
```

Listing 4.6: Unterteilung der Daten

Als Parameter nimmt die Funktion die Beispieltexte und die Kategorien als Listen, somit kann die Funktion die Texte zu ihrer jeweiligen Kategorie zuordnen. Über den Parameter *test_size* wird das Verhältnis von den Teilmengen der Trainings- und Testdaten festgelegt. Die Beispieltexte werden gemäß dem Faktor 0.2 in 20 Prozent Testdaten und 80 Prozent Trainingsdaten unterteilt.

4.4 Erstellung des Klassifikationsmodells

4.4.1 Vokabular definieren

Der CountVektorizer[6] von sklearn bietet eine einfache Möglichkeit Texte in Token zu formatieren und dadurch ein Vokabular bekannter Wörter zu erstellen. Durch Übergabe der Trainingsdaten werden die Aufkommen jedes Wortes gezählt. Zusätzlich können Stopwörter und Token Muster übergeben werden, um die Zählprozedur zu beeinflussen.

Die Bibliothek nltk bietet die Möglichkeit sprachabhängig definierte Stopwörter als Liste im Programm zu nutzen. Diese werden durch benutzerdefinierte Stopwörter ergänzt.

Die Konstante *token_pattern* definiert einen Regulären Ausdruck mit dem Zweck, dass numerische Token nicht mit in die Zählung einfließen.

```
1 nltk.download('stopwords')
2
3 stop_words = set(stopwords.words("german"))
4 new_stopwords_list = stop_words.union(additional_stopwords)
5 token_pattern = r'\b[^\d\W]+\b'
6 count_vect = CountVectorizer(stop_words = new_stopwords_list,
7                               token_pattern=token_pattern,
8                               analyzer='word',
9                               max_features=30)
```

[5] *https://scikit-learn.org/stable/*
[6] https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

```
10
11 docs = count_vect.fit_transform(X_train)
```

Listing 4.7: Initialisierung und Anpassung des Zählvektors

Die Instanz des Objekts *CountVektorizer* wird mit den vordefinierten Stopwörtern, den vordefinierten regulären Ausdruck für die Tokens, einem Indikator für die Erstellung von Charakteristiken aus Wörtern und das Limit an Vokabular initialisiert.

Die Instanz erhält die Trainingsdaten zum Erlernen eines Vokabulars und gibt eine Matrix mit 1120 Zeilen und 30 Spalten zurück, welche die Texte aus dem Trainingsdaten (Anzahl X_train) mit dem jeweilig gezählten Vokabular repräsentieren.

4.4.2 Termfrequenz und inverse Dokumentenhäufigkeit

Das Verfahren zur Berechnung des Tf-idf-Maßes fügt sich dem Bereich der Informationsbeschaffung (*Information Retrieval*).

Die Termfrequenz Tf (*term frequency*) gibt an wie häufig ein Term in einem Dokument vorkommt wobei die inverse Dokumentenhäufigkeit idf (*inverse document frequency*) dessen Relevanz im Dokumentkorpus bestimmt. Jeder Term hat seine eigene TF- und IDF-Bewertung, die als Gewichtungen für das Produkt zur Berechnung des Tf-idf-Maßes verwendet werden [Ram03].

Das Tf-idf-Maß eines Terms ist ein Indikator für dessen seltenes Aufkommen eines Dokuments einer Dokumentenkollektion. Ein Term welches in wenigen Dokumenten oft vorkommt hat ein höheres Maß als ein Term, das entweder in fast jeden Dokument oder sehr geringfügig auftaucht.

Die Instanz vom Obekt *TfidfTransformer()*[7], welches von der Bibiliothek *sklearn* zur Verfügung gestellt wird, berechnet das Tf-idf-Maß anhand dem zuvor berechneten Matrix mit dem Vokabular für jeden Text im Dokumentenkorpus[8].

```
1 tfidf_transformer = TfidfTransformer()
2 X_train_tfidf = tfidf_transformer.fit_transform(docs)
```

Listing 4.8: Initalisierung und Anpassung des Tf-idf-Transformierers

Anhand des Vokabulars *docs* wird für jedes Wort im Dokumentenkorpus das Tf-idf-Maß berechnet.

[7] *https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfTransformer.html*
[8] Sammlung schriftlicher texte

4.4.3 Naive Bayes Multinomialverteilung

Naive Bayes Multinominalverteilung ein auf der Bayeschen Regeln basierender Klassifikator, der als schnell und einfach zu implementieren gilt und speziell in der Textklassifikation Verwendung findet. Der Klassifikator modelliert die Verteilung von Wörter in einem Dokument als Multinomial[**RSTK03**].

Der Klassifikator schätzt die bedingte Wahrscheinlich eines bestimmten Terms t bei einer gegebener Kategorie c als relative Häufigkeit des Terms in den Dokumenten der gegebenen Kategorie c[Man08]:

$$P(t|c) = \frac{T_{ct}}{\sum_{t' \epsilon v} T_{ct'}}$$

Die Motivation des Algorithmus ist die Berücksichtigung der Variationen von der Summe der Vorkommen des Terms t in den Dokumenten der Kategorie

c. Um Diesen Algorithmus auf die Trainingsdaten anzupassen, wird das *MultinomialNB()*[9] Objekt verwendet, welches die Tf-idf-Maße des Vokabulars und die dazugehörigen Kategorien entgegennimmt.

```
1 text_clf = MultinomialNB()
2 text_clf.fit(X_train_tfidf, y_train)
```

Listing 4.9: Initialisierung und Anpassen des Naive Bayes Multinomialverteiler

Um die Präzision des Modells zu testen wird eine Validierung des Modells vorgenommen, welche anhand der Testdaten durchgeführt wird. Das Modell wird durch die Testdaten nicht weiter trainiert werden, da für diesen Zweck ausschließlich die Trainingsdaten benutzt werden und bereits vollständig trainiert wurden ist.

4.4.4 Validierung

Der Validierungsprozess ist mit der Evaluation des Modells gleichzusetzen und dient als Bewertungsmaßstab für die Präzision des Klassifikators. Wie bei dem Trainingsdatensatz sind die Testdaten ebenfalls kategorisiert, was bedeutet, dass die Kategorie der Dokumententexte bekannt ist. Resultierend kann der Klassifikator die Testdaten ohne Einbeziehung der Kategorie klassifizieren und anschließend und mit der richtigen Kategorie der Testdatenbeispiele abgeglichen werden, um so die Präzision des Models zu testen.

[9] https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html

	precision	recall	f1-score	support
invoice	0.98	0.93	0.96	138
regress	0.94	0.99	0.96	142
accuracy			0.96	280
macro avg	0.96	0.96	0.96	280
weighted avg	0.96	0.96	0.96	280

Abbildung 4.1: Auswertungsreport

Durch die Funktion *predict()* werden die Testdaten an den Klassifikator übergeben, wodurch anschließend einen Report erstellt werden kann.

```
1 y_pred = text_clf.predict(X_test)
2 classification_report(y_test, y_pred)
```

Listing 4.10: Evaluation des Textklassifkators

Der Report wird mit der Funktion *classification_report()*[10] generiert und liefert eine Auswertung in der Form, die in Abbildung gezeigt wird.

Der Report zeigt verschiedene Ergebnisse, die aus der Evaluation des Models hervorgegangen sind. Die Bedeutung dieser Werte können anhand der Dokumentation[11] von sklearn abgeleitet und zusammengefasst werden.

precision ist das Maß eines Klassifikators wie gut er negative Ergebnisse von positiven Ergebnisse unterscheiden kann. *recall* ist die Fähigkeit eines Klassifikators korrekte Annahmen in den positiven Ergebnissen zu treffen. Der *f1-score* ist ein Bewertungsmaß, welches aus der Präzision und dem Rückruf gewichtet wurden ist. *support* gibt die jeweilige Anzahl der tatsächlichen Vorkommen der Klassen im Datensatz zurück.

Die Verwirrung Matrix bildet die Positiven und negativen Ergebnisse und stellt sie als Heatmap[12] dar. Sie visualisiert die Bewertung der Qualität der Ausgabe des Klassifikators, wie in der Dokumentation[13] von sklearn beschrieben. Bestenfalls verläuft bei der Verwirrung-Matrix eine „helle“ Diagonale von links oben nach rechts unten, da die Werte auf der Diagonalen die korrekten Vorhersagen repräsentieren.

[10] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html
[11] https://www.scikit-yb.org/en/latest/api/classifier/classification_report.html
[12] Diagramm mit Farbcodierung
[13] https://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html

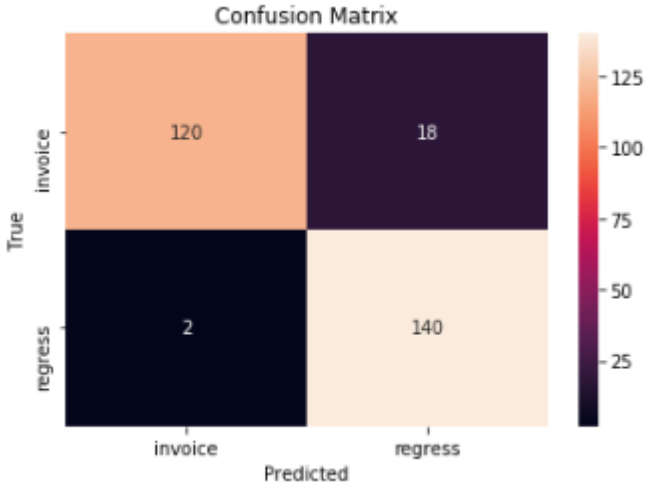

Abbildung 4.2: Verwirrung-Matrix

Interpretation

Insgesamt werden die Rechnungsbelege und Regressbelege sicher (96%) als positiv erkannt. Des Weiteren werden die Belege, die als positiv gewertet sind, richtig (96%) prognostiziert. Die tatsächlichen Vorkommen der Belege ist im geringen Maße unausgeglichen und sollte komplett balanciert sein um strukturelle Schwächen zu vermeiden. Da der Unterschied jedoch sehr gering ist, sollte das keine großen Auswirkungen die Evaluation haben.

In der abgebildeten Verwirrung Matrix 4.4.4 zeigt die Ergebnisse zur Erkennung zweier Klassen Rechnungsbeleg und Regressbeleg auf. Mit einer Gesamtanzahl von 280 Vorhersagen wurden insgesamt 122 Stichproben als Rechnungsbeleg und 158 Stichproben als Regressbeleg identifiziert. In der Realität gibt es 142 Regressbelege und 138 Rechnungsbelege, was bedeutet dass 18 Regressbelege fälschlicherweise als Rechnungsbeleg und 2 Rechnungsbelege als Regressbelege identifiziert wurden sind.

5 Entwicklung der Textextraktion

5.1 Analyse des Dokumentenlayouts

Die Texterkennungssoftware Tesseract kann dafür eingesetzt werden die Koordinaten von verschiedenen Bereichstypen durch eine Übergabe des Bildes zu lesen. Python-tesseract[1] stellt einen Adapter für Tesseract zu Verfügung, der Funktionalitäten rund um die Texterkennungssoftware bereitstellt.

Um bestimmte Bereiche zu Extrahieren wurde ein Dokument erstellt mit einem Format welches für einen Rechnungsbeleg typisch ist.

Im unten dargestellten Programmcode 5.1 wird der Rechnungsbeleg der Texterkennungssoftware übergeben.

Die Rückgabe erzeugt eine Datenstruktur welche die Koordinaten von Regionen enthält, wobei die Regionen modular unterteilt sind. Die Stufen sind so angeordnet, dass die Stufenzahl repräsentatibel für die Regionstypen sind, wobei Stufe 1 das gesamte Dokument als Region ansieht und Stufe 5 einzelne Wörter als Region definiert.

Innerhalb einer Schleife wird der Schlüssel Level iteriert und mit dem Wert BLOCK (2) verglichen. Sollte die Bedingung eintreffen so wird der Block-Bereich anhand der Koordinaten markiert.

Der Rechnungsbeleg, auf welchem die Blöcke gekennzeichnet wurden, wird in der Abbildung 5.1 gezeigt.

Die gekennzeichneten blockartigen Regionen enthalten zum größten Teil Informationen, die in einem Kontext gesehen werden können, wobei noch nicht klar ist welche Box welche Art von Informationen enthält. Dieses Problem ist durch das Zuweisen der Texte anhand der Stufenstruktur lösbar, da die Texte, welche sich auf Stufe 5 befinden, sich den markierten Blöcken zuweisen lassen können, da die Regionen nach dem Stufenmuster gemäß der Abbildung 5.1 angeordnet sind.

[1] https://pypi.org/project/pytesseract/

```
# create dataframe from extracted data
extracted_data = pytesseract.image_to_data(im, output_type=Output.DICT,
    lang='deu')
df = pd.DataFrame.from_dict(extracted_data)
amount_boxes = len(df['level'])

# draw bounding boxes around all boxes
for i in range(n_boxes):
    if df['level'][i] == Level.BLOCK:
                (x, y, w, h) = (df['left'][i], df['top'][i], df['width'][
                    i], df['height'][i])
        cv2.rectangle(cv_img, (x, y), (x + w, y + h), (0, 255, 0), 2)
```

Figure 5.1: Markieren der Regionen von Interesse

Figure 5.2: Bereiche von Interesse

	level	page_num	block_num	par_num	line_num	word_num	left	top	width	height	conf	text
1	2	1	1	0	0	0	260	378	396	14	-1	
2	3	1	1	1	0	0	260	378	396	14	-1	
3	4	1	1	1	1	0	260	378	396	14	-1	
4	5	1	1	1	1	1	260	378	51	12	91	Thomas
5	5	1	1	1	1	2	316	378	39	11	93	GmbH
6	5	1	1	1	1	3	360	378	8	11	91	&
7	5	1	1	1	1	4	372	378	20	11	92	Co.
8	5	1	1	1	1	5	398	378	17	11	94	KG
9	5	1	1	1	1	6	419	384	4	2	93	-
10	5	1	1	1	1	7	427	378	104	12	91	Alexanderstraße
11	5	1	1	1	1	8	535	378	15	11	80	2
12	5	1	1	1	1	9	543	371	7	28	80	-
13	5	1	1	1	1	10	554	378	39	12	93	10178
14	5	1	1	1	1	11	598	378	58	14	91	Hamburg
15	2	1	2	0	0	0	260	464	258	94	-1	

Figure 5.3: Anordnung der Daten

Zunächst wird eine Funktion definiert, die die Indizes für alle Stufe 2 Einträge filtert, wobei zwei Einträge als ein Paar gespeichert werden, da die Indizepaare die Reichweite für einen bestimmten Block darstellen.

```
1  def box_ranges(df):
2          level_two = df['level'] == 2
3          df_two = df[level_two]
4          box_ranges = []
5
6          for i, (a, b) in enumerate(zip(df_two.iterrows(), df_two.iloc
               [1:].iterrows())):                          box_ranges.append((
               a[0], b[0]))
7                          if i == len(df_two) - 1:
8                                  box_ranges.append((b[0], df.index[-1]))
9
10         return  box_ranges
```

Anhand der

```
1  def get_boxes(df, ranges):
2      boxes = []
3      joined = ' '
```

```
4         for current, (start, end) in zip(df_two.iterrows(), ranges):
5         coords = current[1]['left'], current[1]['top'], current[1]['width
              '], current[1]['height']                    curr_subdf = df
              .iloc[start:end+1]
6                 joined = ' '
7                 text_list = []
8                 for current_sub in curr_subdf.iterrows():
9              if current_sub[1]['level'] == 5:
10                                text_list.append(current_sub[1]['text'])
11                                        boxes.append(Box(*coords, text=
                                              joined.join(text_list)))
12
13                return boxes
```

```
1 for box in boxes:
2         if balance_due in box.text:
3                 box.draw_bounding_box(cv_img, COLORS[0], box.x, box.y,
                      box.w, box.h)
4         box.draw_label(cv_img, COLORS[0], LABELS[0], box.x, box.y, box.w,
              box.h)
5          regions_of_interest.append(RegionOfInterest(box.y, box.y + box.h
              , box.x, box.x + box.w, LABELS[0]))
```

```
1 for rol in regions_of_interest:
2         cropped_img = cv_img[rol.y:rol.y2, rol.x:rol.x2]
3         cv2.imwrite('./data/img/RegionsOfInterest/{name}.jpg'.format(name
              =rol.label), crop_img)
```

5.2 Textextraktion auf Basis des Dokumentenlayouts

Figure 5.4: Ausgezeichnete Regionen

6 Entwicklung des Servers

Dieses Kapitel befasst sich mit der Entwicklung des Servers und ist in den zwei Kapiteln Architektur und Implementierung gegliedert.

6.1 Architekturentwurf

In Diesen Unterkapitel wird die Architektur des Servers beschrieben und visualisiert.

6.1.1 REST-Schnittstelle

Die Methoden *PUT* und *DELETE* werden aus Sicherheitsgründen kaum implementiert und werden in diesen Kontext nicht weiter diskutiert.

Um eine REST-konforme Schnittstelle zu programmieren, werden die Grundsätze in der Regel eingehalten. Bedingt können Webservices REST-konformalitäten abweichen. Ein Webservice, über dessen Endpunkt ein Token angefordert werden kann, wird als Anfragemethode ausschließlich *POST* verwenden, obwohl kein schreibender Zugriff erforderlich ist. Anders wie bei der *GET-methode* kann das Benutzerpasswort verschlüsselt an den Server übermittelt werden.

In Abbildung 6.1.1 ist dargestellt, wie die Kommunikation zwischen Client und Server verläuft und wie intern der Server die Anfragen behandelt. Zudem ist zu erkennen, dass die REST-Schnittstelle und die Geschäftslogik auf Modullevel voneinander getrennt sind.

Eine Anfrage wird im Server von der REST-Schnittstelle entgegengenommen und an die zuständige Geschäftslogik weitergeleitet. Die Geschäftslogik wickelt den HTTP-Anfragen entsprechend die Datenbankabfragen ab und übermittelt über die REST-Schnittstelle eine Response mit den erwarteten Ressourcen an den Client.

Die Architektur verwendet das bewährte MVC-Muster (Modell-View-Controller) zur Unterteilung des Systems. Im Kontext stellt das Model die Geschäftslogik und der die Schnittstelle der Controller dar. Beide Komponenten befinden sich in der übergeordneten Server-Komponente. Der in der Client-Komponente befindliche Webbrowser fungiert als Benutzeroberfläche und ist die View

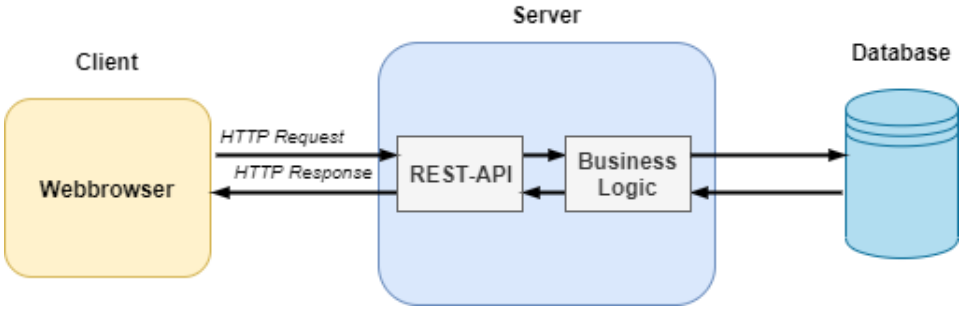

Abbildung 6.1: REST-Architektur

des Systems. Gemäß des MVC-Musters kommunizieren View (Webbrowser) und Model (Business Logic) über den Controller (REST-API).

Durch die Zustandslosigkeit können Webservices einfach skaliert werden, da eine Vielzahl von Requests problemlos auf verschiedene Serverinstanzen verteilt werden können. Durch die Kapselung von Client-und Serverseitigen Logik sind die Systeme als einzelne funktionsfähig. Durch geschicktes mocken der Anfragen und Antworten können Server und Client vollständig unabhängig voneinander entwickelt werden, sodass keine Rücksicht bei der Entwicklung auf die Gegenseite genommen werden muss.

Die Daten in sowohl den Anfragen und auch Antworten werden in einem spezifizierten Datenformat übermittelt. Hier bietet es sich an die Daten in einem JSON-format (*JavaScript Object Notation*) zu transferieren. Das JSON-Format wurde von Douglas Crockford im Jahr 1999 auf der ECMA-262 spezifiert[**JSON˙DG**]. im Gegensatz zu XML (*Extensible Markup Language*) dynamisch ist und keine Spezifizierung notwendig ist. Zudem ist das ein umgängliches und sprachunabhängiges Datenformat, da es nach einer bekannten Datenstruktur aufgebaut ist - Der Schlüssel/Wert-Paare. Diese universelle Datenstruktur wird in jeder modernen Programmiersprache genutzt und ist deswegen einfach zu verarbeiten.

Für die Implementierung der REST-Schnittstelle wird die Programmiersprache Python[1] an. Python wird unternehmensintern verwendet, sodass im weiteren Verlauf die Schnittstelle von anderen Entwicklern mit weitergeführt werden können. Das Webframework Flask[2] liefert hilfreiche Module zur Entwicklung einer REST-Schnittstelle und wird verwendet um die zukünftige Architektur des Unternehmens technisch umzusetzen.

[1]https://www.python.org/
[2]http://www.flask.palletsprojects.com

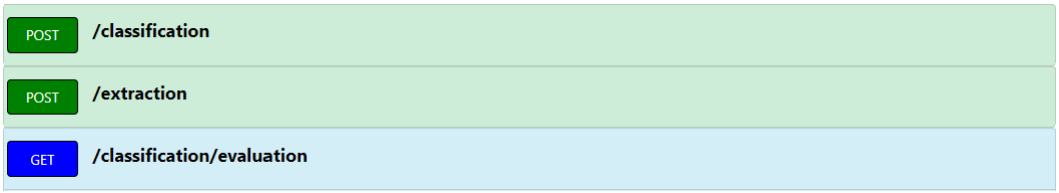

Abbildung 6.2: REST-Services

Schnittstellenspezifikation

Die Schnittstellenspezifikation ist an das Beschreibungsformat *OpenAPI*[3] angelehnt. Sie befindet sich clientseitig auf dem Dashboard und wird später mit *Swagger*[4], einer Beschreibungssprache für REST-Schnittstellen, in die unternehmensinternen Schnittstellenbeschreibung überführt. In Abbildung 6.1.1 befinden sich die ansprechbaren Endpunkte für die zur Verfügung gestellten Ressourcen.

Für eine Einheit der Ressource *evaluation* wird die entsprechende Adresse nach dem folgenden Muster angesprochen: *.../classification/evaluation/*{*id*}. *id* wird mit der Identifikationsnummer der Evaluation ersetzt, sodass ein Zugriff auf genau die bestimmte Ressource erfolgt.

Benutzer, welche die Routen ansprechen wollen, haben durch die Spezifikation eine klare Übersicht. Es wird veranschaulicht welche URI's Ressourcen zur Verfügung stellen und welche Anfragemethode benutzt werden muss.

Jeder Webservice hat eine Definition für die Parameter, die bei der Anfrage transferiert werden müssen, die möglichen Status-Codes einer Antwort und ein Beispiel für eine gelieferten Ressource bei einem reibungslosen Aufruf des Services.

Ein Nutzer mit der Intention *extraction* aufzurufen, muss die nötigen Parameter *file* und *file_type* als Nutzdaten (*Payload*) im JSON-Datenformat an die korrespondierende URI schicken. Beide Parameter sind erforderlich, da der Server die Informationen benötigt um die Anfrage zu bearbeiten und eine Antwort zu schicken. Der Beschreibung von *file* zufolge muss die Datei in der Hexadezimalnotation vorliegen. Der Typ der Datei kann entweder im PDF-Format oder in Bild-Format vorliegen. Der Client muss also technisch in der Lage sein eine Datei in die entsprechende Hexadezimalnotation zu kodieren und den Dateitypen zu kennen, damit eine für den Server passende Anfrage konstruiert werden kann. Der Server antwortet mit dem Statuscode *400 - BAD REQUEST* wenn die nötigen Informationen nicht vorhanden sind oder die falsche Anfragemethode gewählt wurden ist.

[3]https://www.openapis.org
[4]https://www.swagger.io

6.1.2 Authentifizierung

Die Authentifierungprozedur wird über einen IAM-Server (*Identity and Access Management*) abgewickelt. Das Akronym IAM bezeichnet das Identity and Access Management, welches die Aufgabe der Zugriffssteuerung auf Ressourcen übernimmt.

Um auf die Services zuzugreifen müssen sich Nutzer gegen den jeweiligen Service authentifizieren. Über Services, die der IAM-Server zur Verfügung stellt, können Nutzer erstellt, verwaltet und wieder gelöscht werden.

Es wird durch den unternehmensinternen IAM-Server ein Anmeldungsservice angesprochen, bei dem für Benutzername und Passwort ein Token angefordert werden kann. Das Dashboard wird als ein geschützter Bereich (*Realm*) betrachtet. Der Realm hat zugewiesene Benutzer, sessions dies das

Um die zuvor genannten Services anzusprechen muss der Anfrage ein valider Token in den Header hinzugefügt werden. Der Token hat nur eine bestimmte Gültigkeitsspanne und muss nach Ablauf der Gültigkeit erneut beim IAM-Anmeldungsservice beantragt werden.

6.1.3 Persistierung

Für die Persistierung der Daten wird eine PostgreSQL[5]-Datenbank verwendet. PostgresSQL ist performant, hat eine fortgeschrittene Datenbanksprache und ist zudem sehr kompatibel mit Python.

Aus dem Anwendungsszenario lassen sich folgende Datenbankmodelle ableiten: Klassifikation, Evaluation und Bild. Aus den Entitäten ergibt sich das Entity-Relation-Ship, das in Abbildung 6.3 dargestellt ist.

Eine Klassifikation wird durch ein Ergebniss evaluiert. Ein Bild stellt eine Erweiterung eines Ergebnisses dar und ist damit einem Ergebniss zugeordnet.

Objektrelationale Abbildungen

Das Akronym ORM bezeichnet das Object-Relational Mapping, welches eine Technik beschreibt Objekte in eine relationale Datenbank abzulegen.

Basierend auf dem ER-Modell werden in der Implementierung Objekt-relationale Abbildungen erzeugt. Mithilfe vom ORM-Framework SQLAlchemy[6] werden benutzerdefinierte Python-Klassen

[5]https://www.postgresql.org
[6]https://www.sqlalchemy.org/

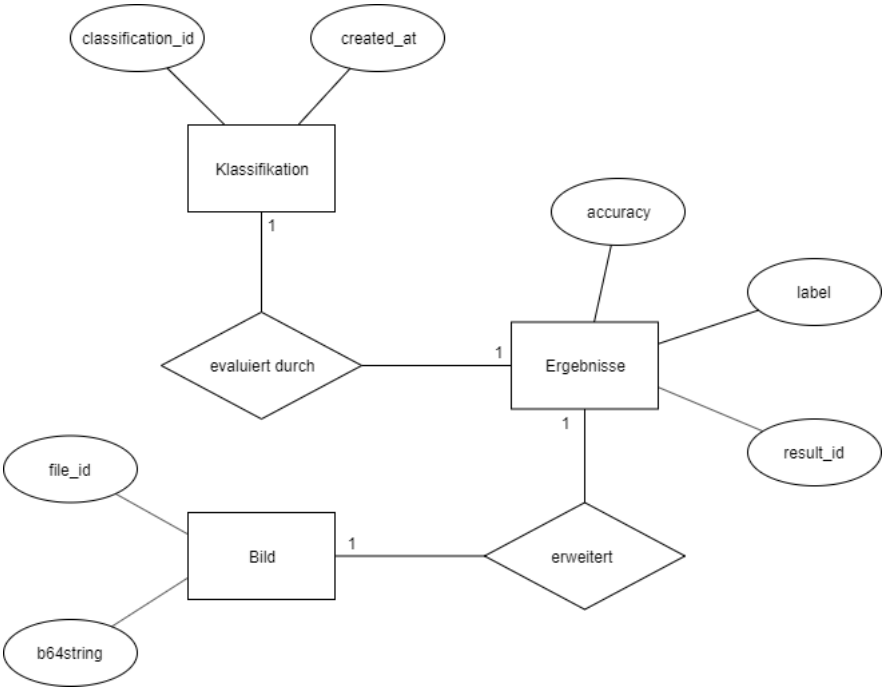

Abbildung 6.3: Entity-Relationship-Modell

erzeugt, die in einer relationalen Datenbank als Zeilen in den korrespondierenden Tabellen abgebildet werden. Die Kardinalitäten werden als Attribute in den Klassenobjekten. ORM synchronisiert die Zustände zwischen den Objekten und den zugehörigen Tabellen. Es werden Datenbankabfragen in Bezug auf die erzeugten Python-Klassen und deren Beziehung untereinander programmatisch ausgedrückt. Die zuständige Programmierung befindet sich in der Geschäftslogik.

Die oben dargestellten Entitäten und deren Beziehungen untereinander werden in abstrahierter Form mit Python-Objekten deren Attribute definiert. Abhängig von Definition der Objekte und deren Beziehungen zueinander, können Datenbankabfragen mit der SQLAlchemy-Ausdruckssprache geschrieben werden. Sollte ein Zugriff auf die Datenbank erfolgen verändert der Ausdruck den Zustand der Datenbank korrespondierend zu dem benutzerdefinierten Statement.

Das Klassenmodell, welches in Abbildung 6.4 veranschaulicht ist, wird aus dem Entity-Relationship-Modell abgeleitet und ist im Stil vom Objektorientierten Programmierparadigma in Python zu überführen.

Zusätzlich sind die Primärschlüssel der Eltern-Entitäten zu den jeweiligen Kind-Entitäten als Fremdschlüssel hinzugefügt. Das Objekt DBClassification enthält zudem eine Methode, welche die Logik enthält eine Klassifizierung zu erzeugen und diese zusammen mit den abhängigen Entitäten in die Datenbank abzulegen.

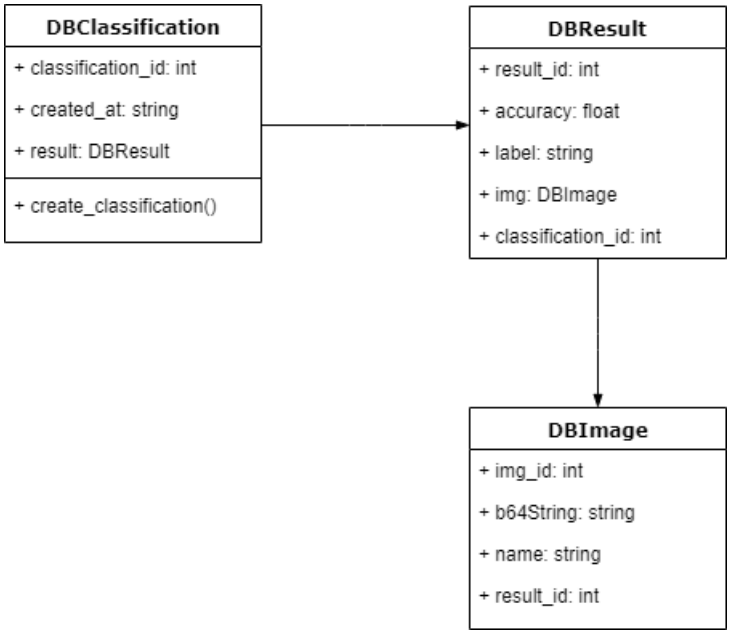

Abbildung 6.4: Klassendiagramm

6.1.4 Validierung

Im Kontext der REST-Schnittstelle werden Anfragen auf benötigte Parameter und deren Zeichenlänge validiert. Durch eine schematische Validierung werden ungültige Anfragen erkannt und abgefangen, was bedeutet dass die Geschäftslogik nicht angesprochen wird. Eine entsprechende Antwort mit dem HTTP-Status-Code 404 - BAD REQUEST wird an den Dienstnutzer, womit ihm kenntlich gemacht wird, dass seine Anfrage nicht validiert werden konnte.

6.2 Implementierung

6.2.1 Controller

In diesem Unterabschnitt wird die Implementierung der Schnittstelle exemplarisch für einen Anwendungsfall erläutert. Die REST-Schnittstelle, welche im Model-Controller Muster den Controller darstellt, wurde mit dem Webframework Flask realisiert. Flask bietet eine Technik an die Webservices einer Webapplikation in Sektionen aufzuteilen. Diese Sektionen werden im Kontext von Flask als *Blueprints* definiert. Diese Blueprints müssen bei der Webapplikation registriert werden.

Insgesamt gibt es, wie in 6.1 aufgezeigt, drei Sektionen die registriert werden.

Exemplarisch wird der Blueprint für den Klassifikations-Service anhand des Programmcode 6.2 beschrieben.

Algorithmus 6.1 Registrieren der Blueprints

```
app.register_blueprint(home.api)
app.register_blueprint(classification.api)
app.register_blueprint(extraction.api)
```

Algorithmus 6.2 Schnittstelle für die Ressource Klassifikation

```
api = Blueprint('/classification', __name__)
@api.route('/classification', methods=['GET', 'POST'])
def classification():
  if request.method == 'POST':
        data = request.form.to_dict()
        validator = Validator(schema_type='classification')
        validator.validate(data)
        if validator.validated:
            resp = Classification().predict_class(req=data)
            if not resp:
                return jsonify({'status': 'bad request'}), 400
        else:
            return jsonify({'status': 'validation failed'}), 422

        return jsonify(resp), 202
```

Der Blueprint wird für die Adresse */classification* und erlaubt die HTTP-Anfragemethoden *GET* und *POST*. Beim Aufruf dieser Funktion wird zunächst überprüft um welche Anfragemethode es sich handelt. *GET* liefert dem Dienstnutzenden alle Ergebnisse über die erstellten Klassifikationen. Sollten keine Ergebnisse existieren wird eine Antwort mit dem Status und dem Status-Code 404 - NOT FOUND zurückgesendet.

6.2.2 Model

Das Model enthält sowohl die Geschäftslogik als auch das Datenmodell.

Datenbankmodell

Im Datenbankmodell werden die Objekte definiert, die in die Datenbank abgelegt werden. Darüber hinaus werden hier die Funktionalitäten für die Datenbankabfragen hinterlegt.

Der Programmcode 6.3 zeigt eine Datenbankabfrage die die Vorhersagewahrscheinlichkeit jeder Klasse ermittelt.

```
class DBClassification(Base):
    __tablename__ = 'classification'

    classification_id = Column(Integer, primary_key=True, autoincrement=
        True)
    created_at = Column(Date)
    result = relationship("DBResult", uselist=False, back_populates="
        classification")
    def __init__(self, created_at, result):
        self.engine = Database().engine
        if not self.engine.dialect.has_table(self.engine, self.
            __tablename__):
             Base.metadata.create_all(self.engine)

        self.created_at = created_at
        self.result = result
```

Abbildung 6.5: Datenbankmodell Klassikation

Algorithmus 6.3 Bestimmen der Vorhersagewahrscheinlichkeiten für jede Klasse

```
def calc_accuracies():

        accuracies = {}
        sess = Database().session()
        for label in sess.query(DBResult.label):
                label = list(label)[0]
                acc_sum = sess.query(func.sum(DBResult.accuracy)
                                                              ).filter(
                    DBResult.label == "{label}"
                                                .format(label=label)).scalar()
        accuracies.update({label: round(acc_sum / Evaluation.
            count_predictions(label), 2)})
    sess.close()

        return accuracies
```

Geschäftslogik

In der Geschäftslogik werden die Anfragen entgegengenommen und nötige Funktionalitäten zur Bearbeitung der Anfragen angewandt. Außerdem werden die Datenbankabfragen über die Datenmodelle vorgenommen

Algorithmus 6.4 Bestimmen der Vorhersagewahrscheinlichkeiten für jedes label

```
1  class Classification:
2      def __init__(self, labels=labels):
3              self.model = load_model('../../invoice_classifier.h5')
4              self._labels = labels
5
6          @property
7          def labels(self):
8              return self._labels
9
10         @labels.setter
11         def labels(self, values):
12         if type(values) is not list:
13             raise TypeError('labels needs to be passed as list!')
14         self._labels = values
15
16         def classify(self, img):
17             y_prob = self.model.predict(img)
18         y_classes = y_prob.argmax(axis=-1)
19         max_y_prob = y_prob[0][y_prob.argmax(axis=-1)[0]]

20             max_percentage = round(float(max_y_prob) * 100, 2)
21             K.clear_session()
22
23             return max_percentage, y_classes
24
25         def predict_class(self, req):
26         decoded_img = decode_b64_to_img(req.get('file'), file_type=req.
               get('type'))
27         if decoded_img is None:
28             return
29             scaled_img = cv2.resize(decoded_img, (32, 32))
30         rgb_img = cv2.cvtColor(scaled_img, cv2.COLOR_RGB2BGR)
31         image_4d = rgb_img[np.newaxis, ...]
32
33         max_y_prob, pred = self.classify(image_4d)
               self.db_entry(result=DBResult(label=self.labels[pred[0]],


                                                 accuracy=max_y_prob,


                                             img=DBImages(
34
44
```

7 Entwicklung des Clients

Dieses Kapitel befasst sich mit der Entwicklung des Clients und ist in den zwei Kapiteln Architektur und Implementierung gegliedert.

7.1 Architekturentwurf

In Diesen Unterkapitel wird die Architektur des Clients beschrieben und visualisiert.

7.1.1 Technologien

Das Dashboard wird als Webapplikation auf Basis von HTML5, CSS und JavaScript entwickelt. Das Dashboard repräsentiert Daten die vom Server über HTTP-Requests geholt werden.

Node.js

Der Webclient läuft als HTTP-Server über Node.js[1] auf it eine V8 JavaScript Laufzeitmaschine, dem derzeit schnellsten JavaScript-Compiler. Die Verwaltung der gesamten abhängigen Frameworks und Bibliotheken verläuft über dem JavaScript-Paketmanager NPM (*Node Package Manager*), der speziell für Node.js-Applikationen entwickelt wurden ist. Die Abhängigkeiten werden zusammen mit Projektinformationen, Konfigurationen und Skriptbefehle in einer JSON-Datei mit der Dateibezeichnung *package.json* festgehalten. Der Kommandozeilen-Befehle *npm install* installiert alle definierten Abhängigkeiten des Projekts über die Node.js Schnittstelle. Die installierten Abhängigkeiten werden im festgelegten Ordner *node_modules* akkumuliert.

React

Zusätlich wird die auf Babel basierende Bibliothek React[2] genutzt. Durch React lässt sich das UI (*User Interface*) in wiederverwendbare und unabhängige Komponenten isolieren. Komponenten

[1] https://nodejs.org
[2] https://reactjs.org/

Algorithm 7.1 Anfrage bei Aufruf des Dashboards

```
1 componentDidMount() {
2     const API = 'http://localhost:8090/classification';
3     this.getResult(API)
4 }
```

stellen mit ihren Funktionen und Attributen eine Abstaktion von JavaScript-Objekten dar. Jede Komponenteninstanz hat ihren Einfluss auf sowohl ihren Document Object Model Knoten als auch auf die Instanzen der Kindskomponenten.

Axios

promise based requests

7.2 Implementierung

Dieser Abschnitt erläutert die Implementierung des Dashboards.

7.2.1 Lazy loading

React lazy loading -> Performance

7.2.2 Anfragen

In Diesem Kontext übernehme ich in der SOA die Rolle des Dienstnutzers.

Der Zeitpunkt der Beanspruchnahme des Dienstes soll dem Benutzer gegenüber kontrolliert sein. Bei einigen Situationen, wie beispielsweise der Aufruf des Dashboards, kann die Anfrage direkt beim Aufruf erfolgen. React bietet eine Funktion die es einer Komponente erlaubt Funktionalitäten beim Eintritt in den DOM auszuführen.

Im Programmcode 7.1 wird eine Methode aufgerufen, die eine Anfrage mit der Anfragemethode *GET* an den Endpunkt */classification* sendet.

Die Methode *getResult(),* die im Programmcode 7.2 aufgezeigt wird, fragt über den Webservice */classification* nach Daten. Im sonst asynchronen Programmablauf sind promises die einzige Möglichkeit einen synchronen ablauf von Code durchzuführen. Während der Anfrage werden Zustandsattribute der Komponente mit der Methode *setState()* gesetzt.

Im Konstruktor der JavaScript Funktion Axios, wird die Anfragemethode, die anzusprechende URL und der Header über ein JSON definiert. Wenn die Anfrage erfolgt ist und der Status Code

Algorithm 7.2 GET Request an Klassifikation

```
getResult = (API) => {
    this.setState({
        loading: true,
    });

        axios({
                method: 'get',
                url: API,
                config: {
                        headers: {
                                'Access-Control-Allow-Origin': '*',
                    'Content-Type': 'multipart/form-data'
                        }
                        }
                })
        .then((response) => {
                        if (response.data.result) {
                                this.setState({
                                        accuracies: response.data.result.
                                            accuracies,
                            amount: response.data.result.amount,
                                        predictions: response.data.result
                                            .predictions,
                                loading: false
                                    });
                        }
                        console.log(response);
        })
                .catch(function (response) {
                console.log(response);
                console.log('error');
        });
};
```

im Rahmen von 200-399 ist, wird im promise *.then()* mit der Antwort die Zustandsattribute der Komponente gesetzt. Das *.catch()* promise loggt die Antwort auf die Browserkonsole, wenn die Anfrage nicht erfolgreich war.

7.2.3 Rendering

Damit die Daten in eine repräsentativ veranschaulicht im Webbrowser angezeigt werden, müssen die Zustandsattribute an Graph-Komponenten übergeben werden. Graph Komponenten werden über die Bibliothek react-chartjs-2 zur Verfügung gestellt und erwarten eine JSON für die Instanziierung der Komponenten. Reactstrap ist eine Bibliothek die responsive Komponenten im Stil von Bootstrap zur Verfügung stellt.

8 Evaluation

In diesen Abschnitt ist wird die Evaluierung der Prozessautomatisierung erläutert. Hierzu wurden Testdurchläufe durchgeführt um Erkenntnisse bezüglich der durchschnittlichen Zeitersparnis zu generieren.

8.1 Testdaten

Zur Errechnung der durchschnittlichen Zeitersparnis wurden mit verschiedenen Testpersonen Szenarien durchlaufen, die sich an die Schadensprotokollierung mit und ohne Textextraktion orientieren, um Richtwerte zu generieren.

8.1.1 Testplanung

Die Testplanung definiert Vorbedingungen und Aspekte bezüglich der Durchführung des Testes. Die Tests werden unternehmensintern auf der graphischen Benutzeroberfläche des Produktes durchgeführt. Es wird der Zeit gemessen, die die Testpersonen benötigt, um ein Formular für die Schadensprotokollierung mit den Daten eines gegebenen Rechnungsbeleg abzuschicken, wobei eine mögliche Vorbedingung getroffen wurden ist:

Leeres Formular

Um den momentane Durchlaufzeit der gegenwärtigen Prozesslaufzeit zu evaluieren, wurden die Formularfelder leer gelassen.

gefülltes Formular

Diese Vorbereitung soll ebenfalls für die Textextraktion sein, wobei eine variable Anzahl von Formularfelder falsch ausgefüllt wurden sind, um Fehler vorzutäuschen.

name	street	city	bank	account number	iban	bic	invoice number	fee	time
1	1	1	1	1	1	1	1	1	15.663
0	1	1	0	1	0	1	1	1	19.663
1	1	1	1	1	1	1	1	1	12.711
0	0	0	1	0	1	0	1	1	35.927
1	1	1	1	1	1	1	1	1	18.034
0	1	1	0	0	1	1	0	0	43.204
1	1	1	1	1	1	1	1	0	65.877
1	0	1	0	0	1	1	0	1	33.410
0	0	1	1	1	1	1	1	1	23.450
1	1	1	1	1	1	1	1	1	16.030

Abbildung 8.1: Testergebnisse mit Extraktion

Die Testperson ist bei einem gefüllten Formular in Unkenntnis gelassen, dass einige Felder fehlerhafte Werte enthalten. Sie hat aus diesem Grund die Aufgabe die Werte gründlich zu überprüfen und gegebenenfalls zu korrigieren, bevor das Formular abgeschickt wird.

8.1.2 Testdurchführung

Eine von der jeweiligen Vorbedingung abhängig präparierte Benutzeroberfläche wird der Testperson zur Verfügung. Die Daten werden quantitativ erhoben, da eine Messung anhand numerischer Zahlen für diesen Zweck sinnvoll erscheint. Die Tests sind unter einer Zeitmessung durchgeführt und wurden kategorisch in einer CSV-Tabelle persistiert, womit automatische und manuelle Ausfüllprozesse getrennt betrachtet werden können. Die beständigen Fehler wurden gezählt, somit kann die Fehleranzahl mit der gebrauchten Zeit korreliert werden.

8.1.3 Testergebnisse

In einem Jupyter Notebook ist die CSV-Tabelle ausgelesen und in einem Dataframe Datenstruktur überführt wurden. Insgesamt wurden 40 Testergebnisse aufgezeichnet.

In der Abbildung 8.1 befindet sich ein Ausschnitt von dem Resultat. Jedes Datenelement verfügt über die entstandenen Fehler, welche über die Felder verteilt sind, und die Dauer für die Schadensprotokollierung *time,* welche in Sekunden wiedergegeben wird. Für die Felder existieren Indikatoren, ob die jeweiligen Informationen richtig extrahiert wurden, dabei wird 1 als richtig (*true*) und 0 als falsch (*false*) gewertet.

Bezeichnung	Anzahl	Anzahl der Fehler	Absoluter Prozentualer Anteil	Relativer Prozentualer Ant
name	40	11	3,06%	13,42%
street	40	12	3,33%	14,64%
city	40	11	3.06%	13,42%
bank	40	16	4,44%	19,52%
account number	40	14	3,89%	17,08%
iban	40	14	3,89%	17,08%
bic	40	15	4,17%	18,3%
invoice number	40	15	4,17%	18,3%
fee	40	14	3,89%	17,08%
Durchschnitt	40	13.56	3,77%	16,54%
Gesamt	360	122	33.89%	100%

Tabelle 8.1: Durchschnittliche Fehlerraten

Wird die Fehler und die Zeit betrachtet, so werden Korrelationen zwischen den verschiedenen Feld-variablen und der Zeit erkannt, da eine erhöhte Fehleranzahl zu einer erhöhten Zeitspanne führt.

8.2 Auswertung

In diesen Unterabschnitt werden die Testdaten ausgewertet und untersucht.

8.2.1 Fehlerraten

Es wurden insgesamt 122 Fehler bei den 40 Tests registriert, wobei 9 Felder vorhanden sind. Auf einen Test gerechnet ergibt das 3.05 Fehler pro Testdurchlauf, was einer durchschnittlichen Fehlerquote von 3.77% entspricht.

Mit jeweils 4,44% Fehlern bei *bank* wurden die höchsten Fehlerquoten festgestellt, wobei *name* und *city* mit 3,06% die geringste Fehlerquote aufweisen. Auf die Gesamtfehlerquote relativiert besteht ein prozentualer Unterschied von 6,1%.

Werden die Fehlerraten, wie in 8.2 aufgezeigt, mit dem durchschnittlichen Fehlerquotienten verglichen, so wird deutlich, dass *name*, *street* und *city* mit relativ hohen Abstand am wenigsten Fehler aufweisen. Die Werte *bank*, *bic* und *invoice number* weisen die häufigsten Fehler auf, wobei die restlichen Werte geringfügig über den Durchschnitt liegen.

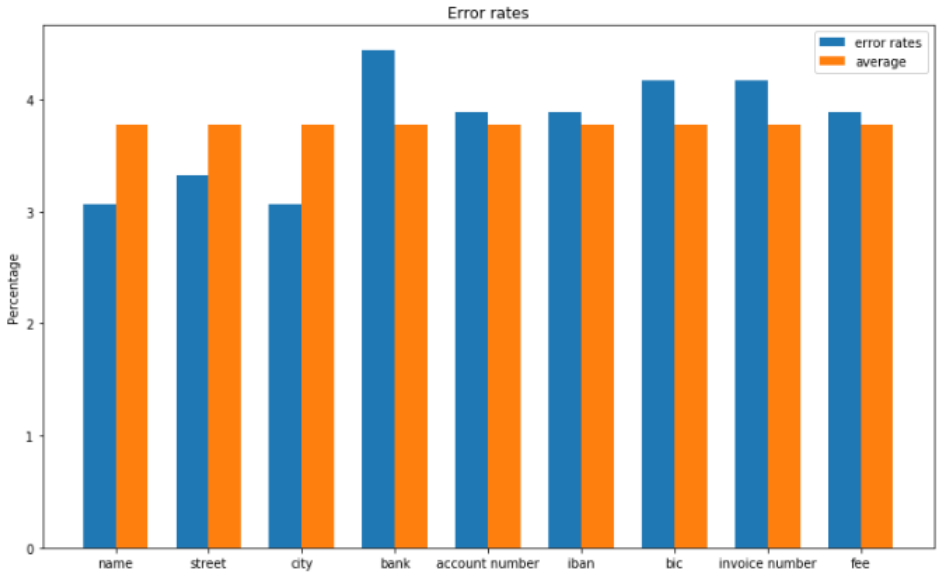

Abbildung 8.2: Absolute prozentuale Fehlerraten

Korrelationen

In der Abbildung 8.3 wird die Korrelationsmatrix als Heatmap abgebildet, die im wesentlichen die Korrelationskoeffizienten und ihre Korrelationen untereinander erfasst. Die Gewichtungen werden anhand der Farbpalette repräsentativ dargestellt, wobei negative Korrelationen einen dunkleren Farbton haben und positive Korrelationen einen helleren Farbton.

Beim näheren Betrachten der Korrelationsmatrix wird festgestellt, dass sich Gruppierungen in den positiven Korrelationen bilden. Die Felder *name*, *city* und *bank* haben starke positive Korrelationen untereinander, wodurch die Interpretation herausgezogen wird, dass die Felder gruppierend Auftreten und häufig entweder richtig oder falsch sind. Der gleiche Fall tritt bei den Werten der Bankdaten *bank*, *account number*, *iban* und *bic* auf.

Das Gruppen-ähnliche Verhalten der Koeffizienten macht Sinn, da die Textextraktion auf der Basis von Blockkoordinaten basiert und eine spezifische Koordinatenkomposition falsche oder richtige Daten verursacht.

8.2.2 Zeitersparnis

Die Zeitersparnis fordert Zeitmessungen anhand Testdurchläufe ohne Extraktion der Information. Damit die durchschnittliche Zeitersparnis errechnet werden kann, wurden Testdurchläufe durchgeführt, um die gebrauchte Zeit zu messen

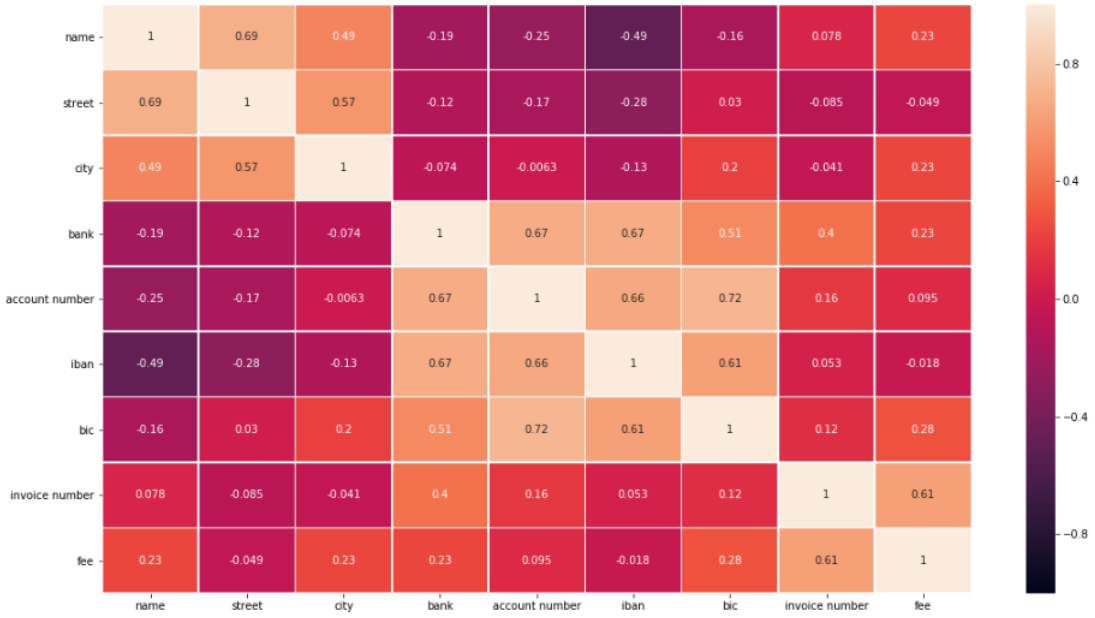

Abbildung 8.3: Korrelationsmatrix als Heatmap

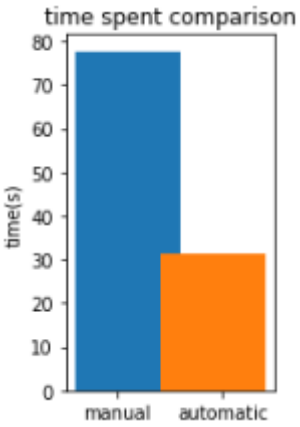

Abbildung 8.4: Vergleich Zeitverbrauch

Der durchschnittliche Zeitverbrauch ist, wie in In Abbildung 8.2.2 veranschaulicht, bei manuellen Eingaben ist 77,71 Sekunden, wobei der durchschnittliche Zeitverbrauch bei automatisierten Eingaben 31,16 Sekunden beträgt. Die Differenz der Zeitangabe beträgt 46,55 Sekunden, was einer prozentualen Zeitersparnis von 59.9% entspricht

9 Zusammenfassung

Dieses Kapitel befasst sich mit der Entwicklung des Clients und ist in den zwei Kapiteln Architektur und Implementierung gegliedert.

9.1 Zusammenfassung

Abbildungsverzeichnis

Listings

List of Tables

Literatur

[Ber09] Gregor Rayman Bernhard Lahres. *Objektorientierte Programmierung*. http://openbook.rheinwerk-verlag.de/oop/oop_kapitel_08_002.htm. 2009.

[Bis06] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. http://cds.cern.ch/record/998831. Feb. 2006.

[Bre03] Thomas M. Breuel. *High Performance Document Layout Analysis*. 2003.

[Cen01] Natural Language Processing Centre. *Syntactic Analysis*. https://nlp.fi.muni.cz/en/MainTopics. 2001.

[EE07] Institute of Electrical und Electronics Engineers. *Ninth International Conference on Document Analysis and Recognition*. 2007.

[Fie00] Roy Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf. 2000.

[Ger17] Aurelien Geron. *Hands On Machine Learning with Scikit Learn and TensorFlow*. 2017.

[Has14] Francoise Beaufays Hasim Sak Andrew Senior. *Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling*. https://storage.googleapis.com/pub-tools-public-publication-data/pdf/43905.pdf. 2014.

[Lan98] Foltz Landauer. *Introduction to Latent Semantic Analysis*. http://lsa.colorado.edu/whatis.html. 1998.

[Man08] Christopher D. Manning. *Tackling the Poor Assumptions of Naive Bayes Text Classifiers*. https://nlp.stanford.edu/IR-book/html/htmledition/naive-bayes-text-classification-1.html. 2008.

[Mel10] Ingo Melzer. *Service-orientierte Architekturen mit Web Services*. 2010.

[Pat09] Klaus Zeppenfeld Patrick Finger. *SOA und Webservices*. 2009.

[Ram03] Juan Ramos. *Using TF-IDF to Determine Word Relevance in Document Queries*. 2003.

[Ran02] Venu Govindaraju Rangachar Kasturi Lawrence O'Gorman. *Document image analysis: A primer*. https://www.ias.ac.in/public/Volumes/sadh/027/01/0003-0022.pdf. 2002.

[Ron11] Jason Weston Ronan Collobert. *Natural Language Processing (Almost) from Scratch*. http://www.jmlr.org/papers/volume12/collobert11a/collobert11a.pdf. 2011.

[SN95] Frank R. Jenkins Stephen V. Rice und Thomas A. Nartker. *The Fourth Annual Test of OCR Accuracy*. http://stephenvrice.com/images/AT-1995.pdf. 1995.

[Spy03] Mike Spykerman. *Typical spam characteristics*. https://www.spamhelp.org/articles/Spam-filter-article.pdf. 2003.

[Ste09] Edward Loper Steven Bird Ewan Klein. *The Fourth Annual Test of OCR Accuracy*. 2009.

[Sul10] Ghazali Sulong. *A Survey on Methods and Strategies on Touched Characters Segmentation*. https://www.researchgate.net/figure/Flow-diagram-of-traditional-OCR-system_fig1_266584079. 2010.

[V S10] R. Anitha V. Srividhya. *Evaluating Preprocessing Techniques in Text Categorization*. 2010.