Universität Stuttgart Institut für Maschinelle Sprachverarbeitung
Pfaffenwaldring 5b
70569 Stuttgart

Fraunhofer Institut für Arbeitswirtschaft und Organisation
Nobelstraße 12
70569 Stuttgart

Bachelorarbeit

Generierung von synthetischen Trainingsdaten für die Erkennung von Absenderdaten aus Brief-Korrespondenz

Jannik Burkhardt

Studiengang:	Medieninformatik
Prüfer/in:	Prof. Dr. Jonas Kuhn
Betreuer/in:	Matthias Blohm, M.Sc. Dr.-Ing. Maximilien Kintz
Beginn am:	1. Dezember 2020
Beendet am:	1. Juni 2020

Kurzfassung

Ein Problem, das sich oft bei Machine-Learning Projekten auftut, ist der Mangel an passenden Trainingsdaten. In dieser Arbeit wird untersucht, wie hoch der Nutzen aus der Verwendung synthetischer Daten in Situationen ist, wo nur sehr wenige echte Trainingsdaten zur Verfügung stehen. Am Beispiel der Absenderdatenerkennung in Briefkorrespondenz wird beschrieben, auf welche Eigenschaften synthetischer Dokumente zu achten ist, damit eine künstliche Intelligenz mit ihrer Hilfe auch echte Dokumente bearbeiten kann. Es wird gezeigt, dass die Ergebnisse einer künstlichen Intelligenz, welche sowohl mit wenigen echten, als auch einem großen Korpus synthetischer Daten trainiert wurde, um ein vielfaches akkurater sind als wenn auf synthetische Daten verzichtet wird. Daraus lässt sich schließen, dass in Situationen, wo echte Trainingsdaten nicht verfügbar sind, synthetische Daten eine brauchbare Alternative darstellen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Listings

Abkürzungsverzeichnis

DIN Deutsches Institut für Normung.

DSGVO Datenschutz-Grundverordnung.

KI Künstliche Intelligenz.

ML Machine Learning / Maschinelles Lernen.

OCRD Initiative for Optical Character Recognition Development (Initative für die Entwicklung von optischer Buchstabenerkennung).

PDF Portable Document Format.

PNG Portable Network Graphics.

1. Einführung & Motivation

Machine Learning / Maschinelles Lernen (ML) erlebt in den letzten Jahren einen großen Aufschwung. Verschiedenste Probleme werden damit angegangen. Die Erwartungen, die man an Lösungen hat sind enorm. Mit der Hilfe von ML sollen Röntgenbilder ausgewertet [WS12], Autos gesteuert [Sti18] und E-Mails automatisch beantwortet werden [KKR+16], und das mindestens so gut wie ein Mensch diese Aufgaben lösen würde.

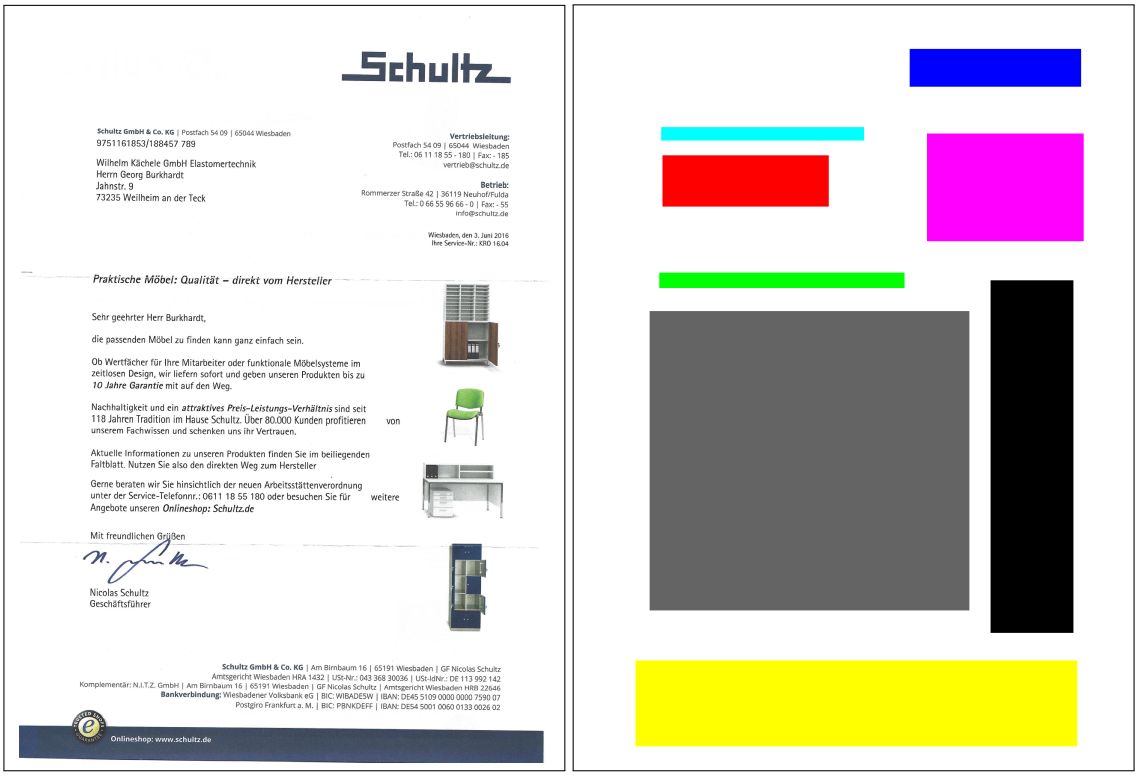


Abbildung 1.1.: Ein echtes Korresponzenzdokument (links) mit dessen Annotation (rechts). Die Annotation muss von Hand erstellt werden, und codiert jede Region in einer eigenen Farbe. Im dargestestellten Dokument ist z.B. die Absenderregion hellblau.

Beim klassischen überwachten Training (ML) werden einem neuronalen Netz Daten zugeführt, die bereits Kategorien zugewiesen wurden. Anhand dieser Daten lernt das Netz, welche Eigenschaften in den Daten mit welcher Kategorie zusammenhängen (Siehe 1.1). Je mehr Daten zu diesem Trainingsprozess vorhanden sind, desto besser kann das Netz diese Eigenschaften feststellen. Ist dieser Trainingsprozess beendet, kann man dem trainierten Netz, welches nun auch als Künstliche

Intelligenz (KI) bezeichnet wird, Daten zuführen, zu denen noch keine Kategorien festgelegt sind. Das Netz trifft dann eine Vorhersage, basierend auf den erlernten Eigenschaften, wo sich eine gewisse Region befinden könnte (Siehe Abbildung 1.2).

Abbildung 1.2.: Links das Originaldokument, bei dem die Absenderadresse erkannt werden soll. Rechts die Vorhersage einer künstlichen Intelligenz, die auf das Erkennen der Absenderzeile trainiert wurde. Je heller der Bereich in der Wahrscheinlichkeitskarte, desto höher die vorhergesagte Wahrscheinlichkeit.

Da die KI nur Eigenschaften erkennt, welche in den Trainingsdaten in ausreichender Zahl vertreten sind, ist es wichtig, sämtliche Sonderfälle auch gut in den Daten zu repräsentieren. Die Qualität der KI hängt somit direkt von den Trainingsdaten ab.

Wie von Garg et al. gezeigt wurde, ist dies bei realen Daten ein Problem [GSJZ18]. Da im Trainingsprozess nicht zwischen Korrelation und Kausalität unterschieden wird, können leicht Zusammenhänge konstruiert werden, die nicht existieren. Während in einem solchen Fall der Trainingsprozess optimiert werden kann, oder die Ergebnisse manuell bewertet werden müssen, liegt der Fehler nicht immer bei der Interpretation.

Echte Daten enthalten zwangsweise die Vorurteile und Fehler derer, die sie erstellt haben [RSC19]. Da man aber nur schwer den Entscheidungsprozess einer künstlichen Intelligenz nachvollziehen kann und diese somit wie eine „Black Box" behandelt wird, ist es schwierig, diese vorurteilsbehafteten Ergebnisse als solche zu entlarven.

Ein weiteres, viel praktischeres Problem ist die Beschaffung von Trainingsdaten. Um eine gut funktionierende KI zu trainieren sind typischerweise tausende annotierte Beispieldaten aus dem Anwendungsbereich nötig. Da diese in aller Regel von Hand annotiert werden müssen ist es

Problem	Beschreibung
Verunreinigte Daten	Beim Beschaffen der Daten können Praktiken verwendet werden, oder Fehler gemacht worden sein, welche die Daten verfälschen.
Große Datenmengen nötig	Korpora mit mehreren tausend Einträgen sind gewünscht, die Beschaffung ist schwierig.
Datenschutz	Das Sammeln personenbezogener Daten wird durch die DSGVO stark erschwert / unmöglich gemacht.

Tabelle 1.1.: Probleme beim Machine Learning und mögliche Lösungen

besonders mühsam und kostspielig, diese Trainingsdaten zu beschaffen [ZEP07]. Für viele Bereiche des ML stehen zwar immer bessere und größere Trainingskorpora zur Verfügung, viele spezialisierte Anwendungen leiden aber stark unter dem Fehlen solcher Korpora. Dieser Zustand wird sich auch in absehbarer Zukunft nicht ändern, viele spezifische Problemstellungen werden schlichtweg nie wichtig genug sein um die Annotation großer Datenmengen zu rechtfertigen.

Zuletzt stellt auch der Datenschutz ein Problem dar. Das Bewusstsein der Bevölkerung wird immer mehr auf den Umgang von Nutzerdaten gelenkt, „Big Data“ ist für viele ein Feindbild und den wenigsten gefällt der Gedanke, dass Google umfangreiche persönliche Daten für alle möglichen Zwecke auswertet. Selbstverständlich spielt auch die seit Mai 2018 gültige, neue Datenschutz-Grundverordnung (DSGVO)[1] eine große Rolle. Sie setzt viele neue Beschränkungen im Umgang mit personenbezogenen Daten, was eine Nutzung für das Trainieren von künstlichen Intelligenzen behindert. Idealerweise muss beim Trainieren also auch auf eine gewisse Anonymität der Daten geachtet werden, jedoch ohne nützliche Informationen zu verlieren.

Ein vielversprechender Ansatz, der die eben genannten und in Tabelle 1.1 zusammengefassten Probleme löst, ist die Nutzung von synthetischen Daten. Das bedeutet, dass die für das Training einer künstlichen Intelligenz benötigten Daten nicht aus echten Quellen stammen, sondern im Voraus künstlich erzeugt wurden. Dabei muss zwar auf eine möglichst realistische Umsetzung geachtet werden, es kann aber auf der Basis weniger Beispiele ein umfassender Korpus generiert werden [SPT+16]. Mit dieser Methode kann die künstliche Intelligenz die entscheidenden Informationen erlernen, für die normalerweise große Datensätze benötigt werden. Der große Vorteil von diesen selbstgenerierten Daten ist, dass auch die Annotation automatisiert erfolgen kann, man spart sich also das aufwändige Annotieren von Hand.

In dieser Arbeit wird untersucht, wie gut der Ansatz mit synthetischen Daten funktioniert, wenn man das daraus erzeugte Modell auf echte Daten anwendet. Es wird untersucht, ob die Trefferquote einer künstlichen Intelligenz, welche zusätzlich zu wenigen realen Daten mit den synthetischen Daten trainiert wurde, tatsächlich höher ist als die einer künstlichen Intelligenz, die nur wenige realen Daten zur Verfügung hatte. Der konkrete Anwendungsfall ist Briefkorrespondenz, auf der die Absenderdaten erkannt werden sollen.

Das Annotieren von Briefen ist sehr aufwändig und die Vielzahl möglicher Layouts legt den Einsatz von ML nahe. Dabei weisen viele Briefe die gleichen Grundelemente und eine ähnliche Struktur auf, was ihre Synthese erleichtert.

[1] https://eur-lex.europa.eu/eli/reg/2016/679/oj

Die Möglichkeiten und Hindernisse bei der Arbeit mit synthetischen Trainingsdaten kommen bei diesem Beispiel gut zur Geltung, und Erkenntnisse können auf andere Gebiete des ML angewendet werden.

Im Kapitel 2 wird der aktuelle Fortschritt bei der Arbeit mit synthetischen Daten erläutert, sowie die Arbeit, die in verwandten Gebieten geleistet wird.

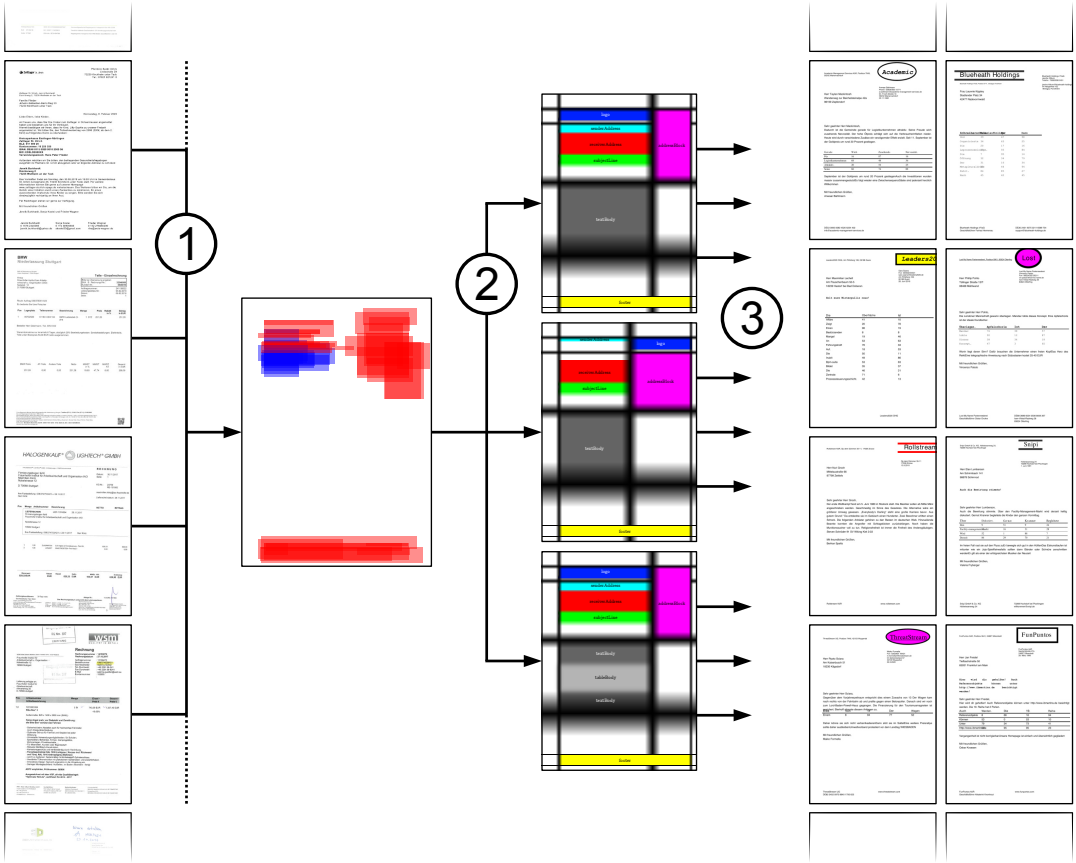


Abbildung 1.3.: Übersicht des Prozesses zur Generierung der synthetischen Daten. Schritt (1), die Analyse echter Daten wird in Kapitel 3 beschrieben, ebenso die Konkretisierung in drei Layouts (Schritt (2)). Schritt (3), wo gezeigt wird wie diese Layouts programmiert sind, und wie auf ihrer Grundlage annotierte Dokumente generiert werden, wird in Kapitel 4 beschrieben.

In Abbildung 1.3 wird der Prozess von Beispieldokumenten zu synthetischen Dokumenten dargestellt. Die Daten, die dieser Anwendung zugrunde liegen, werden in Kapitel 3 erläutert, sowohl in ihrer Struktur (Abschnitt 3.1) als auch in ihrem Inhalt (Abschnitt 3.2). Wie die synthetischen Daten, basierend auf den Erkenntnissen aus Kapitel 3 dann konkret erstellt werden, wird in Kapitel 4 beschrieben. Dort wird auch die Programmstruktur, sowie die der Synthetisierung zugrunde liegenden Datenquellen dargestellt.

Eine Bewertung der mit den synthetischen Daten erzielten Ergebnisse sowie eine Analyse aufgetretener Fehler ist in Kapitel 5 zu finden.

Eine Zusammenfassung der Arbeit, sowie ein Ausblick auf zukünftige Arbeiten in diesem Bereich und ein abschließendes Fazit sind im Kapitel 6 zu finden.

2. Stand der Technik & verwandte Themen

In diesem Kapitel wird beschrieben, wie erfolgreich das Training von künstlichen Intelligenzen mittels synthetischen Daten bereits eingesetzt wurde, sowie der Fortschritt bei der Erkennung von Absenderdaten. Auch Fortschritte in ähnlichen Disziplinen werden beschrieben.

2.1. Synthetische Trainingsdaten

Der in dieser Arbeit verwendete Ansatz, synthetische Trainingsdaten zu verwenden, wurde in anderen Bereichen bereits erfolgreicht umgesetzt. Die folgenden Ergebnisse ermutigten dazu, diese Technik auch in anderen Disziplinen anzuwenden.

Synthetische Daten werden schon lange genutzt, um die Datenmenge, auf der eine künstliche Intelligenz trainiert wird, zu vergrößern. Ein einfacher Ansatz dazu ist es, Variationen von annotierten echten Daten zu generieren. So wurde schon im Jahr 2003 von Tamas Varga und Horst Bunke ein solcher Ansatz zum Generieren neuer Trainingsdaten für das Erkennen von Handschrift eingesetzt [VB03]. In diesem Ansatz wurden die bestehenden Trainingsdaten durch verschiedene Techniken abgewandelt. Diese wurden so gewählt, dass sie nicht von standardmäßigem Pre-Processing rückgängig gemacht werden können.
Mit standardmäßigem Pre-Processing ist gemeint, dass viele Machine-Learning-Architekturen Trainingsdaten in einem Vorverarbeitungs-Schritt so vereinheitlichen, dass später nicht Abweichungen in der Rotation, Größe oder Farbe in den Trainingsprozess einfließen.
Indem Techniken gewählt werden, die sich nicht durch diese Vorverarbeitung umkehren lassen, haben sie auf das Machine Learning den gleichen Effekt wie komplett neue Daten. Auf diese Art können aus einem annotierten handschriftlichen Textstück viele weitere generiert werden. Das verbessert die Abdeckung von Varianten der bekannten Zeichen und erhöht somit die Qualität des Trainingskorpus. Dennoch kann nicht auf einen annotierten Datensatz verzichtet werden, sämtliche Probleme, die beim klassischen ML auftreten sind also nach wie vor präsent. In dem Anwendungsfall der Handschrifterkennung ist das zwar vermutlich kein großes Problem, aber es verhindert das Generalisieren dieses Ansatzes auf andere Problemstellungen, in denen Datenschutz, Vorurteile und unreine Daten ein Problem darstellen können.

Ein neuer Ansatz von Chernyshowa, Gayer und Sheshkus aus dem Jahr 2018 befasst sich ausdrücklich mit dem Problem der kompletten Abwesenheit echter Trainingsdaten [CGS18]. Um Trainingsdaten für das automatisierte Auslesen russischer Personalausweise zu erstellen, können wegen Datenschutz keine echten Ausweise benutzt werden. Chernyshowa et al. nutzen dementsprechend komplett frei generierte Daten, um der künstlichen Intelligenz den Umgang mit verschiedenen Schriftarten und der Überlagerung von Schriftzeichen mit Wasserzeichen oder anderen Sicherheitsmerkmalen beizubringen. Um die synthetischen Daten mit möglichst vielen echten Eigenschaften zu versehen, werden zum Beispiel Unschärfe, Bildrauschen, Helligkeitsunterschiede und Bewegung von Kamera

oder Objekt simuliert. Somit werden realitätsnahe Daten erzeugt, und eine mit ihnen trainierte Maschine, die auf echten Personalausweisen getestet wurde hatte eine Fehlerquote von 1,2 %. Diese ist zwar höher als die, eines mit echten Daten trainierten Modells, welches eine Fehlerquote von 0,2 % erreichte, kommt aber komplett ohne echte Datensätze aus. Deswegen stellen synthetische Daten in vielen Situationen eine Alternative zu echten Daten dar.

Von Shrivastava et al. wurde untersucht, wie man realistischere synthetische Trainingsdaten generieren kann [SPT+16]. Ein Problem, welches mit synthetischen Trainingsdaten auftreten kann, ist nämlich das sogenannte „overfitting“. Dabei werden von der künstlichen Intelligenz Eigenschaften von synthetischen Daten gelernt, die nicht in echten Daten auftreten. Darunter leidet die Qualität der Ergebnisse, weshalb es overfitting zu vermeiden gilt. Shrivastava et al. schlagen dazu den Ansatz „Simuliertes + Unüberwachtes“ (simulated + unsupervised) Lernen vor. Dabei werden synthetische Daten von einem zweiten, neuronales Netz mit unannotierten, echten Daten kombiniert, sodass die Annotation der synthetischen Daten erhalten bleibt. Um die daraus entstandenen, vermeintlich realitätsnahen Daten als solche zu bestätigen, werden sie einem dritten neuronalen Netz vorgelegt, welches darauf trainiert ist, synthetische Daten zu erkennen. Erkennt dieses die optimierten Daten nicht, kann sie also nicht von echten Daten unterscheiden, werden sie zum Training genutzt.

Dieser Ansatz wurde im Zusammenhang mit Bildern genutzt, wo Details wie die Textur von Haut oder Kantenunschärfe einen großen Einfluss spielen. Analog dazu sind bei der Analyse von Bildern Details wie Musterungen im Papier oder Unreinheiten beim Scannen schwer realistisch darstellbar.

2.2. Absenderdatenerkennung

Auch das Gebiet der Absenderdatenerkennung wurde in anderen Arbeiten schon behandelt, auch kleinere Vorstöße in Richtung der Datensynthetisierung wurden bereits umgesetzt. Im Folgenden wird ein moderner Ansatz genauer erläutert.

Das Gebiet der Absenderdatenerkennung aus Briefkorrespondenz lässt sich auf die Problemstellung der Segmentierung von Dokumenten verallgemeinern.

Segmentierung von Dokumenten bedeutet, ihren Inhalt verschiedenen Klassen zuzuordnen, also Bildteile mit bestimmten Kategorien zu versehen. Diese Einteilung erfolgt basierend auf Platzierung, Layout, Größe und Erscheinungsbild der Objekte im Dokument. Der Inhalt spielt bei der Segmentierung historischer Dokumente eine untergeordnete Rolle. Manuelle Segmentierung ist in der Regel leicht, da der Mensch mit den Kriterien vertraut ist und diese nahezu unterbewusst anwendet. Um einer Maschine diese Kriterien möglichst robust beizubringen, muss eine Architektur geschaffen werden, welche auch auf Abweichungen wie Rotation oder Skalierung von Dokumenten eingestellt ist, sowie die Beziehungen der Elemente untereinander versteht.

Dafür haben Oliviera, Seguin und Kaplan im Jahr 2018 einen Deep-Learning Ansatz vorgestellt [OSK18]. Dieser versucht, das Problem der Dokumentenverarbeitung nicht mit speziell angepassten Lösungen anzugehen, sondern eine generische Herangehensweise für den Umgang mit Dokumenten anzubieten.

Dieses öffentlich zugängliche Werkzeug „dhSegment“ wurde mit Erfolg dazu eingesetzt, mit existierenden Trainingsdatensätzen historische Dokumente zu segmentieren. Dabei konnte zwischen Textkörper, Marginalie und Ornament unterschieden werden, so wie es die Trainingsdatensätze vorgaben. Um auch kleinere Datensätze effizient nutzen zu können, und robuste Ergebnisse zu erzeugen, werden Strategien wie Rotation, Skalierung und Spiegelung auf die Trainingsdaten angewendet. Als primitive Form der Datensynthese können auf diese Art Trainingsdaten mehrmals verwendet werden und somit die Qualität der künstlichen Intelligenz weiter verbessern.

DhSegment wurde auch im Rahmen dieser Arbeit verwendet, da sich auch das Problem der Absenderdatenerkennung auf das Segmentieren von Dokumenten verallgemeinern lässt.

3. Daten

In diesem Kapitel wird beschrieben, wie die zu verarbeitenden Dokumente aussehen. Anhand dieser Ergebnisse wird dann erläutert, mit welchen Ansätzen die synthetischen Dokumente diesen realen Vorbildern möglichst nahe kommen sollen.

Um den Umfang der Aufgabenstellung zu verstehen, wurden zunächst dreizehn zufällige Beispieldokumente gesammelt, welche später auch von der künstlichen Intelligenz bearbeitet werden sollen. Diese Dokumente wurden dann mit Fokus auf Absender- und Empfängerdaten betrachtet, und dann hinsichtlich der Struktur und des Inhaltes ausgewertet.

3.1. Struktur

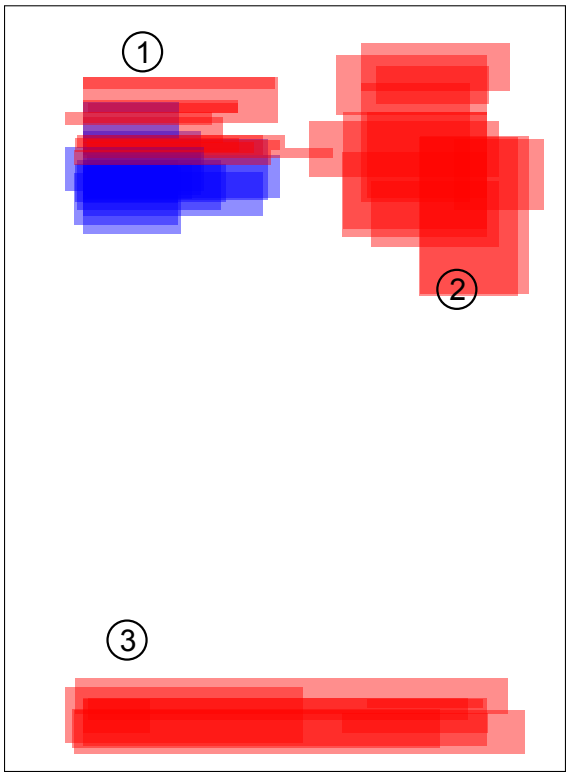


Abbildung 3.1.: Positionen der Absender- und Empfängerdaten in den zufälligen Beispieldokumenten. Absenderdaten sind rot, Empfängerdaten blau markiert.

Bei der graphischen Auswertung in Abbildung 3.1 fällt auf, dass die Daten in drei Regionen einzuteilen sind: Die Absenderadresse im Adressfeld ①, einen Block mit Adressdaten, welcher im Folgenden „Adressblock“ genannt wird ②, sowie die Fußzeile ③. Die Empfängerdaten sind dahingegen nur dort zu finden, wo sie bei einem DIN-A4-lang-Briefumschlag mit Sichtfenster erkennbar sind. Das heißt auch, dass sie in hoher Gefahr stehen später mit den Absenderadressen in Position ① verwechselt zu werden.

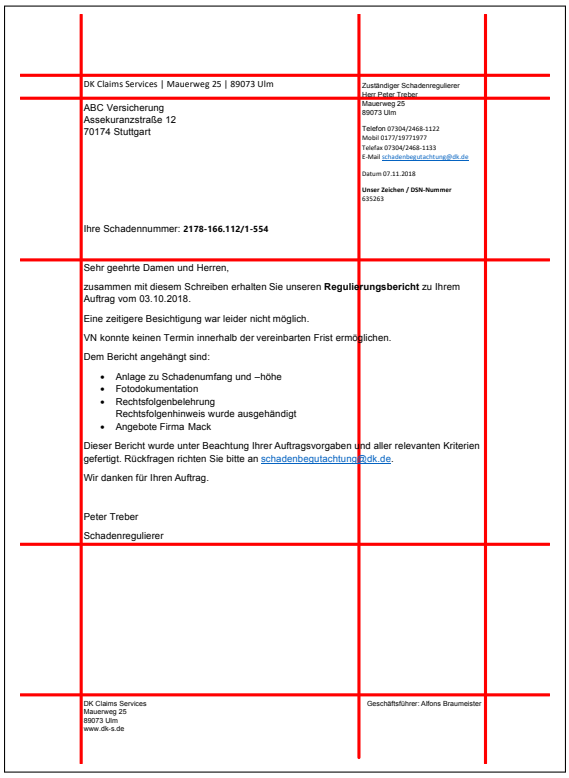


Abbildung 3.2.: Beispiel für die Grundstruktur der Briefe

Außerdem fällt auf, dass alle Briefe eine gleiche optische Struktur haben. Diese lässt sich mit horizontalen und vertikalen Geraden beschreiben, an denen sich die verschiedenen Elemente auf der Seite ausrichten (Siehe Abbildung 3.2).

3.2. Inhalt

In Tabelle 3.1 ist der Inhalt der in Abbildung 3.1 definierten Regionen dargestellt.

Diese Auflistung macht deutlich, dass jeder Eintrag der Tabelle 3.1 für eine Briefsynthese mit künstlich erzeugten Daten nachgeahmt werden muss, da diese Daten aus Korrespondenz extrahiert werden könnnen. Zudem müssen die Daten für den Empfänger erzeugt werden. Diese bestehen aus Vor- und Nachname, Straße, Hausnummer, Postleitzahl und Ort. Um eine korrekte Anrede zu ermöglichen ist auch das Geschlecht von Nutzen.

Position ①		Position ②		Position ③	
Unternehmens-Name	12/13	Unternehmens-Name	5/13	Unternehmens-Name	4/13
Anschrift	13/13	Anschrift	10/13	Anschrift	3/13
davon mit Postfach	6/13	Sachbearbeiter-Name	9/13	Geschäftsführer	5/13
Land	2/13	Sachbearbeiter-Telefon	10/13	Bankverbindung	4/13
		Sachbearbeiter-E-Mail	8/13	Unternehmens-Telefon	3/13
		Website	3/13	Unternehmens-E-Mail	2/13
		Datum	6/13	Unternehmens-Fax	1/13
				Website	2/13
				Steuernummer	2/13
				Amtsgericht	2/13
				Öffnungszeiten	1/13

Tabelle 3.1.: Die Häufigkeit verschiedener Daten in den beobachteten Regionen aus Abbildung 3.1. Die „Anschrift“ besteht aus Straße, Hausnummer, Postleitzahl und Ortsname.

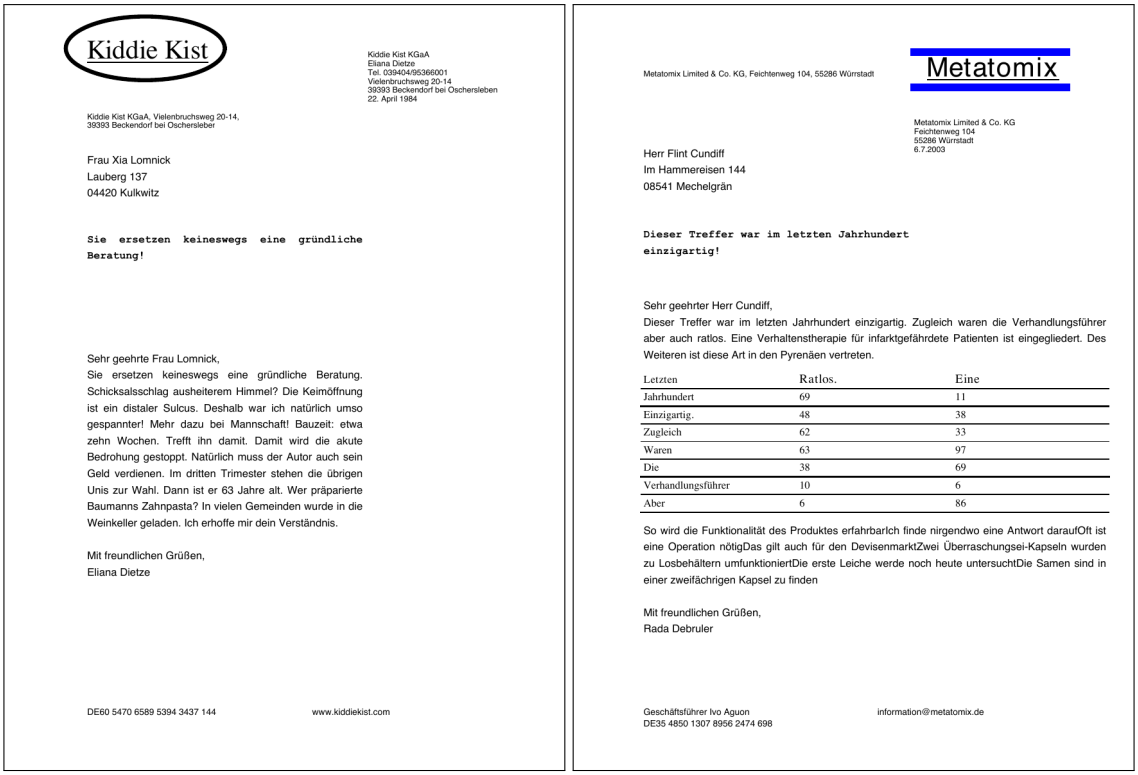
Außerdem wurde anhand der Beispieldokumente klar, dass auch grafische Elemente wie Logos umgesetzt werden müssen, um das Briefbild realistisch wirken zu lassen. Auch Auffälligkeiten wie Tabellen und Betreffzeilen prägen das Erscheinungsbild vieler Dokumente, und sollten deswegen auch bei der Synthese berücksichtigt werden.

3.3. Ergebnisse

Die synthetischen Daten, welche auf der Grundlage dieser Beobachtungen generiert wurden, weisen die gleiche Struktur auf wie die echten Dokumente. Außerdem werden die Daten mit der gleichen Wahrscheinlichkeit in den verschiedenen Regionen dargestellt wie in den echten Dokumenten (Siehe 3.3). Es handelt sich stets um einseitige Briefe, die untereinander keinen Bezug haben, und über den Postweg empfangen worden wären.

Das Logo sorgt dafür, dass auch der Umgang mit graphischen Elementen, Formen und Farben von der künstlichen Intelligenz erlernt wird. Die Absenderadresse wird gegebenenfalls mit Umbruch dargestellt, wie es auch bei echten Briefen der Fall ist, und hat eine kleinere Schriftgröße als andere Textelemente. Ihr Abstand zur Empfängeradresse ist, wie auch die Schriftgröße und eine Unterstreichung, variabel. Unter der Empfängeradresse ist eine Betreffzeile, die sich durch die fette Schrift von den anderen Textelementen abhebt. Sie wurde platziert, da die Nähe zur Absenderadresse die Gefahr erhöht, dass hier fälschlicherweise ähnliche Elemente mit der Absenderadresse verwechselt werden. Insofern ist es sinnvoll, die künstliche Intelligenz auf viele solcher Fälle vorzubereiten indem man ihr ausreichend Beispiele liefert.

Die Ansprache des Textes nutzt den in der Adresse vorkommenden Namen des Empfängers bzw. der Empfängerin, sowie die passende Anrede (Frau / Herr). Der Text selbst setzt sich aus zufälligen Sätzen zusammen, welche als Blindtext dienen. Unterzeichnet wird der Text von dem zuständigen Sachbearbeiter. Oft werden dessen Kontaktdaten im Adressblock oben rechts dargestellt.



Kiddie Kist

Kiddie Kist KGaA
Eliana Dietze
Tel. 039404/95366001
Vielenbruchsweg 20-14
39393 Beckendorf bei Oschersleben
22. April 1984

Kiddie Kist KGaA, Vielenbruchsweg 20-14,
39393 Beckendorf bei Oschersleber

Frau Xia Lomnick
Lauberg 137
04420 Kulkwitz

Sie ersetzen keineswegs eine gründliche Beratung!

Sehr geehrte Frau Lomnick,
Sie ersetzen keineswegs eine gründliche Beratung. Schicksalsschlag ausheiterem Himmel? Die Keimöffnung ist ein distaler Sulcus. Deshalb war ich natürlich umso gespannter! Mehr dazu bei Mannschaft! Bauzeit: etwa zehn Wochen. Trefft ihn damit. Damit wird die akute Bedrohung gestoppt. Natürlich muss der Autor auch sein Geld verdienen. Im dritten Trimester stehen die übrigen Unis zur Wahl. Dann ist er 63 Jahre alt. Wer präparierte Baumanns Zahnpasta? In vielen Gemeinden wurde in die Weinkeller geladen. Ich erhoffe mir dein Verständnis.

Mit freundlichen Grüßen,
Eliana Dietze

DE60 5470 6589 5394 3437 144 www.kiddiekist.com

Metatomix Limited & Co. KG, Feichtenweg 104, 55286 Würrstadt

Metatomix

Metatomix Limited & Co. KG
Feichtenweg 104
55286 Würrstadt
6.7.2003

Herr Flint Cundiff
Im Hammereisen 144
08541 Mechelgrän

Dieser Treffer war im letzten Jahrhundert einzigartig!

Sehr geehrter Herr Cundiff,
Dieser Treffer war im letzten Jahrhundert einzigartig. Zugleich waren die Verhandlungsführer aber auch ratlos. Eine Verhaltenstherapie für infarktgefährdete Patienten ist eingegliedert. Des Weiteren ist diese Art in den Pyrenäen vertreten.

Letzten	Ratlos.	Eine
Jahrhundert	69	11
Einzigartig.	48	38
Zugleich	62	33
Waren	63	97
Die	38	69
Verhandlungsführer	10	6
Aber	6	86

So wird die Funktionalität des Produktes erfahrbarIch finde nirgendwo eine Antwort daraufOft ist eine Operation nötigDas gilt auch für den DevisenmarktZwei Überraschungsei-Kapseln wurden zu Losbehältern umfunktioniertDie erste Leiche werde noch heute untersuchtDie Samen sind in einer zweifächrigen Kapsel zu finden

Mit freundlichen Grüßen,
Rada Debruler

Geschäftsführer Ivo Aguon
DE35 4850 1307 8956 2474 698
information@metatomix.de

Abbildung 3.3.: Beispiele von synthetisch generierten Briefen.

Dieser Adressblock enthält zufällige Daten, die nach der gleichen Wahrscheinlichkeit auftauchen wie in echten Dokumenten. Ebenso wird auch die Fußzeile mit zufälligen, aber innerhalb des Briefes konsistenten Daten gefüllt. Diese Daten sind entweder aus echten Datensätzen extrahiert, oder nach möglichst realistischen Maßnamen erzeugt.

Details zur Umsetzung der Synthese sind in Kapitel 4 erklärt, oder im Programmcode im Anhang zu finden (Kapitel A).

4. Implementierung

Im Folgenden wird die konkrete Umsetzung der Briefsynthese beschrieben. Das Programm wurde in Python[1] geschrieben.

Die Struktur des Programmes wird in Abbildung 4.1 dargestellt. In der `main`-Methode werden die verschiedenen Funktionen, die für das Generieren realistischer Trainingsdaten nötig sind, zusammengefasst. Für jeden Brief werden zuerst in der `createData`-Methode die Daten zusammengestellt, mit denen später das Dokument gefüllt wird (Siehe Abschnitt 4.1). Mit diesem Datensatz wird dann die `createPDF`-Methode aufgerufen. Sie ruft die `applyLayout`-Methode auf, welche zufällig ein vordefiniertes Layout wählt, welches in sich auch noch Variationen aufzeigt (Siehe Abschnitt 4.2). Dieses Layout wird dann in der `createTextBlocks`-Methode mittels des in `createData` definierten Datenpakets in Regionen mit definierter Position und Inhalt umgewandelt (Siehe Abschnitt 4.3). Diese Regionen werden in `createPDF` nun in ein Dokument umgewandelt (Siehe Abschnitt 4.5). Hierbei wird auch ein Logo generiert und eingesetzt (Siehe Abschnitt 4.4). Wenn gewünscht, kann auch eine PageXML-Datei [PA10] ausgegeben werden, dazu wird das Modul `createXML` in der `createPDF`-Methode aufgerufen.

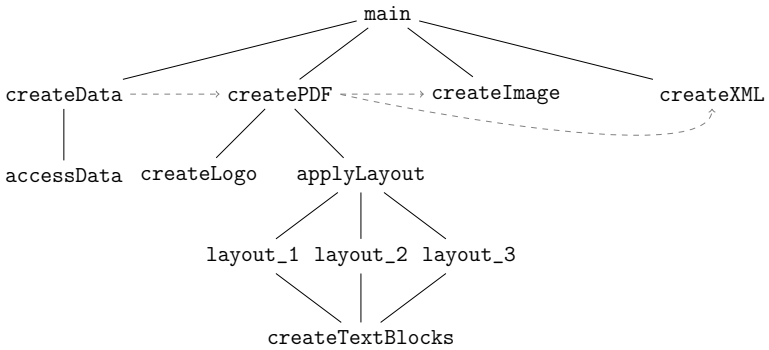


Abbildung 4.1.: Programmstruktur. Gestrichtelte Pfeile symbolisieren Datenfluss.

[1] https://www.python.org

Attribut	Beispiel	Quelle	mögliche Werte
`firstName`	Chiara	Liste der Stadt Köln[2]	7 019
`gender`	w		
`lastName`	Kardos	Onlinekartei[3]	164 385
`street`	Donnersberger Straße	Fraunhofer IAO	443 106
`streetNumber`	120/7	Selbstgeneriert	5 700
`zip`	69168	Onlinekartei[4]	19 670
`city`	Wiesloch		
`country`	Deutschland	Begrenzung auf deutsche Adressen	1

Tabelle 4.1.: Im `receiver`-Objekt enthaltene Datentypen, mit Beispiel, Quelle und der Anzahl möglicher Werte.

4.1. Das `createData`-Modul

In dem Modul `createData` wird das Objekt `letterData` erstellt. Es umfasst alle inhaltlichen Daten, die der Brief umfassen könnte, wie in Tabelle 3.1 genannt. Die Daten sind nicht in verschiedene Regionen vorsortiert, da dieses Objekt als einheitliche „Quelle“ verwendet wird, welche in verschiedenen Regionen abgerufen werden kann.

```
class LetterData:
    def __init__(self):
        self.receiver = Receiver()
        self.company = Company()
        self.clerk = Clerk(self.company)
        self.date = data.getRandomDate()
        self.text = generateText(random.randrange(60, 120))
```

Listing 4.1: Struktur des `letterData`-Objekts. Das `receiver`-Objekt wird in Tabelle 4.1 beschrieben, das `company`-Objekt in Tabelle 4.2 und das `clerk`-Objekt in Tabelle 4.3.

Ein Objekt der Klasse `letterData` besteht, wie in Listing 4.1 dargestellt, aus einem Empfängerobjekt (`receiver`), einem Firmenobjekt (`company`), dem Sachbearbeiter (`clerk`) sowie einem zufällig gewählten Versanddatum (`date`) und dem Textblock (`text`). Diesen Objekten wurden die, in den realen Dokumenten beobachteten, Datentypen (Siehe Tabelle 3.1) zugeordnet.

[2] Veröffentlicht von der Redaktion der Stadt Köln am 19. Januar 2018 unter Linzenz Creative Commons Namensnennung 3.0 DE: https://offenedaten-koeln.de/dataset/vornamen

[3] https://github.com/philipperemy/name-dataset/blob/master/names_dataset/last_names.all.txt

[4] https://www.datendieter.de/item/Postleitzahlen-Datenbank_Deutschland

[5] https://gist.github.com/jvilledieu/c3afe5bc21da28880a30

[6] https://de.wikipedia.org/wiki/Liste_von_Rechtsformen_von_Unternehmen_in_Deutschland

[7] Siehe Fußnote 4

Attribut	Beispiel	Quelle	mögliche Werte
name	Farelogix	Onlinekartei[5]	47 759
website	www.farelogix.com		
eMail	support@farehelper.de	Mit name selbstgeneriert	9
legalForm	Stiftung & Co. KG	Wikipedia-Eintrag[6]	36
street	Siehe Tabelle 4.1 bei „street“		443 106
streetNumber	Siehe Tabelle 4.1 bei „streetNumber“		5 700
zip	06333	Onlinekartei[7]	19 670
city	Wiederstedt		
areaCode	034785		
state	Baden-Württemberg		
country	Siehe Tabelle 4.1 bei „country“		1
postbox	Postbox 1523	Selbstgeneriert	8 999
phone	Telefon: 06222 124093-1	Mit areaCode selbstgeneriert	159 983 984
fax	Fax: 06222 124093-2		
bankInfo	DE79 6688 3765 5748 5224 071	Selbstgeneriert	10^{18}
taxNr	1263/9850/813016	Selbstgeneriert	10^{10}
ceo	Jürgen Yldriz	Selbstgeneriert	1 153 818 315

Tabelle 4.2.: Im company-Objekt enthaltene Datentypen, mit Beispiel, Quelle und der Anzahl möglicher Werte.

Attribut	Beispiel	Quelle	mögliche Werte
gender	Siehe Tabelle 4.1 bei „gender“		7 019
firstName	Siehe Tabelle 4.1 bei „firstName“		
lastName	Siehe Tabelle 4.1 bei „lastName“		164 385
eMail	petra.kübel@farelogix.de	Mit firstName, lastName und companyName selbstgeneriert	4
phone	Siehe Tabelle 4.2 bei „phone“		159 983 984
fax	Siehe Tabelle 4.2 bei „fax“		

Tabelle 4.3.: Im clerk-Objekt enthaltene Datentypen, mit Beispiel, Quelle und der Anzahl möglicher Werte.

Viele der insgesamt 35 Attribute werden aus öffentlich zugänglichen Datensätzen generiert, so trägt zum Beispiel jede Person einen Vornamen aus einer Liste der beliebtesten Vornamen in Köln im Jahr 2017 (Siehe Tabelle 4.1 unter „firstName“). In dieser Liste sind auch die Geschlechter der Namen hinterlegt, es existieren also so viele Kombinationen aus Vorname und Geschlecht wie Einträge hinterlegt sind, also 7 019.

Manche Attribute, wie zum Beispiel die Firmen-E-Mail (Siehe Tabelle 4.2 unter eMail benötigt zur Generierung ein anderes Attribut. In diesem Fall basiert eMail auf dem name-Attribut des Unternehmens. Der Wert in „mögliche Werte“ gibt hier an, wie viele Möglichkeiten mit diesem vorgegebenen Unternehmens-Namen noch generiert werden können.

In den Fällen, wo keine langen Listen mit Datenbeispielen verfügbar sind, wird versucht, diese Werte möglichst realistisch selbst zu generieren. Dazu wird, eng am Vorbild orientiert und unter Berücksichtigung möglichst vieler Sonderfälle, ein Wert aus einer Zufallsfunktion generiert. Im Beispiel der Telefonnummer wird zufällig aus einer Liste gewählt, wie sie beschrieben werden soll, ob „Tel.“ geschrieben wird oder „Telefon“, und passend dazu „Fax“ oder „Telefax“. Auch ob die Leserlichkeit der Nummer durch einen Querstrich oder ein Leerzeichen zwischen den Ziffernblöcken verbessert wird obliegt einer zufälligen Entscheidung.

In dem Erstreben, möglichst realistische Daten zu erzeugen, werden auch möglichst viele Daten auf der Grundlage anderer erzeugt. Somit werden nicht nur Vorwahlen passend zu Ortschaften gewählt, auch die Postleitzahlen sind korrekt, Personen mit weiblichen Vornamen werden als „Frau“ und männliche als „Herr“ angesprochen, und die Steuernummer eines Unternehmens wird nach einem Schema generiert, das von dem Bundesland abhängt, in dem das Unternehmen seinen Sitz hat. Auf diese Art werden möglichst viele Daten sehr realitätsnah imitiert, auch mit Hinsicht auf mögliche Varianten in der Schreibweise. Das führt zu einer guten Abdeckung mit realen Briefen, was die Ergebnisse der mit den synthetischen Daten trainierten Modelle verbessert.

Zusätzlich zu diesen Objekten besteht das `letterData`-Objekt noch aus dem Verfassungsdatum `date` und dem Inhalt `text` (Siehe Listing 4.1). Das Verfassungsdatum wird zufällig zwischen dem aktuellen Datum und dem 1. Januar 1970 gewählt. Es wird als einziges Attribut nicht als `String`, sondern als `date`-Objekt hinterlegt. Abgesehen davon wird es aber wie jedes andere Attribut behandelt und auch im `generateTextBlocks`-Modul (Siehe Abschnitt 4.3) in die finale Form gebracht.

Für das `text`-Attribut werden zufällig gewählte Sätze aus der Leipzig Corpora Collection [GEQ12] aneinandergereiht, und dienen somit gewissermaßen als Blindtext. Der in Listing 4.1 übergebene Parameter an die `generateText`-Funktion beschreibt die Länge des Textes in Sätzen und gewährleistet unterschiedlich lange Briefkörper. Von der Verwendung des „Lorem ipsum“-Textes wurde abgesehen, da das trainierte Modell die Textkörper nicht durch das Erkennen des Schriftbilds oder Inhalts erkennen soll.

Die Details zu der Generierung sämtlicher Daten, wie Abhängigkeiten umgesetzt wurden und mit welcher Wahrscheinlichkeit welche Sonderfälle abgedeckt werden, sind im Programmcode im Anhang A.2 näher beschrieben.

Um die Realitätsnähe der Daten weiter zu steigern, ist es in vielen Bereichen denkbar, noch mehr echte Daten zu nutzen und ausgefeiltere Techniken bei der Generierung zu verwenden. Insbesondere ein Matching von Straßennamen auf Städte ist denkbar, auch wenn fragwürdig ist ob dies noch viel Einfluss auf die Qualität der Trainingsdaten hat.

4.2. Das `applyLayout`-Modul

Das im `createData`-Modul generierte Objekt wird, wie durch den gestrichelten Pfeil in Abbildung 4.1 angedeutet, nun dem `createPDF`-Modul übergeben. Dieses übergibt es direkt dem `applyLayout`-Modul, welches den Inhalten nun die Form gibt, die sie auf dem Dokument später haben werden. Um diese Form möglichst abwechslungsreich, aber dennoch in gewissen Vorgaben zu halten, lassen sich verschiedene Layouts definieren, welche als separate Module eingefügt werden.

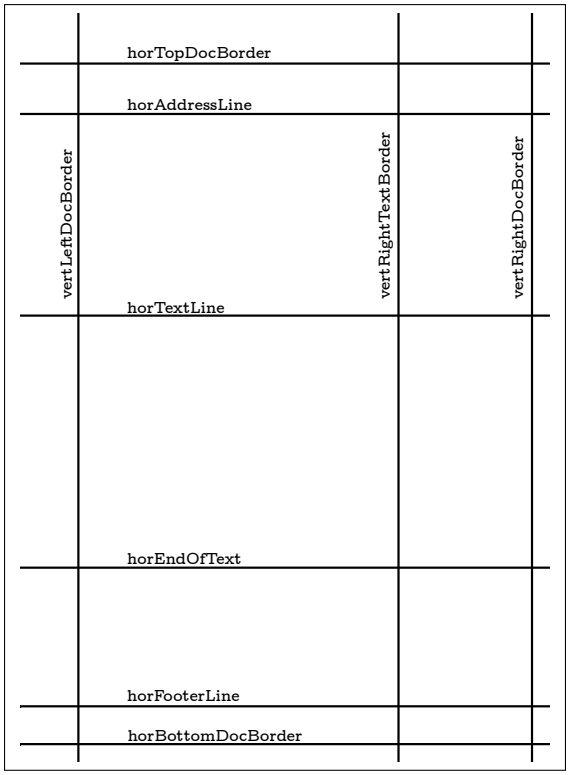


Abbildung 4.2.: Begrenzungslinien der layout-Module

Ein Layout besteht aus Grenzlinien, welche die Grundstruktur des Dokuments vorgeben. Die so definierten Grenzlinien sind für jedes Layout gleich, siehe Abbildung 4.2. Sie wurden aufgrund der in Abbildung 3.2 beobachteten Struktur von Briefen gewählt. Um diese Struktur abwechslungsreich zu gestalten, wird jede Grenzlinie über eine Gaussverteilung bestimmt. Die Standardabweichung für jede Gerade wird in den jeweiligen Layouts definiert, somit treten die Variationen an den gewünschten Stellen im Dokument auf, was für die Qualität der Trainingsdaten ausschlaggebend ist. Das führt zu Ausreißern, aber auch vielen „Standardwerten“, siehe Abbildung 4.3. Innerhalb dieser Linien können in jedem Layout beliebig viele Regionen definiert werden, die jedoch nicht unbedingt von diesen Linien begrenzt werden müssen. Textfelder können auch relativ zu diesen Geraden positioniert werden, so wird zum Beispiel die Höhe der `senderAddress`, `receiverAddress` und `subjectLine`-Regionen, wie in Abbildung 4.3 erkennbar, nicht durch Grenzlinien, sondern durch aus den Grenzlinien berechnete Zwischenlinien begrenzt.

Insgesamt sind drei Layouts definiert, welche für abwechslungsreiche Dokumente sorgen, mit verschiedenen, individuell angepassten Eigenschaften. In einem Fall, in dem andere Layouts relevant sind, ist eine Erweiterung des Programms über das Erstellen neuer Layouts an dieser Stelle leicht möglich.

Neben der Position wird in diesem Schritt auch die Schriftgröße festgelegt und jeder Region wird ein Typ sowie ein Name gegeben. Der Typ legt den grundsätzlichen Inhalt der Region fest, und heißt bei den textlastigen Regionen `TextRegion`. Die Firmenlogo-Region bildet mit dem Regionstyp `Logo` die einzige Ausnahmen dazu. Von diesem Typ hängt ab, wie die Region bei der Darstellung auf dem Portable Document Format (PDF) behandelt wird. Der Name spezifiziert die

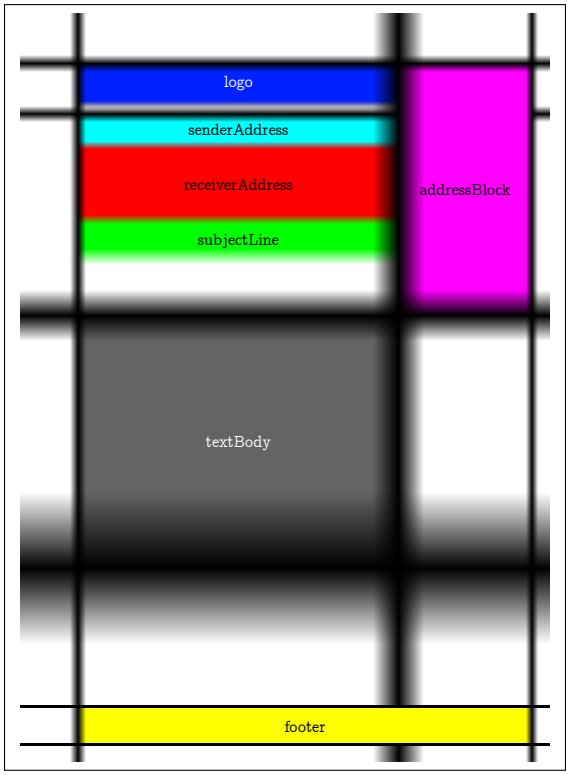


Abbildung 4.3.: Die Implementierung des Layouts im `layout_1`-Modul. Die Breite der Grenzlinien deutet die Standard-Abweichung der Gauss-Verteilung an.

genaue Rolle der Region, und wird dafür genutzt, individuelle Behandlungen mancher Regionen umzusetzen. Der Inhalt jeder Region wird auch festgelegt, dazu wird für jede Region eine Methode des `createTextBlocks`-Moduls aufgerufen.

4.3. Das `createTextBlocks`-Modul

In diesem Modul werden die Daten aus dem `letterData`-Objekt zu Regionen verarbeitet, um in dem Brief an der richtigen Stelle angezeigt zu werden. Dazu müssen sie teilweise noch in eine passende Form gebracht werden.

Für jede, in einem Layout definierte `TextRegion` existiert eine Methode, welche für sie einen möglichst realitätsnahen Inhaltstext zurückgibt. Mögliche Regionen sind `logo`, `senderAddress`, `receiverAddress`, `subjectLine`, `addressBlock`, `textBody`, `tableBody` und `footer`.

Für manche dieser Regionen, wie der Fußzeile `footer` und dem Block mit Kontaktdaten `addressBlock`, werden dafür zufällig Daten aus dem Datensatz gewählt, welche ins Dokument übernommen werden. Die Wahrscheinlichkeiten dafür basieren darauf, wie oft die entsprechende Datenart in der repräsentativen, echten Beispielkorrespondenz vorkommt (Siehe 3.1). Bei ihnen, wie auch bei Absenderadresse (`senderAddress`) und Empfängeradresse (`receiverAddress`) wird jeder gewählte

Eintrag einer Liste übergeben, zusammen mit einem Label, welches die Datenart beschreibt. Dieses Label ermöglicht es später, feingranulare Masken für die Briefe zu erstellen, in denen jeder Datentyp eine andere Farbe hat.

In anderen Regionen, dem Briefinhalt `textBody` zum Beispiel, müssen die Daten des `letterData`-Objekts noch in eine gewisse Form gebracht werden. So wird dem Attribut `text` des `letterData`-Objekts an dieser Stelle noch eine, vom Geschlecht des Empfängers abhängige Begrüßung vorangestellt, sowie eine Grußformel des Sachbearbeiters angehängt. Auch der Tabellen-Region `tableBody` wird ein Inhalt zugewiesen, auch wenn dieser später nicht in dieser Form benutzt wird (Siehe Abschnitt 4.5).

Die `subjectLine`-Region, welche als Betreffzeile am Anfang des Briefes steht, wird einfach aus dem ersten Satz des Textkörpers gebildet, indem diesem ein Ausrufungszeichen angehängt wird.

Sind die Regionen des Layouts mit Inhalten gefüllt, werden sie als Liste dem `applyLayout`-Modul übergeben. Dort werden auf einem Zufallsfaktor basierend Regionen gelöscht, um die Varianz in den generierten Briefen zu erhöhen und die Abdeckung auch auf untypischere Dokumente zu erweitern. Diese eventuell reduzierte Liste von Regionen wird dann der ursprünglich aufrufenden Methode in `createPDF` übergeben.

4.4. Das `createLogo`-Modul

Das `createLogo`-Modul fügt jedem Dokument noch eine zufällig generierte, farbige und grafisch vom restlichen Dokument abweichende Region hinzu. Als einziges überwiegend grafisches Element ist eine Logo-Region nun auch keine `TextRegion`, sondern von der Kategorie `Logo`. Um ein Logo zu generieren wird aus dem Firmennamen, sowie simplen geometrischen Formen ein Objekt erzeugt, welches einem Firmenlogo ähnelt und die synthetischen Daten näher an reale Brieflayouts bringt (Siehe Abbildung 4.4).

Abbildung 4.4.: Beispiele generierter Logos

4.5. Das `createPDF`-Modul

In diesem Modul werden die in `createTextBlocks` festgelegten Inhalte zusammen mit den in `applyLayout` definierten Positionen grafisch umgesetzt. Mittels dem der Freeware PyFPDF[8] werden drei PDF-Dateien pro `letterData`-Objekt erstellt: der Brief in Klartext, eine grobe Labeldatei

[8] https://pyfpdf.readthedocs.io/en/latest/index.html

für das Training mit dhSegment (Siehe Abschnitt 5) und eine feine Labeldatei für das Training mit OCRD-Segment (Siehe Abschnitt 6.1). Die Paare aus Klartext und Label können dann als Trainingsdaten dem entsprechenden Tool übergeben werden und somit Modelle trainiert werden.

Beim Erstellen dieser Dokumente wird jede Region einzeln in die PDF geschrieben. Um Besonderheiten umsetzen zu können wird hier Typ und Name jeder Region berücksichtigt. Für die Label wird die Fläche, die normalerweise der Text einnimmt mit einer vordefinierten Farbe gefüllt, die die jeweilige Region codiert.

Auch grafische Besonderheiten werden hier umgesetzt, darunter die Tabellen der `tableBody`-Regionen. Dazu werden Wörter aus dem Text genommen, welche dann die erste Spalte sowie die erste Zeile füllen. Im Tabellenkörper werden zufällige Zahlen verwendet um den Eindruck einer Tabelle wiederzuspiegeln. Wie viele Trennstriche sie hat, und die Anzahl der Spalten und Zeilen hängt vom Zufall ab. Dieser Schritt ist wichtig um das Briefbild auch an Rechnungen, in denen typischerweise viele Tabellen vorkommen, anzupassen.

Für die `Logo`-Region wird hier ein neues Modul aufgerufen, welches für eine optische Abwechslung sorgt und somit das Modell auf graphische Elemente vorbereitet.

Die kompletten Details zur Implementierung dieses Moduls können dem Code im Anhang A.1 entnommen werden.

5. Evaluation und Fehleranalyse

Im Folgenden wird beschrieben, wie Tests mit den Trainingsdaten umgesetzt werden, nach welchen Eigenschaften sie bewertet werden und an welchen Stellen noch Fehler existieren.

5.1. Durchführung der Tests

Die Evaluation der synthetischen Trainingsdaten fand über das Tool dhSegment statt. DhSegment ist ein von Sofia Ares Oliviera, Benoit Seguin und Frederic Kaplan an der École polytechnique fédérale de Lausanne im Jahr 2018 veröffentlichter und 2019 überarbeiteter generischer deep-learning Ansatz zur Bildsegmentierung [OSK18]. Die Architektur von dhSegment ist ein U-förmiges Convolutional Neural Network. Dieses besteht aus einer verkleinernden und einer erweiternden Hälfte. Die Verkleinernde ist der ResNet-50 [HZRS15] Architektur nachempfunden. Über fünf Stufen wird die Auflösung des Eingangsbildes über Convolution (Faltung) verringert, bei jeder Stufe werden neue „Feature-Maps“ hinzugefügt. Diese werden mit den bereits trainierten Werten befüllt, so entsteht die Vorhersage des Modells. Als Pendant zu dem verkleinernden ResNet-50 existieren in der erweiternden Architektur ebenfalls fünf Stufen. In jeder Stufe werden Features mittels bilinearer Interpolation zusammengefasst, um dann im letzten Schritt jedem Pixel des Originaldokuments zugeordnet werden zu können. Details zu dieser Architektur finden sich in der Arbeit zu dhSegment [OSK18], sowie in den Veröffentlichungen zu ResNet-50 [HZRS15] und U-Net [RFB15].

Die pixelweisen Vorhersagen dieses neuronalen Netzes werden in einem zweiten Schritt zu Blöcken verbunden. Um diesen Schritt effizient und allgemein zu halten, werden dazu nur einfache Standard-Methoden verwendet. Dazu zählt der Einsatz von Schwellwerten, die morphologischen Operationen Erosion und Dilatation sowie eine Analyse zur Feststellung zusammenhängender Komponenten, die nach den vorherigen Operationen noch vorhanden sein können. Abschließend wird dann die verbleibende Form vektorisiert, um sie mit Hilfe von Koordinaten darstellen zu können. Dieser allgemeine Ansatz liefert gute Ergebnisse und kann mit anderen Segmentationsansätzen mithalten. Der Fokus auf einer aufgabenunabhängigen Implementierung führt außerdem dazu, dass das Programm mit wenigen Anpassungen für die verschiedensten Anwendungen verwendet werden kann. Der von Oliviera et al. demonstrierte Einsatz von dhSegment bei der Erkennung von historischen Dokumenten ist nahe an den hier verwendeten Korrespondenzbriefen, somit konnten viele Vor- und Nachverarbeitungseinstellungen übernommen werden. Geändert wurde im Pre-Processing, dass die Briefe weniger stark skaliert und weniger rotiert werden, da bei Briefkorrespondenz wenige Variationen dieser Art auftreten.

In den Labeldateien, die für dhSegment verwendet werden, wurden die Regionen als ganzes annotiert. Es wird also jedes grafische Element auf dem Dokument einem Regionstyp (aus Abschnitt 4.3) zugeordnet (Siehe Abbildung 5.1). Somit kann die Segmentierung die visuellen Auffälligkeiten jeder Region lernen und das dann auf die Vorhersagen anwenden.

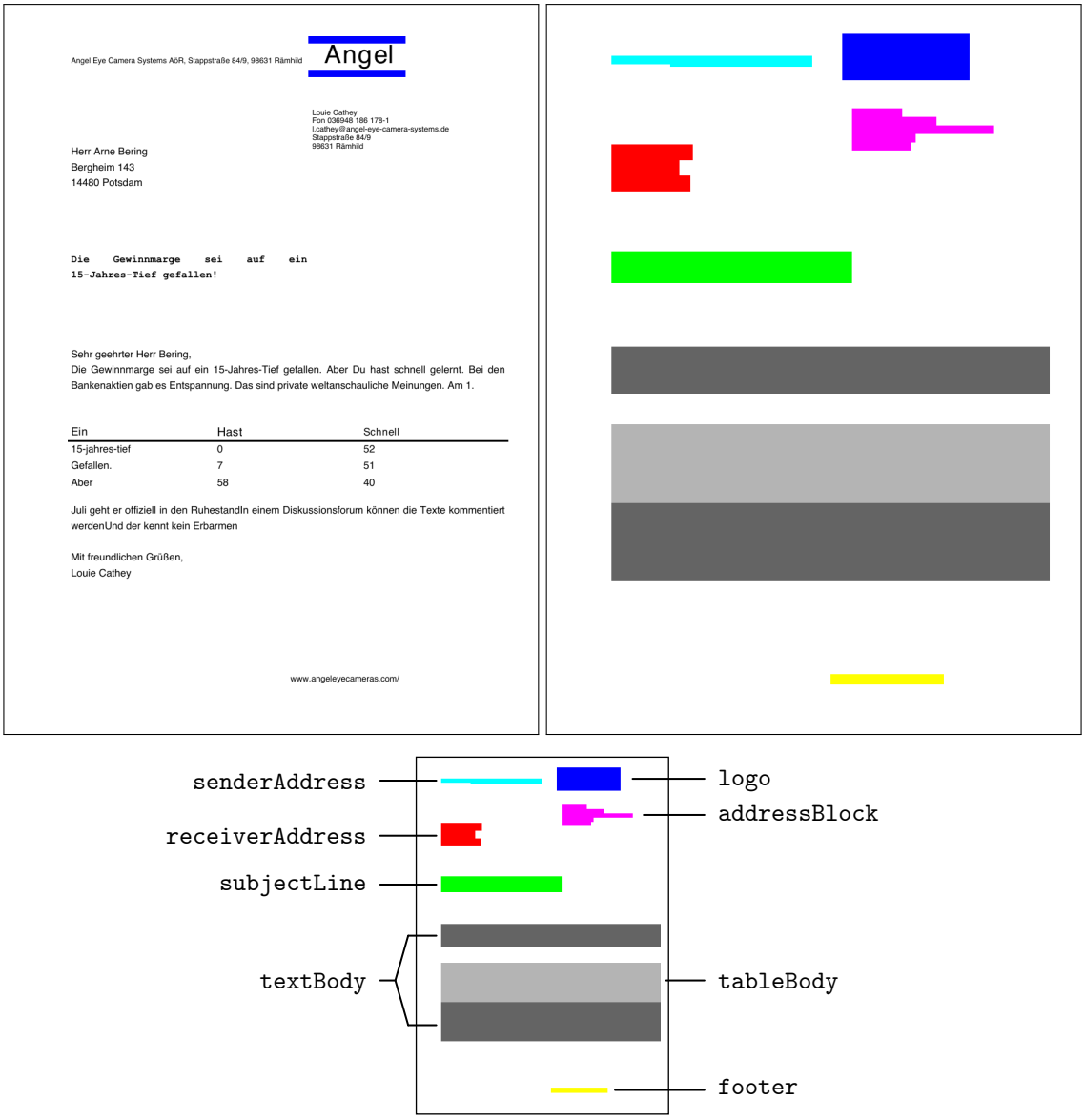


Abbildung 5.1.: Eine für dhSegment verwendete, synthetische Labeldatei (rechts oben), in welcher komplette Textregionen gleichfarbig annotiert sind. Links oben das zugehörige, synthetische Dokument. Darunter eine Erklärung der für die Annotation verwendeten Farben.

Für die Tests wurde ein 1 000 Datenpaare umfassender Korpus synthetischer Daten verwendet, sowie ein Korpus von 58 echten Korrespondenzdaten, die von Hand annotiert wurden.

Um vergleichen zu können, wie groß der Vorteil durch synthetische Daten ist, wurde eine Baseline aus echten Dokumenten trainiert. Damit diese auch an echten Dokumenten, welche nicht trainiert wurden, getestet werden kann, wird hier eine Kreuzvalidierung mit fünf Teilmengen durchgeführt. In Tabelle 5.1 werden die durchschnittlichen Ergebnisse dieser Validierung als Baseline betitelt.

Kreuzvalidierung kommt zum Einsatz, wenn man eine künstliche Intelligenz mit den Dokumenten aus dem gleichen Korpus testet, mit denen man sie auch trainiert. Würde man mit Dokumenten testen, die auch für das Training verwendet wurden könnten die Ergebnisse keine Aussage über die Qualität der KI geben, da man ihr die Lösung bereits gezeigt hat. Stattdessen wird der Korpus von Trainingsdaten im Voraus in Trainings- und Evaluations-Daten geteilt. In diesem Fall wurde er in fünf Teilmengen unterteilt, von denen dann vier zum Training, und eine zur Evaluation verwendet wurden. Damit ein repräsentatives Ergebnis entsteht, und nicht kein Ausreißer möglich ist, wird jede Teilmenge einmal als Evaluation verwendet. Das bedeutet, dass jedes Trainingsdokument genau einmal zur Bewertung eines Modells genutzt wurde (Siehe Abbildung 5.2). Aus den Ergebnissen der Modelle, welche mit den Teilmengen trainiert wurden, wird abschließend der Durchschnitt ermittelt, welcher dann als Indikator für die Qualität der zugrunde liegenden Trainingsdaten fungiert.

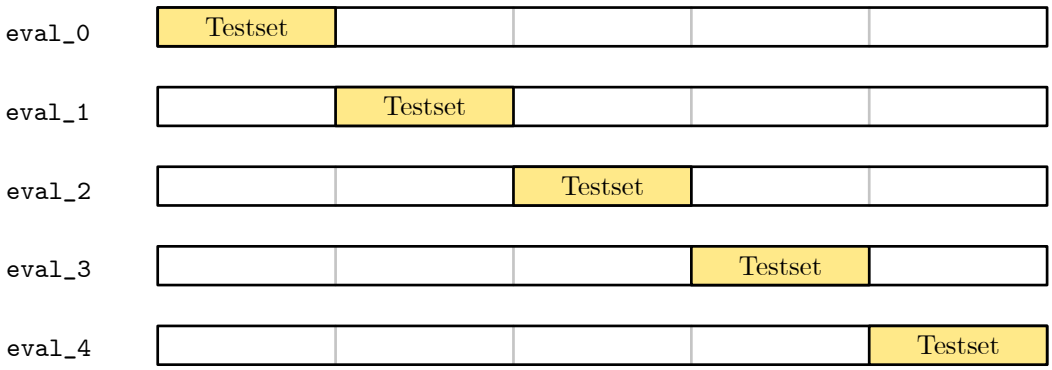


Abbildung 5.2.: Darstellung der Funktionsweise einer fünfstufigen Kreuzvalidierung. Der ganze Balken symbolisiert die Menge aller verfügbaren annotierten Dokumente, die Markierung wird aus diesen Menge entnommen und dient dem Trainingsprozess. So wird jedes Dokument des Korpus auch zur Evaluation genutzt

Das Ergebnis dieser Arbeit, ein Korpus aus 1 000 synthetischen Daten, wurde ebenfalls evaluiert, um die Qualität eines rein synthetisch trainierten Modells bewerten zu können. In Tabelle 5.1 werden diese Ergebnisse als „Ansatz 1“ festgehalten.

Die synthetischen Daten, kombiniert mit einer Teilmenge echter Daten, über Kreuzvalidierung zugewiesen, wurde ebenfalls ausgewertet. Auch diese Kreuzvalidierung verfügt über fünf Teilmengen. Es wurde diese Anzahl an Teilmengen gewählt, da die Ergebnisse von Hand ausgewertet werden mussten, und so eine gute Balance zwischen Verlässlichkeit der Daten und Evaluationsaufwand erzielt wurde. Das Ergebnis dieser Kombination synthetischer und echter Daten wird in der Tabelle 5.1 als „Ansatz 2“ festgehalten.

5.2. Kriterien für Auswertung

Bei der Auswertung der Ergebnisse wurde unterschieden zwischen „erkannt“, „teilweise erkannt“, „nicht erkannt“ und „falsch erkannt“. Beispiele dafür sind in Abbildung 5.3 zu sehen. Als „erkannt“ wurden Ergebnisse gewertet, in denen genau die Absenderadresse markiert wurde, und

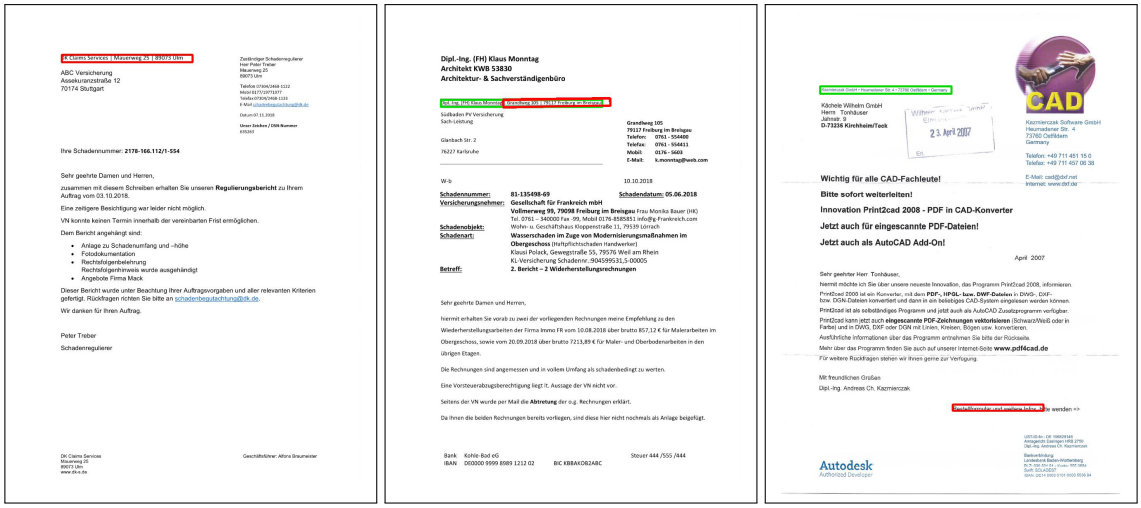


Abbildung 5.3.: Beispiele für die verschiedenen Klassifizierungen bei der Evaluation in Tabelle 5.1. Grün markiert ist die Adresszeile, rot die maschinelle Klassifizierung. Von links nach rechts: Ein Dokument wo die Absenderadresse „erkannt“ wurde, ein Dokument, wo sie „teilweise erkannt“ wurde und ein Dokument, wo sie „nicht erkannt“ wurde. Als „teilweise erkannt“ werden Dokumente gewertet, wo nicht die ganze Adresszeile erkannt wurde, also einzelne Zeichen oder gar Wörter nicht in der markierten Region liegen.

nicht wesentlich mehr. „Teilweise erkannt“ sind Ergebnisse, in denen Teile der Absenderadresse abgeschnitten sind oder zusätzlich leere Flächen in der Markierung inbegriffen sind. Zusammen mit den als „erkannt“ gewerteten Dokumenten sind diese Dokumente als Treffer zu werten, da die Region richtig klassifiziert wurde und oft über ein simples, nachträgliches Erweitern der Region in horizontaler Richtung der Rest der Absenderadresse mit erfasst wird.

Wenn die Intelligenz Regionen markiert hat, welche nicht die Absenderadresse sind, aber auch keine andere Form einer Adresse (insbesondere die Empfängeradresse) wird das als „nicht erkannt“ gewertet. Diese Ergebnisse sind zwar nicht hilfreich, aber können leicht von Hand aussortiert werden, da sie leicht als falsch erkannt werden.

Die problematischste und damit unerwünschteste Kategorie Ergebnisse wird als „falsch erkannt“ klassifiziert. In diesem Fall wurde eine Adresse oder eine Region innerhalb einer Adresse erkannt, welche nicht die Absenderadresse ist. Das macht den Fehler schwer zu erkennen, da es sich um eine echte Adresse handelt. Es sind viele Szenarien denkbar, in denen ein solches fehlerhaftes Ergebnis unbemerkt in eine Datenbank wandert, und erst bei einem konkreten Vorfall entdeckt wird. Das gilt es so gut wie möglich zu vermeiden.

Glücklicherweise treten Fehler dieser Art selbst in der Baseline, welche auf sehr wenigen Daten trainiert wurde, nicht auf. Das bedeutet, dass der Einsatz von künstlicher Intelligenz in diesem Gebiet nicht ausgeschlossen werden muss. Würden hingegen Adressen „falsch erkannt“ werden, würde das diesen Ansatz disqualifizieren.

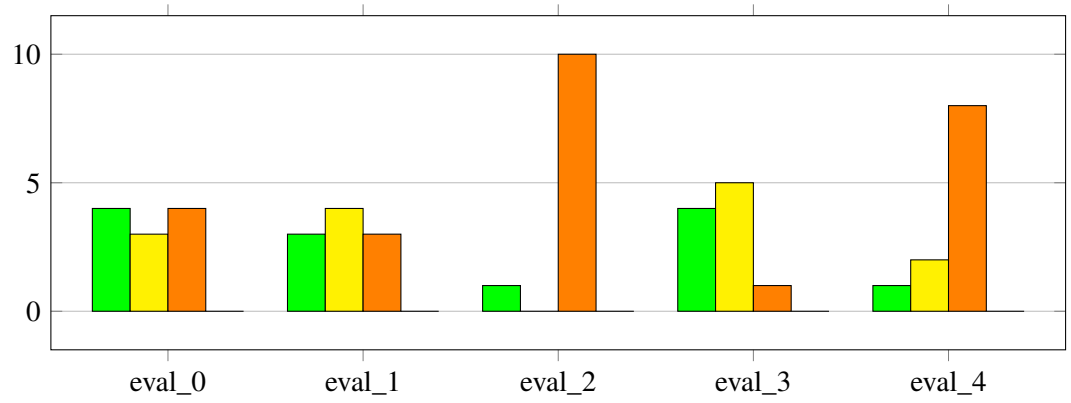


Abbildung 5.4.: Die Ergebnisse der Kreuzvalidierung der Baseline. Der Durchschnitt dieser Werte wird in Tabelle 5.1 genutzt, um die Qualität der Ergebnisse von Ansatz 2 zu beschreiben.

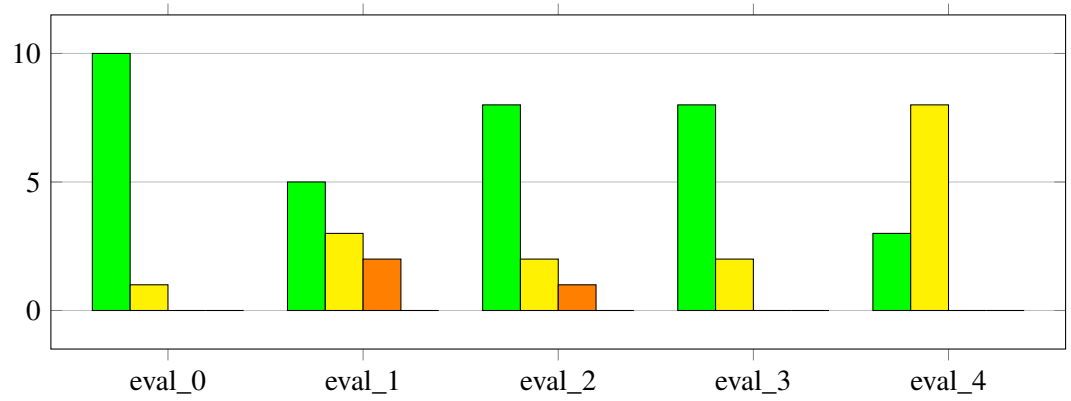


Abbildung 5.5.: Die Ergebnisse der Kreuzvalidierung von Ansatz 2. Der Durchschnitt dieser Werte wird in Tabelle 5.1 genutzt, um die Qualität der Ergebnisse der Baseline zu beschreiben.

Bei den Ergebnissen der Kreuzvalidierung von dem an synthetischen und echten Daten trainierten Ansatz 2 in Abbildung 5.5 fällt auf, dass die einzelnen Modelle in ihrer Qualität stark voneinander abweichen. Dies ist ein Zeichen dafür, dass die wenigen echten Trainingsdaten einen großen Einfluss auf das Modell haben, und manche wichtiger sind für seine Qualität als andere.

Ein ähnliches Bild bieten die Ergebnisse der Kreuzvalidierung der Baseline 5.4. Auch dort besteht eine hohe Variation zwischen den einzelnen Modellen. Da die Baseline an den gleichen Teilmengen von echten Daten trainiert wurde wie Ansatz 2 lassen sich die Werte in Abbildung 5.5 und Abbildung 5.4 direkt miteinander vergleichen. Es wird deutlich, wie groß der Einfluss der synthetischen Daten ist, und wie sehr die Ergebnisse in Ansatz 2 durch sie verbessert wurden.

	Erkannt	Teilweise erkannt	Nicht erkannt	Falsch erkannt	Treffer
Baseline (echt)	24,91 %	27,09 %	48,00 %	0 %	52,00 %
Ansatz 1 (synthetisch)	18,90 %	32,80 %	48,30 %	0 %	51,70 %
Ansatz 2 (kombiniert)	64,72 %	31,45 %	5,83 %	0 %	96,17 %

Tabelle 5.1.: Die Ergebnisse der Modelle mit ausschließlich echten Daten („Baseline (echt)“), ausschließlich synthetischen Daten („Ansatz 1 (synthetisch)“) und der Kombination von echten und synthetischen Daten („Ansatz 2 (kombiniert)“). In der Spalte „Treffer“ werden erkannte und teilweise erkannte Dokumente zusammengezählt.

5.3. Auswertung der Tests

Beim Blick auf die Ergebnisse in Tabelle 5.1 fällt direkt auf, dass Ansatz 2 deutlich bessere Ergebnisse liefert als die Baseline mit echten Dokumenten. Wenn man die Ergebnisse der Baseline in Abbildung 5.6 anschaut, erkennt man dort typische Symptome für ein Modell, welches an zu wenigen Trainingsdaten trainiert wurde. Es wurden Textblöcke markiert, die oftmals den Absenderdaten nicht ähneln, was andeutet, dass die Intelligenz noch kein genaues Bild der Merkmale erstellen konnte. Dass dennoch 24,91 % der Tests erkannt wurden, kann daran liegen, dass diese Dokumente starke Ähnlichkeiten mit Dokumenten im Trainingskorpus zeigten. Ein weiteres Problem, welches vor allem bei den Teilmengen `eval_2` und `eval_4` auftritt (Siehe Abbildung 5.4), ist eine zu hohe Konfidenz in anderen Regionen. Dabei wurde in der Wahrscheinlichkeitskarte zwar eine Region als wahrscheinlich markiert, da aber die anderen Regionen im Dokument mit einer ebenfalls relativ hohen Konfidenz markiert wurden, wurden sie vom Schwellwert nicht eliminiert. Das führt dazu, dass sämtliche Regionen dann von der künstlichen Intelligenz markiert werden, was das Ergebnis wertlos macht (Siehe Abbildung 5.6 links).

Auch Ansatz 1, welcher nur mit synthetischen Daten trainiert wurde, liefert keine guten Ergebnisse. Mit einer Trefferquote von 51,7 % werden nur knapp über die Hälfte aller Adressen richtig erkannt. Dieser Wert ist der Baseline sehr nahe und zeigt deutlich, dass die synthetischen Daten nicht alle möglichen Fälle ausreichend abdecken können. Es ist auffällig, dass in diesem, mit ausschließlich synthetischen Daten trainierten Modell weniger „perfekt“ erkannte Adressen auftreten als in der Baseline, dafür aber viele, die teilweise erkannt wurden.

Hier könnte ein Effekt eingetreten sein, bei dem sich die künstliche Intelligenz an Eigenschaften orientiert, die ausschließlich in den synthetischen Daten auftreten, und sich nicht auf echte Briefe verallgemeinern lassen. Demnach wird nicht die komplette Absenderadresse erkannt, sondern nur der Teil, dessen Eigenschaften mit den erlernten übereinstimmt (Siehe Abbildung 5.7). Um diesem Effekt, der auch „overfitting“ genannt wird, entgegenzuwirken, könnte man die synthetischen Daten nachträglich an echte Daten angleichen, wobei die Annotation erhalten bliebe. In Abschnitt 2.1 wird ein Vorgehen von Shrivastava et al. beschrieben, wie mit mehreren neuronalen Netzen realistischere Daten erzeugt werden können, die nicht mehr über diese synthetischen-typischen Eigenschaften verfügen.

Ansatz 2, welcher sowohl mit den echten als auch mit den synthetischen Daten trainiert wurde, liefert Ergebnisse, die besser sind als die der ausschließlich synthetischen oder ausschließlich echten Daten zusammen. Mit einer Trefferquote von 91,13 % werden so am meisten Absenderadressen

Abbildung 5.6.: Beispiele für Dokumente, welche von der Baseline nicht erkannt wurden.

erkannt, und an Beispielen für nicht erkannte Dokumente wird klar, dass diese relativ stark von den beobachteten Layouts abweichen (Siehe Abbildung 5.8). Große Farbflächen und Grafiken können von dem Modell nicht eingeordnet werden. Es könnte helfen, diese Fälle auch in den synthetischen Daten zu repräsentieren und so eine noch größere Abdeckung zu erreichen.

Über die Ursachen von Ergebnissen künstlicher Intelligenzen zu spekulieren ist jedoch sehr schwierig, und keine Erklärung kann als richtig oder falsch bewiesen werden. Die Erklärbarkeit von künstlicher Intelligenz eröffnet insofern ein komplett neues Forschungsfeld, siehe Abschnitt 6.3.

Die eingangs erwähnten Probleme (Siehe Tabelle 1.1) werden in Ansatz 1 vollständig gelöst. Ein komplett synthetischer Trainingsdatensatz sollte keine Verunreinigungen aufweisen, welche nicht vom Programmierer gewünscht sind. Dass große Datenmengen nötig sind ist kein Problem, da sich die Trainingsdaten mit geringem Aufwand generieren lassen. Auch das große Problem des Datenschutzes spielt keine Rolle, da keine echten Daten verwendet werden, und sämtliche potentiell personenbezogene Daten frei generiert werden. Für sich allein genommen sind die Ergebnisse von geringem Wert, erst bei dem Einsatz in Ansatz 2 wird der Vorteil durch synthetische Daten deutlich.

In Ansatz 2 wird ein kleiner Teil echter Daten verwendet. Wie alle echten Daten können diese fehlerhaft oder vorurteilsbelastet sein. Ihr Einfluss auf die Entscheidungen der künstlichen Intelligenz sollte aber nicht so groß sein, um dem Modell eine Verzerrung beizubringen. Sollten die Daten jedoch starke Tendenzen aufweisen, können eventuell jedoch moderierende Maßnahmen getroffen werden. Da keine großen Datenmengen genutzt werden, ist ihre Beschaffung kein unüberwindbares Problem. In dem Fall, dass eine Annotation von Hand nötig ist, ist diese von überschaubarem

Abbildung 5.7.: Beispiele für Dokumente, welche von Ansatz 1 (synthetisch) teilweise erkannt wurden.

Aufwand, da schon wenige Dokumente die Qualität des Modells stark verbessern. Datenschutz hingegen ist in diesem Ansatz ein gewisses Hindernis. Nicht jedes Szenario lässt es zu, dass echte Dokumente zum Training genutzt werden. Dennoch sind kleine Mengen echter Daten in der Regel um ein vielfaches leichter zu erhalten als riesige Korpora.

Insofern überwindet dieser Ansatz viele Hindernisse, die sich bei der Arbeit und dem Training problemspezifischer künstlicher Intelligenzen ergeben.

Abbildung 5.8.: Beispiele für Dokumente, welche von Ansatz 2 (kombiniert) nicht erkannt wurden.

6. Zusammenfassung und Ausblick

Dieses Kapitel fasst die Arbeit zusammen und gibt einen Ausblick auf zukünftige Arbeiten, die auf ihrer Grundlage bearbeitet werden könnten.

Das Ziel der Arbeit war es festzustellen, ob synthetische Daten einen Vorteil bieten, wenn nur kleine Mengen echter Trainingsdaten zur Verfügung stehen, und das am Beispiel der Absenderdatenerkennung an Briefkorrespondenz. In diesem Rahmen wurden zu Beginn Korrespondenzbriefe ausgewertet. Eine Analyse ihrer Struktur im Hinblick auf Absenderdaten enthüllte ein typisches Muster, nach dem die Briefe strukturiert sind. Daraus konnten drei typische Layouts abgeleitet werden, welche Variationen im Aufbau der Briefe berücksichtigen. Jedes Layout verfügt über eine Vielzahl zufälliger Faktoren, welche bei jeder Anwendung so gewählt wurden, dass möglichst viele Varianten des Layouts abgedeckt wurden.

Um diese Layouts mit passenden Beispieldaten zu füllen, wurde ein Datenobjekt generiert, in dem Absender, Empfänger und Sachbearbeiter hinterlegt sind. Diese Daten sind entweder aus öffentlich zugänglichen Korpora zusammengestellt, oder nach einem Zufallsprinzip generiert. Es wurde darauf geachtet, möglichst viele Sonderfälle in der Schreibweise zu berücksichtigen, um ein realistisches Schriftbild zu gewährleisten.

Diese Daten wurden dann in das Layoutobjekt übernommen, und gegebenenfalls in einem Zwischenverarbeitungsschritt noch auf den Brief angepasst. Der fertige Brief wurde dann als PDF-, sowie als Portable Network Graphics (PNG)-Datei abgespeichert.

Mit jedem Brief wurde auch eine Labeldatei generiert, ebenfalls als PDF- und PNG-Datei, welche an jeder Stelle im Bild die Art der dort vorhandenen Region farblich codiert. Das Datenpaar Briefdatei und Labeldatei kann dann zum Trainieren einer künstlichen Intelligenz verwendet werden.

Neben dem Label wurde zu jedem generierten Brief auch eine PageXML-Datei erstellt, welche in zukünftigen Arbeiten als „Truth“ verwendet werden kann.

Tests an den durch synthetische Daten trainierten Modellen bewahrheiteten die Hypothese, dass synthetische Daten in Fällen, wo wenige echte Trainingsdaten vorhanden sind, einen Vorteil bieten. Die entscheidenden Informationen über Dokumente konnten der künstlichen Intelligenz beigebracht werden, und in der Kombination mit den wenigen vorhandenen echten Daten wurden in Tests 96,17 % der Daten erkannt oder teilweise erkannt. Im Gegensatz dazu erzielte die Baseline, ohne die synthetischen Daten, gerade einmal 52 % erkannte oder teilweise erkannte Absenderadressen. Daran wird der Nutzen vom Einsatz von synthetischen Daten beim Training in Fällen mit wenigen echten Trainingsdaten deutlich erkennbar.

Im Folgenden werden Problemstellungen beschrieben, die in zukünftigen Arbeiten bearbeitet werden können und nicht im Rahmen dieser Arbeit gelöst werden konnten.

6.1. Feingranulare Labels für Textmustererkennung

Im Zuge dieser Arbeit wurden auch feingranularere Labeldateien generiert, welche die Arbeit mit Techniken ermöglicht, die präzisere Auswertungen zulassen. Ein dafür infrage kommendes Tool ist OCRD-Segment[1] welches für den speziellen Anwendungsfall der Textregionenerkennung entwickelt wurde.

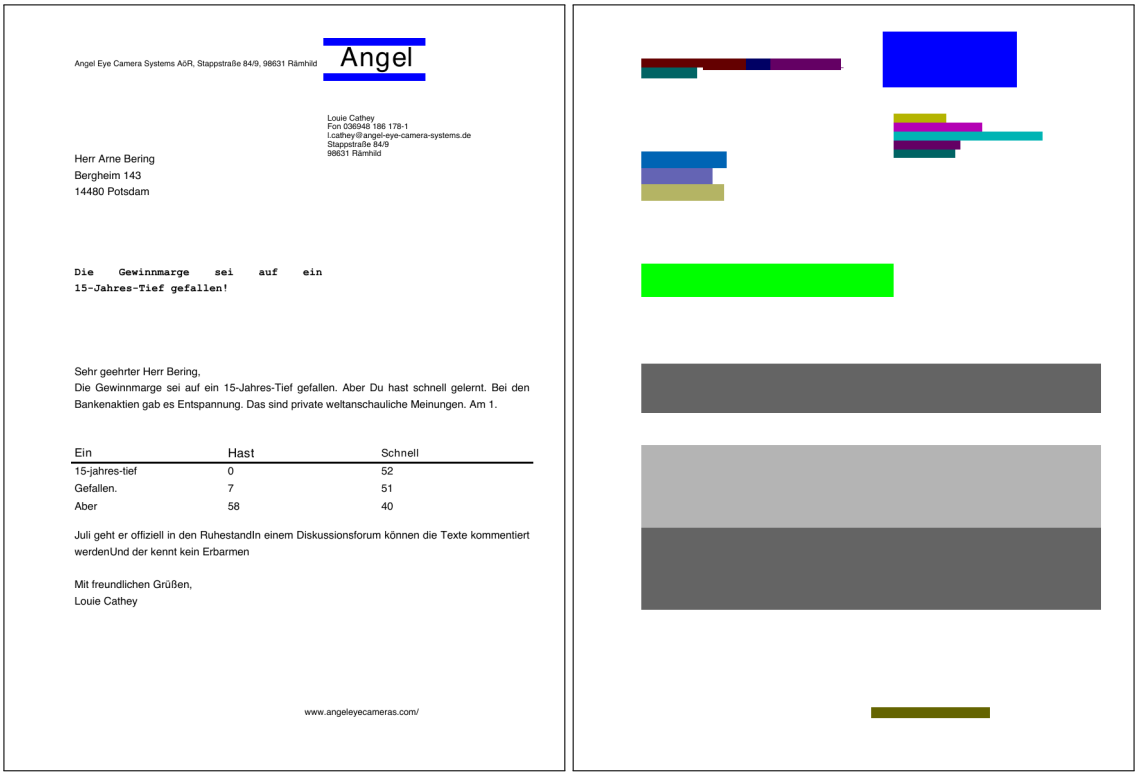


Abbildung 6.1.: Rechts eine für OCRD-Segment verwendbare Labeldatei, in welcher jede Datenart unterschiedlich eingefärbt ist. Links das synthetische Dokument, wie auch in Abbildung 5.1.

Der Vorteil dieser feingranularen Annotation ist, dass eine künstliche Intelligenz genauer die graphischen Eigenschaften der in den Regionen vorhandenen Zeichen erlernen kann. Typographische Eigenschaften zu erlernen, wie z.B. dass Postleitzahlen stets aus fünf Ziffern bestehen, oder Telefonnummern stets ähnlich aufgebaut sind, gibt einer KI zusätzliche Informationen. Mit diesen, sowie mit den existierenden Informationen über das Dokument können dann bei der Segmentierung mehr Eigenschaften berücksichtigt werden, was bessere Ergebnisse ermöglicht.

[1] https://github.com/OCR-D

6.2. Post-Processing mit regelbasiertem Ansatz

Ein vielversprechender Ansatz zur Optimierung der Ergebnisse ist eine Kombination des hier präsentierten Machine-Learning Ansatzes mit einem regelbasierten Ansatz. Dafür könnte man nicht nur die Region betrachten, die von der KI als wahrscheinlichster Treffer eingeordnet wurde, sondern auch sämtliche weitere Regionen, denen eine Wahrscheinlichkeit zugeordnet wurde. Wenn man aus diesen Regionen nun den Text extrahiert und diesen über einen regelbasierten Ansatz darauf untersucht, ob es sich um eine gültige Adresse handelt, kann man sämtliche Regionen, die laut Tabelle 5.1 als „nicht erkannt“ klassifiziert wurden, als unerwünscht identifizieren. Hätte die KI die tatsächliche Absenderadresse erkannt, aber mit einer geringeren Wahrscheinlichkeit bewertet, könnte man sie auf diesem Weg dennoch extrahieren. Da dieser Ansatz nur Ergebnisse beeinflusst, bei denen die Adresse nicht direkt erkannt wurde, kann er die Erkennungsquote nur erhöhen und hat keinen negativen Einfluss auf die Qualität der Ergebnisse.

Ein zu lösendes Problem kann sein, dass in manchen Fällen auch die zu meidende Empfängeradresse als Region von der KI markiert wird. Ein regelbasierter Ansatz könnte diese nicht von der Absenderadresse unterscheiden und es würde in diesem Fall eine Adresse „falsch erkannt“.

Da, wie in Tabelle 5.1 gezeigt, die Empfängeradresse in keinem Testdokument die höchste Konfidenz erhielt, ist dieses Problem bei dem reinen Machine-Learning-Ansatz nicht vorhanden. Dieses Ergebnis kann aber nicht auf Fälle, in denen eben auch Regionen mit niedrigeren Konfidenzen betrachtet werden, verallgemeinert werden.

6.3. Erklärbare KI

Wie bereits erwähnt, entstehen beim Versuch, die Ergebnisse einer künstlichen Intelligenz zu erklären, Schwierigkeiten. Es ist nicht ersichtlich, auf welche Beispiele eine konkrete Entscheidung gestützt wird. Zudem lässt sich nicht erkennen, welche Eigenschaften die KI nutzt um die Entscheidungen zu treffen. Das ist ein generelles Problem bei der Arbeit mit Machine Learning, und somit von großer Relevanz.

Die Modelle, welche eine KI beim Training erstellt, verfügen in aller Regel über keine vom Menschen verstehbaren Parameter. Es kann also nicht im Voraus über eine künstliche Intelligenz gesagt werden, aufgrund welcher Eigenschaften sie ihre Entscheidungen fällt. Dementsprechend werden KIs häufig als „Black Box“ behandelt. Es werden Daten eingespeist, und ein Ergebnis ausgegeben, die Arbeitsweise ist aber nicht erkennbar.

Genauso wie sich die Ergebnisfindung einer künstlichen Intelligenz nicht nachvollziehen lässt, kann man ihre Eignung für Sonderfälle nicht im Voraus feststellen. Das birgt das zusätzliche Problem, dass die Qualität einer KI nicht ohne Beispiele festgestellt werden kann, und unter Umständen erst im Betrieb auffällt, dass sie für gewisse Fälle ungeeignet ist.

Um sich dennoch auf die Ergebnisse einer künstlichen Intelligenz verlassen zu können, ist es also von zentraler Bedeutung, ihre Entscheidungsfindung nachvollziehbar zu machen. In vielen der in Kapitel 1 erwähnten Anwendungen für künstliche Intelligenz, wie das Analysieren von Röntgenaufnahmen [WS12] oder das Steuern von Autos [Sti18], gewinnen erst dann Vertrauen,

wenn auch ihre Entscheidungen von Menschen evaluiert werden können, nicht nur die Ergebnisse. Gerade im Falle von Fehlern ist eine Diagnose von großer Bedeutung, um diese mit Sicherheit beseitigen zu können.

Die Frage, wie nachvollziehbare KI aussehen kann, und wie das Vertrauen in ihre Ergebnise begründet werden kann, ist ein aktuelles Forschungsthema ([DSB17] [HBPK17]) und bietet noch viel Raum für weitere Arbeiten.

Fazit

Synthetische Daten können den eingangs erwähnten Problemen (Siehe Tabelle 1.1) Abhilfe verschaffen. Eine Kombination mit wenigen echten Daten steigert die Qualität der künstlichen Intelligenz erheblich, und bietet in Situationen, wo diese zur Verfügung stehen, deutlich bessere Ergebnisse als rein synthetische Trainingsdaten. Doch selbst ohne echte Trainingsdaten liefert ein rein synthetisch trainiertes Modell moderate Ergebnisse, die mit geeigneten Post-Processing-Strategien genutzt werden können. Zu beachten ist, dass die synthetischen Daten so nah wie möglich an den realen liegen sollten. Je besser die Abdeckung der Spezialfälle, desto besser die Leistung der künstlichen Intelligenz.

Literaturverzeichnis

[CGS18] Y. S. Chernyshova, A. V. Gayer, A. V. Sheshkus. „Generation method of synthetic training data for mobile OCR system“. In: *Tenth International Conference on Machine Vision (ICMV 2017)*. Hrsg. von A. Verikas, P. Radeva, D. Nikolaev, J. Zhou. Bd. 10696. International Society for Optics und Photonics. SPIE, 2018, S. 640–646. DOI: 10.1117/12.2310119. URL: https://doi.org/10.1117/12.2310119 (zitiert auf S. 21).

[DSB17] D. Doran, S. Schulz, T. R. Besold. „What Does Explainable AI Really Mean? A New Conceptualization of Perspectives“. In: *CoRR* abs/1710.00794 (2017). arXiv: 1710.00794. URL: http://arxiv.org/abs/1710.00794 (zitiert auf S. 50).

[GEQ12] D. Goldhahn, T. Eckart, U. Quasthoff. „Building Large Monolingual Dictionaries at the Leipzig Corpora Collection: From 100 to 200 Languages.“ In: *LREC*. Bd. 29. 2012, S. 31–43 (zitiert auf S. 32).

[GSJZ18] N. Garg, L. Schiebinger, D. Jurafsky, J. Zou. „Word embeddings quantify 100 years of gender and ethnic stereotypes“. In: *Proceedings of the National Academy of Sciences* 115.16 (2018), E3635–E3644 (zitiert auf S. 16).

[HBPK17] A. Holzinger, C. Biemann, C. S. Pattichis, D. B. Kell. „What do we need to build explainable AI systems for the medical domain?“ In: *CoRR* abs/1712.09923 (2017). arXiv: 1712.09923. URL: http://arxiv.org/abs/1712.09923 (zitiert auf S. 50).

[HZRS15] K. He, X. Zhang, S. Ren, J. Sun. „Deep Residual Learning for Image Recognition“. In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385 (zitiert auf S. 37).

[KKR+16] A. Kannan, K. Kurach, S. Ravi, T. Kaufmann, A. Tomkins, B. Miklos, G. Corrado, L. Lukacs, M. Ganea, P. Young et al. „Smart reply: Automated response suggestion for email“. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2016, S. 955–964 (zitiert auf S. 15).

[OSK18] S. A. Oliveira, B. Seguin, F. Kaplan. „dhSegment: A generic deep-learning approach for document segmentation“. In: *2018 16th International Conference on Frontiers in Handwriting Recognition (ICFHR)*. IEEE. 2018, S. 7–12 (zitiert auf S. 22, 37).

[PA10] S. Pletschacher, A. Antonacopoulos. „The PAGE (page analysis and ground-truth elements) format framework“. In: *2010 20th International Conference on Pattern Recognition*. IEEE. 2010, S. 257–260 (zitiert auf S. 29).

[RFB15] O. Ronneberger, P. Fischer, T. Brox. „U-Net: Convolutional Networks for Biomedical Image Segmentation“. In: *CoRR* abs/1505.04597 (2015). arXiv: 1505.04597. URL: http://arxiv.org/abs/1505.04597 (zitiert auf S. 37).

[RSC19] R. Richardson, J. Schultz, K. Crawford. „Dirty Data, Bad Predictions: How Civil Rights Violations Impact Police Data, Predictive Policing Systems, and Justice“. In: *N.Y.U. L. REV. ONLINE*. Bd. 94. 2019, S. 192–233. URL: https://ssrn.com/abstract=3333423 (zitiert auf S. 16).

[SPT+16] A. Shrivastava, T. Pfister, O. Tuzel, J. Susskind, W. Wang, R. Webb. „Learning from Simulated and Unsupervised Images through Adversarial Training“. In: *CoRR* abs/1612.07828 (2016). arXiv: 1612.07828. URL: http://arxiv.org/abs/1612.07828 (zitiert auf S. 17, 22).

[Sti18] J. Stilgoe. „Machine learning, social learning and the governance of self-driving cars“. In: *Social Studies of Science* 48.1 (2018). PMID: 29160165, S. 25–56. DOI: 10.1177/0306312717741687. eprint: https://doi.org/10.1177/0306312717741687. URL: https://doi.org/10.1177/0306312717741687 (zitiert auf S. 15, 49).

[VB03] T. Varga, H. Bunke. „Generation of synthetic training data for an HMM-based handwriting recognition system“. In: *Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings.* 2003, 618–622 vol.1 (zitiert auf S. 21).

[WS12] S. Wang, R. M. Summers. „Machine learning and radiology“. In: *Medical Image Analysis* 16.5 (2012), S. 933–951. ISSN: 1361-8415. DOI: https://doi.org/10.1016/j.media.2012.02.005. URL: http://www.sciencedirect.com/science/article/pii/S1361841512000333 (zitiert auf S. 15, 49).

[ZEP07] O. Zaidan, J. Eisner, C. Piatko. „Using “Annotator Rationales” to Improve Machine Learning for Text Categorization“. In: *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*. Rochester, New York: Association for Computational Linguistics, Apr. 2007, S. 260–267. URL: https://www.aclweb.org/anthology/N07-1033 (zitiert auf S. 17).

Alle URLs wurden zuletzt am 18. 05. 2020 geprüft.

A. Programmcode

A.1. createPDF.py

```
from fpdf import FPDF     # Dieses Package ermöglicht das Erstellen von pdf-Dateien
import xmlschema
import applyLayout
import random
import createLogo as cl
import createXML as cxml
from datetime import datetime
dt = datetime.now().__str__().replace(" ", "T")

def newPDF():
  # Erstelle ein neues, leeres pdf-Dokument
    pdf = FPDF('P', 'mm', 'A4')
    pdf.add_page()
    pdf.set_margins(0, 0.1, 0)
    pdf.set_auto_page_break(False, 0.0)
    pdf.set_font('Arial', '', 11)
    return pdf

def addToPage(pdf, elementName, elementType, fontSize, upperLeft, lowerLeft, lowerRight,
upperRight, content, counter, label):
  # Füge dem pdf-Dokument ein Objekt hinzu
    setColor(pdf, elementType, label)
    pdf.set_font('Arial', '', int(fontSize))
    pdf.set_xy(upperLeft[0], upperLeft[1])
    if elementType == 'logo':
        cl.addLogo(pdf, elementName, elementType, upperLeft, lowerLeft, lowerRight, upperRight
, content, label, colors['logo'])
    elif elementType == 'tableBody':
        drawTable(pdf, upperLeft[0], upperLeft[1], upperRight[0] - upperLeft[0], lowerLeft[1]
- upperLeft[1], content, int(fontSize), label)
    elif isinstance(content, list):
        dealWithLists(pdf, elementName, elementType, fontSize, upperLeft, lowerLeft,
lowerRight, upperRight, content, counter, label)
    else:
        dealWithBlocks(pdf, elementName, elementType, fontSize, upperLeft, lowerLeft,
lowerRight, upperRight, content, counter, label)

def dealWithBlocks(pdf, elementName, elementType, fontSize, upperLeft, lowerLeft, lowerRight,
upperRight, content, counter, label):
    if elementType == 'subjectLine':
        pdf.set_font('Courier', 'B', int(fontSize))
```

```
    pdf.multi_cell(upperRight[0] - upperLeft[0], int(fontSize) - 5.5, content, 0, 'J', label)

def setColor(pdf, elementType, label=False, textColor=[0, 0, 0], drawColor=[0, 0, 0],
fillColor=[0,0,0]):
  # Setze die Text-Farbe, Linien-Farbe und Füll-Farbe
    tempTextColor = textColor
    tempDrawColor = drawColor
    tempFillColor = fillColor
    for color in colors:
        if color == elementType:
            tempFillColor = colors[elementType]
            if label:
                tempTextColor = colors[elementType]
                tempDrawColor = colors[elementType]
    pdf.set_text_color(tempTextColor[0], tempTextColor[1], tempTextColor[2])
    pdf.set_draw_color(tempDrawColor[0], tempDrawColor[1], tempDrawColor[2])
    pdf.set_fill_color(tempFillColor[0], tempFillColor[1], tempFillColor[2])

def dealWithLists(pdf, elementName, elementType, fontSize, upperLeft, lowerLeft, lowerRight,
upperRight, content, counter, label):
  # Stelle Daten dar, die als Liste hinterlegt sind
    pdf.set_xy(upperLeft[0], upperLeft[1])
    index = 0
    if elementType == 'footer':
        width = (upperRight[0] - upperLeft[0]) / 2
        for line in content:
            if len(line[0]) > 1:
                if annotationType == 1:
                    setColor(pdf, line[1], label)
                if index == round(len(content) / 2):
                    pdf.set_xy(upperLeft[0] + width, upperLeft[1])
                pdf.cell(pdf.get_string_width(line[0]) + 2, int(fontSize) - 5.5, line[0], 0,
2, 'L', line[1] and label)
                index += 1
    elif elementType == 'addressBlock' or elementType == 'receiverAddress':
        width = upperRight[0] - upperLeft[0]
        for line in content:
            if annotationType == 1:
                setColor(pdf, line[1], label)
            if (pdf.get_x() + pdf.get_string_width(line[0])) > upperRight[0]:
                pdf.set_xy(upperLeft[0], pdf.get_y() + int(fontSize) - 5.5)
            pdf.cell(pdf.get_string_width(line[0]) + 2, int(fontSize) - 5.5, line[0], 0, 2, 'L
', label)
            index += 1
    elif elementType == 'senderAddress':
        width = upperRight[0] - upperLeft[0]
        if random.randrange(0, 6) == 0: pdf.set_font('Arial', 'U', int(fontSize))
        for line in content:
            temp = line[0]
            if isinstance(line, list):
                if index == 1 or index == 2:
                    line[0] =  temp + ', '
                elif index != len(content) - 1:
```

```
                line[0] = temp + ' '
            if (pdf.get_x() + pdf.get_string_width(line[0])) > upperRight[0]:
                setColor(pdf, '', label, [255,255, 255], [255,255,255], [255,255,255])
                pdf.cell(pdf.get_string_width('A'), int(fontSize) - 5.5, '', 0, 0, 'L', True)
                pdf.set_xy(upperLeft[0], pdf.get_y() + int(fontSize) - 5.5)
            if annotationType == 1:
                setColor(pdf, line[1], label)
            pdf.cell(pdf.get_string_width(line[0]), int(fontSize) - 4.5, line[0], 0, 0, 'L',
label)
            if index == len(content) - 1:
                setColor(pdf, '', label, [255,255, 255], [255,255,255], [255,255,255])
                pdf.cell(pdf.get_string_width('A'), int(fontSize) - 4.5, '', 0, 0, 'L', True)
            index += 1

def drawTable(pdf, x, y, width, height, content, fontSize, label):
  # Zeichne eine Tabelle
    if label:
        pdf.rect(x, y, width, height, 'F')
    else:
        columns = random.randrange(3, 6)
        activeColumn = 0
        columnWidth = width / columns
        # Generiere Daten
        index = 0
        table = []
        for i in range(0, columns):
            column = []
            for j in range(0, round(height / (fontSize - 5))):
                if j == 0 or i == 0:
                    words = content.split(' ')
                    column.append(words[(index + 7) % len(words)].capitalize())
                    index = index + 1
                else:
                    column.append(str(random.randrange(0, 100)))
            table.append(column)

        # Füge Daten in pdf ein
        fonts = ['Arial', 'Times', 'Courier']
        font = fonts[random.randrange(0, len(fonts))]
        pdf.set_font(font, '', fontSize)

        for column in table:
            x_pos = x + activeColumn * columnWidth
            pdf.set_xy(x_pos, y)
            activeLine = 0
            rand = random.randrange(0, 4)
            for line in column:
                if activeLine == 0:
                    pdf.set_font(font, 'B', fontSize + random.randrange(0, 3))
                addToParagraph(pdf, columnWidth, fontSize - 5, line, 'L', False)
                pdf.set_font(font, '', fontSize)
                if activeLine == 0 or rand == 0:
                    pdf.set_line_width(random.uniform(0.2, 1))
```

```
                    pdf.line(x, pdf.get_y(), x + width, pdf.get_y())
                activeLine = activeLine + 1
                if pdf.get_y() >= y + height or activeLine == len(column) - 1:
                    break
            activeColumn = activeColumn + 1

def setParagraphStyle(pdf, x, y, font, fontSize, fontDecoration, textColor, labelColor, label)
:
    pdf.set_font(font, fontDecoration, int(fontSize))
    if label:
        pdf.set_text_color(255, 255, 255)
    else:
        pdf.set_text_color(textColor[0], textColor[1], textColor[2])

def addToParagraph(pdf, width, height, content, align, label):
    pdf.cell(width, height, content, 0, 2, align, label)

def exportPDF(pdf, counter, path):
  # Speichere die pdf unter dem angegebenen Pfad
    try:
        pdf.output('../' + path + '/letter_' + str(counter) + '.pdf', 'F')
        return path + '/letter_' + str(counter) + '.pdf'
    except UnicodeEncodeError:
        pass

def createPDF(data, counter, path):
  # Diese Methode fasst alle Schritte, die zum Erstellen eines Briefs mit Annotation nötig
sind, zusammen
    global annotationType
    annotationType = 1

    # generiere Layout-Daten
    rand = random.randrange(0, 3)
    if rand == 0:
        regions = applyLayout.getLayoutOne(data)
    elif rand == 1:
        regions = applyLayout.getLayoutTwo(data)
    else:
        regions = applyLayout.getLayoutThree(data)

    # generiere einen Brief
    pdf = newPDF()
    id_counter = 0
    for x in regions:
        addToPage(pdf, x.elementName, x.elementType, x.fontSize, x.upperLeft, x.lowerLeft, x.
lowerRight, x.upperRight, x.content, id_counter, False)
        id_counter = id_counter + 1
    result1 = exportPDF(pdf, counter, path)

    # generiere grobe Annotation für den Brief
    pdf = newPDF()
    id_counter = 0
    for x in regions:
```

```
        addToPage(pdf, x.elementName, x.elementType, x.fontSize, x.upperLeft, x.lowerLeft, x.
lowerRight, x.upperRight, x.content, id_counter, True)
        id_counter = id_counter + 1
    result2 = exportPDF(pdf, counter, 'GeneratedLetterLabels')

  # generiere feingranulare Annotation für den Brief
    annotationType = 2
    pdf = newPDF()
    id_counter = 0
    for x in regions:
        addToPage(pdf, x.elementName, x.elementType, x.fontSize, x.upperLeft, x.lowerLeft, x.
lowerRight, x.upperRight, x.content, id_counter, True)
        id_counter = id_counter + 1
    result3 = exportPDF(pdf, counter, 'GeneratedLetterLabelsB')

  # erstelle eine xml-Datei zu dem Brief
    cxml.createXML(regions, counter, 'GeneratedXML', result1)

    return [result1 ,result2, result3]
```

Listing A.1: Die createPDF.py-Datei

A.2. createData.py

```
import accessData as data   # In dieser Klasse wird auf Daten aus Listen zugegriffen
import random

class LetterData:
    def __init__(self):
        self.receiver = Receiver()
        self.company = Company()
        self.clerk = Clerk(self.company)
        self.date = data.getRandomDate()
        self.text = generateText(random.randrange(60, 120))

    def __str__(self):
        result = str(self.receiver) + '\n' + str(self.company) + '\n' + str(self.clerk)
        return result

class Receiver:
    def __init__(self):
        firstName = data.getRandomFirstName()
        zipAndCity = data.getRandomZipAndCity()
        self.gender = firstName[1]
        self.firstName = firstName[0]
        self.lastName = data.getRandomLastName()
        self.name = self.firstName + ' ' + self.lastName
        self.street = data.getRandomStreet()
        self.streetNumber = data.getRandomStreetNumber()
        self.zip = zipAndCity[1]
        self.city = zipAndCity[0]
        self.country = 'Deutschland'
```

```
    def __str__(self):
        return f"Empfaenger: {self.firstName} {self.lastName} ({self.gender}), {self.street} {
self.streetNumber}, {self.zip} {self.city}, {self.country}"

class Company:
    def __init__(self):
        zipAndCity = data.getRandomZipAndCity()
        nameAndWeb = data.getRandomCompanyName()
        phoneAndFax = data.generatePhoneAndFax(zipAndCity[2])
        self.street = data.getRandomStreet()
        self.streetNumber = data.getRandomStreetNumber()
        self.name = nameAndWeb[0]
        self.legalForm = generateLegalForm()
        self.zip = zipAndCity[1]
        self.state = zipAndCity[3]
        self.areaCode = zipAndCity[2]
        self.city = zipAndCity[0]
        self.country = 'Deutschland'
        self.postbox = data.generatePostbox()
        self.eMail = generateCompanyEMail(self.name)
        self.fax = phoneAndFax[1]
        self.phone = phoneAndFax[0]
        self.website = nameAndWeb[1]
        self.openingHours = ''
        self.bankInfo = data.getBankInfo()
        self.taxNr = data.getRandomTaxNr(self.state)
        self.localCourt = ''
        self.ceo = data.getRandomFirstName()[0] + ' ' + data.getRandomLastName()

    def __str__(self):
        return f"Firma:    {self.name}, {self.street} {self.streetNumber}, {self.zip} {self.
city}, {self.country}. {self.postbox} {self.website} Mail: {self.eMail} Tel: {self.phone} Fax:
 {self.fax} Opening Hours: {self.openingHours} Bank: {self.bankInfo} SteuerNr: {self.taxNr}
Amtsgericht: {self.localCourt} CEO: {self.ceo}"

class Clerk:
    def __init__(self, company):
        firstName = data.getRandomFirstName()
        phoneAndFax = data.generatePhoneAndFax(company.areaCode)
        self.gender = firstName[1]
        self.firstName = firstName[0]
        self.lastName = data.getRandomLastName()
        self.name = self.firstName + ' ' + self.lastName
        self.company = company
        self.eMail = generateClerkEMail(self.firstName, self.lastName, self.company.name)
        self.phone = phoneAndFax[0]
        self.fax = phoneAndFax[1]

    def __str__(self):
        return f"Arbeiter: {self.firstName} {self.lastName} ({self.gender}), {self.eMail} Tel:
 {self.phone} Fax: {self.fax}"
```

```
def generateWebsite(company):
    return 'www.' + company.lower().replace(' ', '-') + '.de'

def generateClerkEMail(firstName, lastName, company):
    prefix = convertToAscii(firstName) + '.' + convertToAscii(lastName)
    rand = random.randrange(0, 10)
    if rand == 0:
        prefix = firstName[0] + '.' + convertToAscii(lastName)
    elif rand == 1 and len(firstName) >= 3:
        prefix = firstName[0:3] + '.' + convertToAscii(lastName)
    elif rand < 5:
        prefix = convertToAscii(firstName) + '-' + convertToAscii(lastName)
    result =  prefix + '@' + convertToAscii(company) + '.de'
    return result.lower()

def generateCompanyEMail(company):
    options = ['support', 'contact', 'help', 'kontakt', 'willkommen', 'info', 'information', '
sekretariat', 'office']
    result = options[random.randrange(0, len(options))] + '@' + convertToAscii(company) + '.de
'
    return result.lower()

def generateLegalForm():
    options = ['', 'OHG', 'KG', 'GbR', 'AG', 'KGaA', 'GmbH/UG', 'eG', 'AG & Co. KG', 'GmbH &
Co. KG', 'Limited & Co. KG', 'Stiftung & Co. KG', 'Stiftung GmbH & Co. KG', 'UG & Co. KG', '
GmbH & Co. OHG', 'AG & Co. OHG', 'Partenreederei', 'PartG', 'PartG mbB', 'gAG', 'GmbH', 'gGmbH
', 'InvAG', 'KGaA', 'AG & Co. KGaA', 'SE & Co KGaA', 'GmbH & Co. KGaA', 'Stiftung & Co. KGaA',
 'REIT-AG', 'UG', 'AöR', 'e. V.', 'KöR', 'VVaG', 'Stiftung']
    return options[random.randrange(0, len(options))]

def convertToAscii(word):
    return word.lower().replace(' ', '-').replace('ä', 'ae').replace('ö', 'oe').replace('ü', '
ue').replace('ß', 'ss')

def generateText(length):
    text = str(data.getRandomSentence())
    while len(text.split()) < length:
        text = str(text) + ' ' + str(data.getRandomSentence())
    return text
```

Listing A.2: Die `createData.py`-Datei

Erklärung

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

Ort, Datum, Unterschrift