



Fakultät für Informatik

Studiengang Informatik

Erstellung einer standardisierten
Material-Kostengliederung für Projekte einer
Bausoftware mittels Natural Language Processing

Bachelor Thesis

von

Florian Weidner

Datum der Abgabe: tt.mm.jjjj

Erstprüfer: Prof. Dr. Marcel Tilly

Zweitprüfer: Prof. Dr. Johannes Jurgovsky

EIGENSTÄNDIGKEITSERKLÄRUNG / DECLARATION OF ORIGINALITY

Hiermit bestätige ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken (dazu zählen auch Internetquellen) entnommen sind, wurden unter Angabe der Quelle kenntlich gemacht.

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Rosenheim, den tt.mm.jjjj

Vor- und Zuname

Abstract

Industry Foundation Classes (IFC) ist ein öffentlicher Standard im Bauwesen zur digitalen Beschreibung von Gebäudemodellen. In diesen Modellen können auch die Materialien der einzelnen Bauteile spezifiziert werden. Ziel ist es, eine Kostengliederungsstruktur in der Bausoftware ORCA AVA aus den Materialinformationen einer IFC-Datei zu generieren. Materialien können an verschiedenen Stellen im Modell angegeben werden. Ein Algorithmus soll diese Möglichkeiten in ein Interface zusammenführen. Außerdem handelt es sich bei der Materialangabe um ein Freitextfeld. Hier soll mithilfe von Natural Language Processing (NLP) und Künstlicher Intelligenz eine Lösung entwickelt werden, um eine standardisierte Liste der Materialien zu erschaffen. **Todo: Am Ende neu schreiben**

Inhaltsverzeichnis

1	Einleitung	1
1.1	Ausgangssituation	1
1.2	Motivation	2
1.3	Ziel der Arbeit	3
1.4	Wissenschaftliche Vorgehensweise	5
2	Grundlagen	6
2.1	Projektmanagement	6
2.1.1	Vorgehensmodell	6
2.1.2	DevOps	7
2.2	IFC	7
2.2.1	Geschichte	7
2.2.2	Format	8
2.2.3	Verwendung	8
2.2.4	Möglichkeiten für die Materialangabe eines Bauteils	9
2.3	Das Format Kostengliederung in der ORCA AVA	10
3	Problemstellung und Anforderungen	12
3.1	Problemstellung	12
3.2	Anforderungen	13
3.2.1	Funktionale Anforderungen	13
3.2.2	Weitere Anforderungen	14
4	Theoretische Konzeption	15
4.1	Grundlegende Architektur und Vorgehensweise	15
4.1.1	Implementierung als Web-Service	16
4.1.2	Konzept der Materialstrukturierung	17
4.1.3	Konzept für das Auslesen von Materialien aus einer IFC Datei	19
4.1.4	Preprocessing der Materialien	19

4.2	Algorithmen für die Textklassifizierung	20
4.2.1	Ablauf einer Textklassifizierung	20
4.2.2	Kriterien der Messbarkeit	21
4.2.3	ML.NET Multiclass Classification	23
4.3	Möglichkeiten für die Feinstrukturierung	25
4.3.1	Spezifikation und Kriterien der Messbarkeit	25
4.3.2	Generative Pretrained Transformer (GPT) Modell von OpenAI .	26
4.3.3	fastText mit Density-Based Spatial Clustering of Applications with Noise (DBSCAN)	28
5	Gegenüberstellung der möglichen Konzepte	32
5.1	Vergleich der Textklassifizierung	32
5.1.1	Auswertung	32
5.1.2	Auswahl eines Algorithmus	33
5.2	Vergleich der Feinstrukturierung	33
5.2.1	Auswertung	34
5.2.2	Auswahl eines Algorithmus	35
6	Praktische Umsetzung	36
6.1	Implementierung des ASP.NET Services	36
6.2	Python-Interop	37
6.3	Implementieren des Auslesens aller Materialien aus einer IFC Datei . .	38
6.4	Implementierung der Material-Strukturierung	38
6.4.1	Erstellen einer Datengrundlage	38
6.4.2	Implementierung des Preprocessings	39
6.4.3	Training des Textklassifizierungsmodells	39
6.4.4	Training des <i>fastText</i> Modells	41
6.4.5	Ausführen der Feinstrukturierung	41
7	Maßnahmen zur Qualitätssicherung	43
7.1	Clean Code	43
7.2	Technische Hilfsmittel	43
7.3	Tests und Abnahme	44
8	Abschluss	46
8.1	Fazit	46
8.2	Integration in die ORCA AVA	48

8.3 Ausblick	48
A Anhang	49
B Abkürzungsverzeichnis	67
C Definitionsverzeichnis	69
D Quellcodeverzeichnis	69
Literaturverzeichnis	71

Abbildungsverzeichnis

1.1	Relevanz Building Information Modeling (BIM)	3
1.2	Google Trends	4
1.3	Kano Modells	5
2.1	IfcMaterial	10
4.1	Anwendungsfalldiagramm	16
4.2	Dokumentation	17
4.3	Ablauf	17
4.4	Textklassifizierung	21
4.5	CBOW	29
A.1	IFC Manager	49
A.2	IfcMaterialLayerSet	49
A.3	MaterialCategories	50
A.4	IfcMaterialProfileSet	51
A.5	IfcMaterialConsituentSet	51
A.6	CostStructure	51
A.7	Verteilungsdiagramm	52
A.8	Import	53
A.9	Takeover	53
A.10	Rating	54
A.11	Rating	54
A.12	CBOW	55

1 Einleitung

Inhalt dieser wissenschaftlichen Arbeit ist das automatische Erstellen einer Material-Kostengliederung aus einer IFC-Datei mithilfe von NLP zur Erweiterung einer Bausoftware. Diese Bausoftware ist die ORCA AVA aus dem mittelständigen Softwarehaus „ORCA Software GmbH“ aus Neubuern. In diesem Kapitel soll eine kurze Einführung über die „ORCA Software GmbH“ und das Produkt ORCA AVA gegeben werden. Außerdem wird die Motivation für die Programmerweiterung und die wissenschaftliche Vorgehensweise dieser Arbeit beschrieben.

1.1 Ausgangssituation

Die im Titel beschriebene Bausoftware ist die ORCA AVA aus dem Softwarehaus „ORCA Software GmbH“. Dieses wurde im Jahr 1990 von Dipl.-Ing. Siegfried Tille und Dipl.-Ing. Heinz Nießen gegründet. Der Hauptsitz des Unternehmens ist in Neubuern, bei Rosenheim. Das Unternehmen ist auf die Produktentwicklung von Software für die Baubranche spezialisiert. Im Vordergrund stehen die Ausschreibungssoftware ORCA AVA und die Ausschreibungstext-Plattform AUSSCHREIBEN.DE (ADE). Ziel der Entwicklung ist es die Ausschreibung, Vergabe und Abrechnung (AVA) eines Bauvorhabens für Planer, Architekten und Bauingenieure zu vereinfachen. Der Leitfaden ist, Software zu entwickeln, die jeder versteht, intuitiv bedienbar ist, einen optimalen Workflow gewährleistet und viele Import- und Exportmöglichkeiten für den Datenaustausch bietet. Diese Arbeit fokussiert sich auf eine Erweiterung der ORCA AVA. Sie ist für alle Architektur- und Ingenieurbüros, Wohnungsbaugesellschaften, Unternehmen und Behörden zur einfachen Abwicklung von Bauprojekten mit Ausschreibung, Vergabe und Abrechnung. Zusätzlich bildet sie das Kostenmanagement von solchen Projekten ab. Die Software ist außerdem BIM fähig und bietet Deutsches Institut für Normung e. V. (DIN) zertifizierte Schnittstellen für den Datenaustausch an. Neben der ORCA AVA gibt es den IFC-Manager als eigene Instanz, der IFC-Modelle anzeigen kann. Die ORCA AVA kann dann Mengen aus dem Gebäudemodell übernehmen. Es stehen drei verschiedene Editionen der Software zur Verfügung. Die ORCA AVA Starter Edition (SE), die ORCA

AVA Professional Edition (PE) und die ORCA AVA Enterprise Edition (EE). Die aktuellste Version ist die 25.0.

Technisch wird die ORCA AVA in .NET entwickelt. Ein Großteil der Anwendung besteht noch aus Visual Basic (VB) Code. Alle neuen Komponenten und Erweiterungen werden in C# implementiert. Neue Graphical User Interface (GUI)-Komponenten werden dementsprechend mit Windows Presentation Foundation (WPF) entwickelt. WPF ist ein .NET Framework für das Erstellen von Windows Applikationen mit grafischer Benutzeroberfläche von Microsoft. (vgl. Microsoft, 2022*d*) Die ORCA AVA und der IFC Manager (siehe Unterabschnitt 2.2.3) laufen in eigenen Prozessen und kommunizieren auf Prozessebene in der lokalen Umgebung.

1.2 Motivation

Ziel der „ORCA Software GmbH“ ist es, BIM noch mehr in die ORCA AVA zu integrieren. BIM bedeutet, dass die Planung von Bauprojekten vollständig auf digitaler Basis durchgeführt wird. Für jeden Projektbeteiligten besteht somit jederzeit Zugriff auf alle projektrelevanten Daten über Kosten, Mengen und Zeitabläufen. Somit können Baukosten einfacher ermittelt und der Bauprozess besser überwacht werden. In Abbildung 1.1 ist zu sehen, dass BIM ein relevanter Begriff in der Baubranche ist. Die Verwendung der Praxis ist allerdings noch nicht verbreitet. (vgl. Thomas, Baumanns and Dr. Philipp-Stephan, Freber and Dr. Kai-Stefan, Schober and Dr. Florian, Kirchner, 2016, p. 20) Mit dem „IFC First“ Ansatz ist das langfristige Ziel der ORCA AVA das Thema BIM noch mehr abzudecken. Aus den Daten eines 3D-Gebäudemodells soll automatisch ein Ausschreibungstext erstellt werden können. Bauteile aus dem Gebäudemodell sollen dann mit Kurztext, Langtext, Menge, Preis und vordefinierten Kostengliederungen in den Programmteil Bauelemente der ORCA AVA übernommen werden. Aufgrund der hohen Relevanz des Themas, spricht das Ganze viele Kunden an und stellt somit ein effektives Werbemittel für den Verkauf der Software dar. Ein Teil des langfristigen Ziels ist die Übernahme der Baumaterialien aus einer IFC-Datei. Diese bietet die erste Kostengliederung für den „IFC First“ Ansatz. Die Übernahme von weiteren Daten aus dem IFC-Modell können darauf aufgebaut werden.

Ein weiterer Trendbegriff, der mit der Programmerweiterung abgedeckt werden soll, ist „Künstliche Intelligenz“ und „Maschinelles Lernen“. Das Suchinteresse der beiden Themen ist in Abbildung 1.2 zu sehen. Die Werte geben das Google-Suchinteresse relativ zum höchsten Punkt im angegebenen Zeitraum an. Der Begriff „Maschinelles Lernen“ hat seit Anfang 2015 ein steigendes Suchinteresse. Bei „Künstliche Intelligenz“ ist das

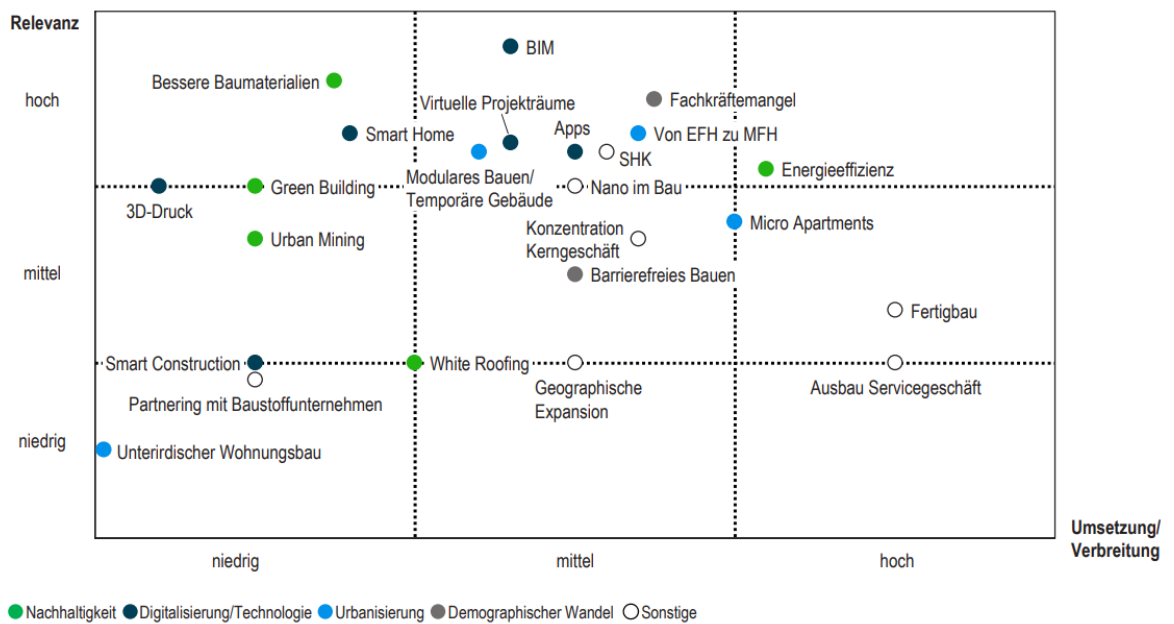


Abbildung 1.1: Megatrends Nachhaltigkeit und Digitalisierung in der Bauwirtschaft. Bilder aus (vgl. Thomas, Baumanns and Dr. Philipp-Stephan, Freber and Dr. Kai-Stefan, Schober and Dr. Florian, Kirchner, 2016, p. 20)

Suchinteresse von Anfang 2017 bis Ende 2021 konstant hoch. Der starke Anstieg ab November 2022 ist signifikant und korreliert mit der Veröffentlichung der Software ChatGPT. Man erkennt insgesamt, dass das Interesse über die letzten Jahre stetig ansteigt. Die Abdeckung dieser Begriffe ist seit geraumer Zeit ein Wunsch der Vertriebsseite für den noch besseren Verkauf der Software. Die Erweiterung wäre der erste Einsatz von maschinellem Lernen. Es ist also zusätzlich auch ein Pilotprojekt in der ORCA AVA Entwicklung, um sich mit Machine-Learning-Algorithmen vertraut zu machen und Erfahrungen in diesem Themengebiet zu bekommen.

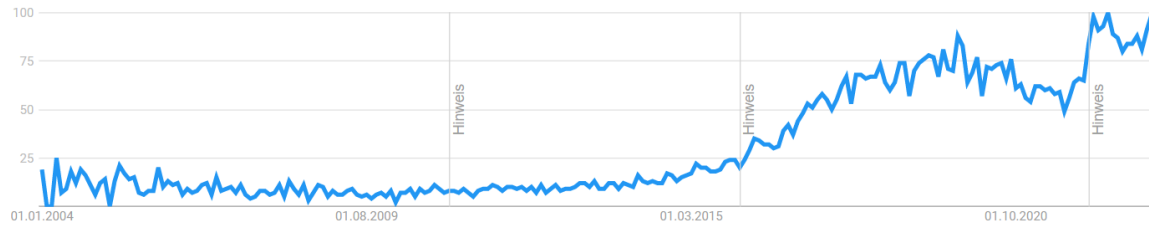
Durch die genannten Punkte ist die Programmerweiterung dem Kano-Modell nach als Begeisterungsfeature zuzuordnen. Die Kundenzufriedenheit steigt also exponentiell mit dem Erfüllungsgrad der Anforderung. (siehe Abbildung 1.3) Mit der Zeit wandelt sich es dann erst in ein Leistungsfeature und irgendwann in ein Basisfeature. (vgl. Hölzing, 2008, p. 3-4) Es besteht also die Motivation die Erweiterung möglichst exklusiv zur Verfügung zu stellen. Es eignet sich somit sehr für die ORCA AVA EE Edition.

1.3 Ziel der Arbeit

Um die Anforderungen aus Abschnitt 3.2 umsetzen zu können, muss zuvor ein Ziel formuliert werden. Die Zieldefinition für diese Arbeit wird nach der SMART Technik



(a) Suchinteresse: Künstliche Intelligenz



(b) Suchinteresse: Machine Learning

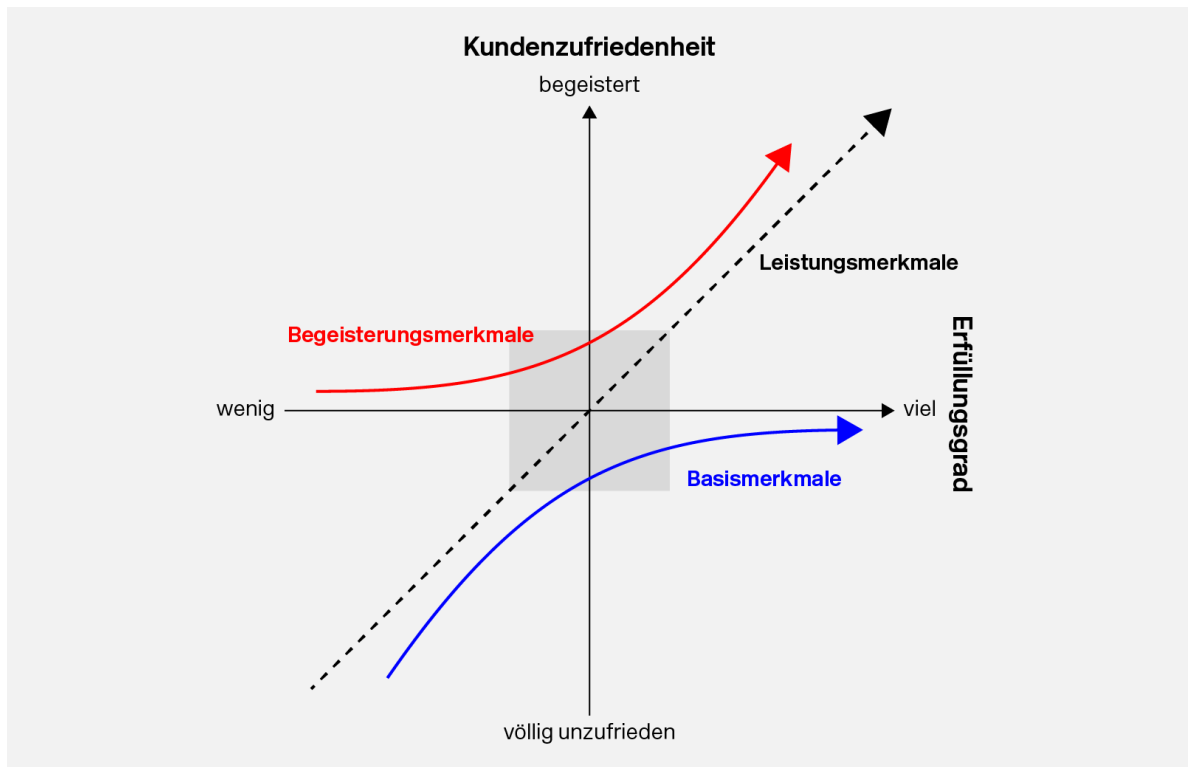
Abbildung 1.2: Google Suchinteresse der beiden Begriffe „Künstliche Intelligenz“ und „Maschinelles Lernen“ seit 2004 bis 2023

definiert. Dabei muss ein Ziel folgende Eigenschaften haben:

- Spezifisch
- Messbar
- Attraktiv
- Realisierbar
- Terminiert

Anhand dieser Kriterien wurde folgendes Ziel definiert:

Ziel. Bis zum 15.06.2023 werden für die Implementierung passende Algorithmen nach den Kriterien der Messbarkeit (siehe Unterabschnitt 3.2.1) ausgewählt. Dazu wird ein „Draftprojekt“ mit allen vollständigen funktionalen Anforderungen für die ORCA AVA implementiert. Zusätzlich sollen die Maßnahmen zur Qualitätssicherung (Kapitel 7) durchgeführt werden und die Qualitätssicherungs-Abteilung alle Funktionen freigegeben haben.



Quelle: <https://dmexco-lightsails-media.s3.eu-central-1.amazonaws.com/wp-content/uploads/2021/01/04112832/Kano-Modell.png> (Accessed: 2023-2-24)

Abbildung 1.3: Kano Modell der Kundenzufriedenheit

1.4 Wissenschaftliche Vorgehensweise

Im Folgenden wird die wissenschaftliche Vorgehensweise der Arbeit aufgezeigt. In Kapitel 1 wurde bereits die Ausgangssituation beschrieben, das Entwickeln der Programmerweiterung motiviert und das Ziel dieser Arbeit definiert. Im nächsten Kapitel, Kapitel 2, werden Grundlagen erläutert, auf die diese Arbeit aufbaut. Im anschließenden Kapitel 3 sollen die Problemstellung und Anforderungen analysiert werden. Danach (Kapitel 4) wird eine Lösung für die Problemstellung theoretisch konzipiert und verschiedene mögliche Algorithmen analysiert. In Kapitel 5 werden die Algorithmen gegenübergestellt und nach den definierten Kriterien ausgewählt. Kapitel 7 zeigt die genutzten Maßnahmen der Qualitätssicherung auf. Am Ende wird das Ergebnis bewertet und ein Ausblick gegeben.

Todo: Muss noch vervollständigt werden

2 Grundlagen

In diesem Kapitel werden die Grundlagen im Bereich der fachspezifischen Themen, Technik und Projektmanagement vermittelt. Das soll Verständnis für die folgenden Kapitel schaffen. Zuerst geht es um das Projektmanagement. Anschließend geht es um das Format IFC, dessen Geschichte und dessen Nutzungsmöglichkeiten für diese Arbeit. Außerdem wird die Struktur und der Nutzen einer Kostengliederung in der ORCA AVA veranschaulicht.

2.1 Projektmanagement

Bevor auf technische und fachliche Aspekte von IFC und Kostengliederungen eingegangen wird, folgt die Einführung in das Projektmanagements mit SCRUM und Development Operations (DevOps).

2.1.1 Vorgehensmodell

Das Projekt wurde mithilfe agiler Softwareentwicklung durchgeführt. Im ORCA AVA Entwicklungsteam wird das SCRUM Modell verwendet. Die Entwicklung der Produkterweiterung in dieser Arbeit läuft in diesem SCRUM-Prozess.

Definition 1 (Scrum). „*Scrum is a lightweight framework that helps people, teams and organizations generate value through adaptive solutions for complex problems.*“ (vgl. Ken and Jeff, 2020, p. 3)

SCRUM verwendet einen iterativen, inkrementellen Ansatz. So kann Sprint für Sprint aus gewonnenen Erfahrungen der Entwicklungsablauf optimiert werden. (vgl. Ken and Jeff, 2020) Product Owner in der ORCA AVA Entwicklung ist die „ORCA Software GmbH“, in Vertretung des Produktmanagement-Teams. Die Umsetzung der Sprint-Ziele wird von einer Person implementiert.

2.1.2 DevOps

Die Bezeichnung DevOps vereint die beiden Praktiken „Development“ (Entwicklung) und „Operations“ (Vorgänge). Die traditionelle Trennung von Entwicklung und Softwarebetrieb führt oft zu Interessenskonflikten. Entwickler wollen stetig die Software verbessern und der Betrieb will Änderungen vermeiden, um die Stabilität des Systems zu gewährleisten. Durch DevOps entsteht ein Softwareentwicklungsprozess, den man durch Praktiken wie Continuous Integration, Continuous Delivery, Continuous Deployment, automatisiertes Testen, Infrastructure-as-Code und automatische Veröffentlichungen beschleunigt. DevOps steht auch für eine Entwicklungskultur mit offener Zusammenarbeit, Kommunikation, Transparenz und Eingestehen von Fehlern, um Konflikte im Team zu vermeiden. (vgl. Beetz and Harrer, 2021) Im Entwicklungsteam der ORCA AVA wird diese Praktik umgesetzt. Die technischen Hilfsmittel, die für den DevOps Prozess verwendet werden, sind in Abschnitt 7.2 beschrieben.

2.2 IFC

Die Daten für die Material-Kostengliederung werden aus einem digitalen Gebäudemodell entnommen. Der öffentliche internationale Standard (ISO 16739-1:2018) für Gebäudemodelle ist IFC. (vgl. *Industry Foundation Classes*, 2017) Dieser wird auch in der bestehenden ORCA AVA benutzt, um den Ausschreibungsprozess zu unterstützen. IFC Dateien können geöffnet, angeschaut und Informationen über das Modell in die Hauptsoftware übernommen werden. Im folgenden Kapitel wird die Geschichte, das Format von IFC und die Verwendung in der ORCA AVA erläutert.

2.2.1 Geschichte

IFC ist die Hauptleistung der buildingSMART International, Ltd. Die non-profit Organisation will mit der Spezifikation den BIM Prozess fördern und voranbringen. (vgl. Grutters, 2022) Angefangen hat die Organisation als der Verein *Industriellianz für Interoperabilität IAI e. V.* mit Sitz in Berlin. 1994 startete die Entwicklung an dem offenen Datenmodellstandard IFC. Dieser sollte die Anforderungen der Industrie an Interoperabilität gerecht werden und eine gemeinsame Basis zum Austausch von Informationen durch verschiedenen Anwendungen schaffen. Im Zusammenhang mit BIM sollten Daten lesbar, editierbar für verschiedene Systeme durch den Bauprozess und kompletten Lebenszyklus eines Gebäudes geteilt werden. (vgl. Laakso and Kiviniemi, 2012) Nach einigen Prototypen wurde 1999 IFC 2.0 veröffentlicht. Diese wurde bis 2007

mit der Version 2.3.0.1 stetig verbessert. Die Version 2.3 wird auch in heutigen Projekten noch verwendet. 2013 wurde IFC4 veröffentlicht, welche mit der Version 4.0.2.1 die aktuellste, offizielle IFC Version ist. Das Format wird weiterhin stetig weiterentwickelt. Neue Versionen stehen schon vor einer Abstimmung der International Organization for Standardization (ISO). (vgl. van Berlo, 2022) Es gibt folgendermaßen aktuell zwei offizielle Versionen. Beide werden in der Praxis benutzt und sind, wie in Unterabschnitt 2.2.3 beschrieben, mit der ORCA AVA kompatibel.

2.2.2 Format

Das IFC Format kodiert folgende Daten:

- Identität, Semantik, Attribute und Relationen von Objekten
- Abstrakte Konzepte wie Performance oder Kosten
- Prozesse wie z.B. Installationen und Operationen
- Personen wie z.B. Eigentümer oder Lieferanten

Die Spezifikation kann also für das Bauen, Betreiben oder Nutzen eines Gebäudes genutzt werden. IFC ist ein Implementierungs-Unabhängiges Datenmodell, welches in verschiedenen Umgebungen und elektronischen Formaten benutzt werden kann. Es kann beispielsweise in ein relationales Datenbankschema gegossen oder auch als Dateiformat implementiert werden. Das weitverbreitetste Format ist Standard for the exchange of product model data (STEP) Physical Format (SPF) (vgl. Laakso and Kiviniemi, 2012; Grutters, 2022). SPF ist das kompakteste Format für den dateibasierten Import und Export von IFC-Dateien. Die Dateiendung des Formates ist „.ifc“. Des Weiteren kann IFC als Extensible Markup Language (XML) oder ZIP Datei verwendet werden. (vgl. Grutters, 2022)

2.2.3 Verwendung

„Today, IFC is typically used to exchange information from one party to another for a specific business transaction.“ (Grutters, 2022)

In der ORCA AVA wird IFC für das Einlesen und Übernehmen von Maßen und Mengen in die Ausschreibung verwendet. Es vereinfacht den Prozess, die in der Computer-aided design (CAD)-Software erstellten Daten einfach in die Leistungsverzeichnisse der ORCA AVA zu überführen. Die ORCA AVA kann IFC-Dateien einlesen. Im IFC Manager

wird das 3D-Modell dann angezeigt. In dem geöffneten Fenster gibt es einige fachliche Ansichten, die jegliche IFC-Daten nochmal fachlich abstrahieren. In den Ansichten können zum Beispiel bestimmte Maße oder die Anzahl verschiedener Bauteile, die aus dem IFC-Modell berechnet werden, in die ORCA AVA übernommen werden. Auch alle im IFC definierten Eigenschaften eines Bauteils sind in der Eigenschaften-Ansicht sichtbar. Hier kann man auch die Materialbezeichnung des Bauteils finden, die im Modell hinterlegt ist. In Abbildung A.1 ist die Oberfläche des IFC Managers zu sehen. Man sieht die Materialangabe rechts unten im Eigenschaftsfeld unter dem 3D-Modell.

Für das Arbeiten mit IFC-Dateien wird die open-source Bibliothek *xbim-toolkit* verwendet. Die .NET Bibliothek kann IFC-Dateien lesen, schreiben und anzeigen. Außerdem unterstützt es bei der Berechnung von komplexer Geometrie, um die Modelldaten für Analysen nutzbar zu machen. Die Entwicklung der Bibliothek startete 2007 und läuft seit 2009 in Zusammenarbeit mit der Norhumbria University. Mittlerweile bildet es die Standards IFC Version 2.3 (IFC2x3) und IFC Version 4 (IFC4) zu 100% ab. Zudem bietet es an, auch IFC2x3 Modelle über das IFC4 Interface anzuprogrammieren. Somit können mit einer Codebasis beide Formate abgebildet und unterstützt werden. (vgl. Xbim Ltd., n.d.)

2.2.4 Möglichkeiten für die Materialangabe eines Bauteils

In einem IFC-Modell können Materialien auf unterschiedliche Weise einem Bauteil zugewiesen werden. Die Spezifikation bietet die Klasse *IfcMaterial*. Diese bildet fachlich ein Material ab. Es hat die Attribute Name, Description und Category. (vgl. buildingSMART International Ltd., 2017a) Eine Instanz von *IfcMaterial* kann mit einem Element oder Elementtyp über die *IfcRelAssociatesMaterial* verbunden werden. Diese Zuweisung kann über verschiedene Weisen stattfinden. In Abbildung 2.1 ist eine direkte Zuweisung über das *IfcRefAssociatesMaterial* zu sehen. Hier hängt das *IfcMaterial* direkt am Element. Bei einer Platte mit mehreren Schichten kann das *IfcMaterial* auch wie in Abbildung A.2 zu sehen an einem *IfcMaterialLayerSet* zugewiesen sein. Jede Schicht hat hier seine eigene Dicke und eigene Materialangabe. Ein Träger kann das *IfcMaterial* über ein *IfcMaterialProfileSet* abgebildet haben. (siehe Abbildung A.4) Bei zum Beispiel einer Tür können verschiedene Materialien des Rahmens und der Verglasung über das *IfcMaterialConstituentSet* abgebildet werden. (siehe Abbildung A.5) (vgl. buildingSMART International Ltd., 2017b) Zusätzlich gibt es noch ältere Möglichkeiten Materialangaben an ein Element zu binden. In IFC2x3 gibt es die Klasse *IfcMaterialList* (vgl. Thomas et al., 2007). Diese Möglichkeit ist deprecated, in der Praxis kommt das IFC2x3 Format

noch regelmäßig vor. Die *IfcMaterialList* muss also auch beachtet werden.

Neben verschiedenen Verbindungen zu *IfcMaterial* hat jedes Element bei IFC auch verschiedene Property Sets. Property Sets sind Container, die Eigenschaften je nach ihrem Namensattribut enthalten. Einige Property Sets sind in der Spezifikation des IFC Standards enthalten. Es können auch zusätzlich benutzerdefinierte Property Sets erfasst werden. Diese können auch Informationen über das Material eines Elements enthalten. (vgl. buildingSMART International Ltd., 2017c) Das PropertySet *Pset_MaterialConcrete* weist darauf hin, dass es sich z.B um Beton handelt.

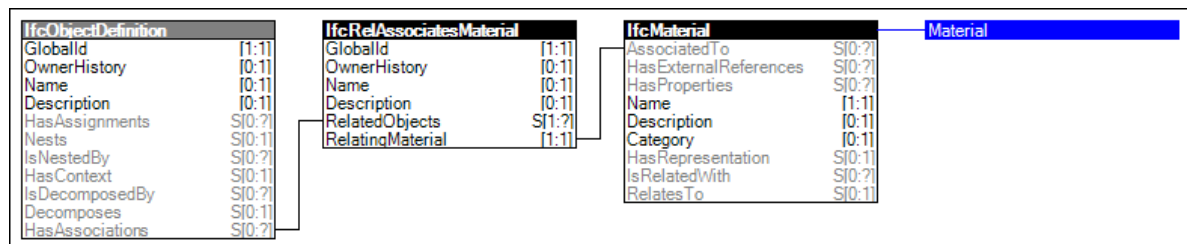


Abbildung 2.1: Material Single Association

2.3 Das Format Kostengliederung in der ORCA AVA

Die Kostengliederung bietet eine Struktur, um Gesamtkosten einer Baumaßnahme in Kostengruppen unterteilt auswerten zu können. Logisch zusammengehörende Kosten können so in eine Kostengruppe (KG) zusammengerechnet werden. Außerdem handelt es sich beim Aufbau um eine Baumstruktur, wodurch Kostengruppen hierarchisch addiert werden können. In Abbildung A.6 ist ein Beispiel einer Kostengliederung mit Materialien zu sehen. Der Gliederungspunkt *Mineralisch* wird in die Unterpunkte *Mauerwerk*, *Beton*, *Dachziegel* und *Kies* unterteilt. Die Kosten der Unterpunkte können also für den Gliederungspunkt *Mineralisch* addiert werden, um die Kosten aller Mineralien zu erfassen. Bei einem Bauprojekt kann man so Kosten über alle Projektphasen vergleichen. Von der Kostenschätzung über die Ausschreibung bis zur Rechnungsfreigabe. So können Kostenauswertungen nach den verschiedenen Kostengruppen durchgeführt werden. In einem ORCA AVA Projekt kann man verschiedene Kostengliederungen definieren. Es existieren bereits Standardkostengliederungen beim Erstellen eines neuen Projektes. Zusätzlich können neue Kostengliederungen erstellt oder importiert werden. (vgl. ORCA Software GmbH, n.d.)

Technisch ist eine Kostengliederung als Modell im C# Code definiert. Der Aufbau bildet die Baumstruktur über eine Referenz zur ParentNode und mehreren ChildrenNo-

des ab. Das Model kann über die ORCA AVA interne Middleware in der Datenbank persistiert werden. Die persistierten Kostengliederungen werden dann in entsprechenden Programmteilen angezeigt. So müssen nach der generierten Kostengliederungs-Struktur die strukturierten Materialbezeichnungen in das C# Modell abgebildet werden. So wird nach dem Import die erstellte Kostengliederung automatisch in der Oberfläche angezeigt und kann für Kostenauswertungen genutzt werden.

3 Problemstellung und Anforderungen

Im folgenden Kapitel werden die Problemstellung und Anforderungen aufgeführt, um das Projekt einzugrenzen. Diese kommen vom Auftraggeber, der „ORCA Software GmbH“ selbst.

3.1 Problemstellung

Die Problemstellung entsteht aus der Ausgangssituation (Abschnitt 1.1) und der Motivation (Abschnitt 1.2) der Bausoftware ORCA AVA. Der BIM Prozess soll noch mehr in die Software integriert werden. Deswegen wurde sich die Programmerweiterung der Materialkostengliederung gewünscht, um den ersten Schritt zu „BIM First“ (siehe Abschnitt 1.2) zu schaffen. Die Problemstellung dieser Arbeit bezieht sich der dementsprechenden Umsetzung dieser Programmerweiterung:

Problemstellung 1. *Wie lässt sich am besten eine Liste vom Materialbezeichnungen in eine hierarchische Material-Kostengliederung strukturieren?*

Da es sich hier um eine offene Frage handelt, sind Spielräume für die Antwort offen. Es soll ein Konzept und eine Architektur für die Programmerweiterung entstehen. Um ein fachlich richtiges Ergebnis der Strukturierung zu erreichen, ist aber vor allem folgende Problemstellung wichtig:

Problemstellung 2. *Welcher NLP Algorithmus eignet sich am besten für das Klassifizieren und Strukturieren von Material-Daten?*

Bei dieser offenen Frage, soll am Ende ein Algorithmus aus einer Auswahl ausgewählt werden. Um verschiedene Ansätze und Algorithmen vergleichen zu können, benötigt man Kriterien der Messbarkeit. Diese werden in Unterabschnitt 3.2.1 definiert. So kann man neben der Berücksichtigung der Anforderungen den geeignetsten Algorithmus für das Problem auswählen.

3.2 Anforderungen

Alle definierten Anforderungen kommen vom Projektmanagement Team der ORCA Software GmbH und dem Teamleiter der ORCA AVA Entwicklung. Die Gesamtheit von Anforderungen kann in funktionale Anforderungen, Leistungsanforderungen, spezifische Qualitätsanforderungen und Randbedingungen unterteilt werden.(vgl. Glinz, 2007)

3.2.1 Funktionale Anforderungen

Definition 2 (Funktionale Anforderung). „*A functional requirement is a requirement that pertains a functional concern.*“(Glinz, 2007)

Wie in Definition 2 beschrieben, geht es bei einer funktionalen Anforderung um die Funktion einer Software. Die im Mittelpunkt stehende Funktion ist die erläuterte Erstellung der hierarchischen Materialstruktur (Anforderung 1). Weitere Funktionen sind die Anforderung 2 und Anforderung 3. Nach dem Importieren der Materialstruktur soll bei der Übernahme von Mengen die Materialkostengliederung automatisch zugewiesen werden. Zusätzlich sollen die Ergebnisse sich stetig verbessern. Dafür soll eine Möglichkeit geschaffen werden, dass Nutzer der ORCA AVA Verbesserungen eintragen und somit den Datenbestand erweitern und verbessern.

Anforderung 1. *Aus einer Liste von Materialbezeichnungen soll eine hierarchische Baumstruktur entstehen, welche in einer ORCA AVA Kostengliederung benutzt werden kann.*

Anforderung 2. *Bei der Übernahme einer Menge aus dem IFC Manager soll die Materialkostengliederung zugewiesen werden.*

Anforderung 3. *Bei schlecht generierter Struktur soll eine persistente, dauerhafte Verbesserung des Benutzers möglich sein.*

Die in Anforderung 1 generierte Materialkostengliederung soll fachlich möglichst richtig sein. Dazu werden Kriterien der Messbarkeit definiert, um den passensenden Algorithmus auswählen zu können. Diese sind in Unterabschnitt 4.2.2 für die Textklassifizierung und in ?? zu finden. Die funktionalen Anforderungen werden noch nicht direkt in die ORCA AVA integriert. Ein Testprojekt soll zur Demonstration der Funktionen dienen. Es wird allerdings einen Vorschlag zur Integration in die Oberfläche der ORCA AVA geben. (siehe Abschnitt 8.3)

3.2.2 Weitere Anforderungen

Neben den funktionalen Anforderungen gibt es vor allem technische Anforderungen an die Programmerweiterung. Diese wurden vom Teamleiter der ORCA AVA Entwicklung gestellt:

Anforderung 4. *Die Programmerweiterung soll wenn möglich in .NET (C#) implementiert werden. Darüber hinaus ist auch das Nutzen von Python möglich.*

Die technische Einschränkung der Anforderung 4 macht den Code und die Programmerweiterung für die ORCA Software GmbH wartbar. C# ist die Hauptsprache der Firma. Durch das dementsprechende Wissen in der Entwicklung, kann bei Fehlerbehebungen eine erneute Einarbeitungszeit vermieden werden. Zusätzlich gibt es auch schon ein Modul für ADE in Python.

Anforderung 5. *Die Strukturierung soll mit der Erweiterung der Datengrundlage immer besser gemacht werden können.*

Durch das Einlesen neuer IFC-Dateien von Benutzern, werden immer neue Materialbezeichnungen gesammelt, die die Datengrundlage erweitern. Dadurch müssen immer wieder neue Begriffe klassifiziert werden können, um das Modell stetig zu verbessern. Dementsprechend müssen Künstliche Intelligenz (KI)-Modelle auch immer wieder neu trainierbar sein. Diese Funktionen sollen also einem ORCA Internen Mitarbeiter zur Verfügung stehen.

Durch die beschriebenen Anforderungen aus Abschnitt 3.2 entstand das beschriebene Ziel der Arbeit (siehe Abschnitt 1.3)

4 Theoretische Konzeption

In diesem Kapitel wird ein theoretisches Konzept für das Lösen der Problemstellung erstellt und begründet. Zuerst wird das allgemeine Konzept vorgestellt. Dann werden mögliche Algorithmen erläutert und evaluiert, wie sie zu den Anforderungen passen. Im Folgekapitel werden die Algorithmen miteinander verglichen und der geeignetste ausgewählt.

4.1 Grundlegende Architektur und Vorgehensweise

In diesem Abschnitt wird das erarbeitete Konzept für die Erweiterung vorgestellt und begründet. In Abbildung 4.1 ist ein Anwendungsfalldiagramm der neuen Funktionen zu sehen. Die Anwendungsfälle des Benutzers der ORCA AVA decken sich mit den schon beschriebenen funktionalen Anforderungen (Unterabschnitt 3.2.1) Die zusätzlichen Anwendungsfälle ergeben sich aus Anforderung 5.

Das Ergebnis des Hauptanwendungsfalles der Material-Strukturierung ist abhängig von den Inhalten der IFC-Datei. Durch die Nutzung verschiedener CAD-Programme und eigene Vorlieben bei der Materialangabe der ORCA AVA Benutzer entstehen viele unterschiedliche Inhalte bei der Materialangabe. Um die hierarchische Strukturierung zu vereinfachen, werden Materialien zuerst nach dem „Teile-und-hersche“ Verfahren in Oberkategorien klassifiziert. In Abbildung A.3 sind die Kategorien mit Beispielen aufgelistet. So müssen anschließend nur noch eine kleinere Liste von Materialien einer Oberkategorie weiter hierarchisch strukturiert werden.

In Abbildung A.7 ist ein Verteilungsdiagramm mit allen relevanten Objekten zu sehen. Auf dem Lokalem Computer des Benutzers laufen die beiden Prozesse der ORCA AVA und des IFC Managers. Die ORCA AVA kommuniziert mit einem Webserver über Representational State Transfer (REST). Dieser verwaltet die anhängende SQL-Datenbank und übernimmt das Strukturieren des Material-Kostengliederungs-Import.

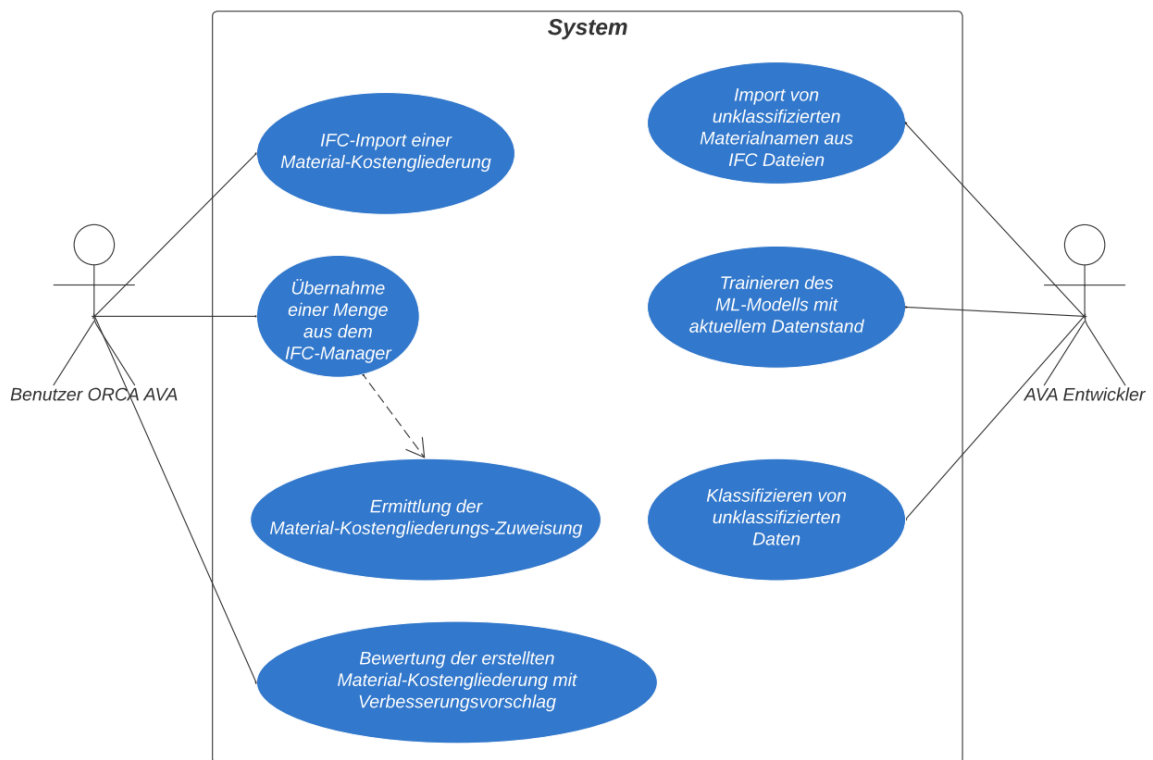


Abbildung 4.1: Anwendungsfalldiagramm

4.1.1 Implementierung als Web-Service

Für die Materialstrukturierung ist ein neuer Webservice vorgesehen, mit den die ORCA AVA kommunizieren kann. Dies bietet einige Vorteile:

- Der Import greift immer auf das aktuellste KI-Modell zu, welches zentral verwaltet werden kann
- Wegen der nötigen Verwendung von Python muss nicht bei jedem Client eine lauffähige Pythonumgebung vorhanden sein oder durchs Setup installiert werden.
- Neue Materialbezeichnungen und Klassifizierungen werden direkt zentral in der Datenbank persistiert.
- Eine Änderung/Verbesserung der Datengrundlage kann durch automatisches Trainieren direkt zu einem verbessertem KI-Modell führen.

Für die ORCA AVA bestehen schon Services für die Bereitstellung von News, die Lizenzierung oder das Updaten der Software. Durch die Vorgabe der ORCA Software GmbH wird der Service mit dem Framework ASP.NET implementiert. ASP.NET ist

ein Open-Source und Plattform-unabhängiges Web-Framework für die Entwicklung cloudbasierter Anwendungen wie Web-Applikationen, Internet of Things (IoT)-Apps oder mobile Backends. (vgl. .NET Foundation and Contributors, n.d.). Die Architektur dieses Services entspricht dementsprechend den schon existierenden Services.

Daten werden über den Service in einer SQL-Datenbank Server gespeichert. Hier werden auch die ORCA Software GmbH internen Standards verwendet. Daten für den Material-Kostengliederungs-Import sind in der Tabelle *Materials* persistiert. Die Tabellendokumentation ist in Abbildung 4.2 zu sehen. Zusätzlich bestehen Tabellen für die Autorisierung der Zugriffe auf den Service.

Name	Typ	Beschreibung	Besonderheit
Id	INT	Generierte Id	Primary Key
MaterialBezeichnung	NVARCHAR(100)	Materialname	
MaterialCategory	INT	Zugewiesene Material-Überkategorie (siehe enum: MaterialCategory)	
IfcFile	NVARCHAR(255)	interner Pfad zu IFC-Datei, die diese Materialbezeichnung enthält	
BauteilGuid	NVARCHAR(100)	GUID des Bauteiles in der IfcFile angegebenen IFC-Datei	
Created	DATETIME2	Datum der Erstellung dieses Datensatzes	Default Value: getutcdate()
Updated	DATETIME2	Datum der Aktualisierung dieses Datensatzes (z.B. Klassifizierung)	

Abbildung 4.2: Dokumentation der Tabelle *Materials* der Datenbank

4.1.2 Konzept der Materialstrukturierung

Der Ablauf der Materialstrukturierung ist in drei Schritten aufgeteilt. Materialien werden nach dem Einlesen zuerst nach dem Preprocessing in übergeordnete Kategorien klassifiziert. Danach wird jede Überkategorie nochmals feiner strukturiert. Dieser Ablauf ist in Abbildung 4.3 anhand eines Beispiels zu sehen. Am Ende entsteht bei erfolgreicher Feinstrukturierung eine Baumstruktur mit einer Tiefe von drei. Falls in nach der Zuweisung in die Überkategorien nicht mehr als 3 Materialien in der gleichen Kategorie sind, wird nicht weiter fein-strukturiert.

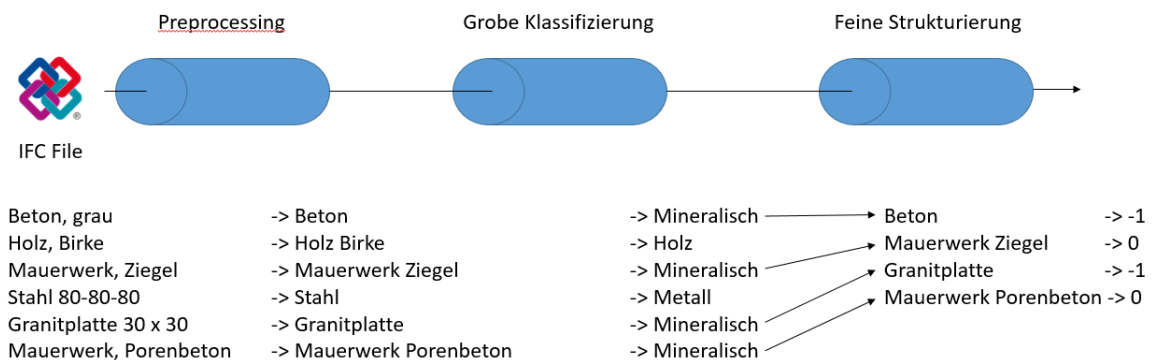


Abbildung 4.3: Ablauf der Strukturierung mit Beispiel

Bei der Strukturierung von Materialien gibt es einige Probleme, die gelöst werden müssen.

Problemstellung 3. *Bei den Materialangaben handelt es sich um Sehr kurze Stichpunkte aus nur einem oder wenigen Wörtern. (z.B Metall, Edelstahl - gebürstet)*

Problemstellung 3 ist ein bekanntes aber noch wenig erforschtes Problem in der Textklassifizierung. Textklassifizierungen stützen sich meistens auf Textrepräsentationen. Diese machen den Text für den Klassifizierungs-Algorithmus erst lesbar. Bei der Repräsentation von Text haben kurze Texte folgende Nachteile:

- Kurzen Texten fehlt es an Kontext. Kontextbezogene Word Embeddings wie z.B. Word2Vec führen folgendermaßen zu unbrauchbaren Textrepräsentationen für die Textklassifizierung.
- Kurze Texte entsprechen nicht immer der Syntax der natürlichen Sprache
- Kurze Texte sind in der Regel mehrdeutig, weil sie Polyseme und Tippfehler enthalten. (vgl. Wang et al., 2017)

Um das Problem zu überwinden, schlagen Wang et al. (2017) und Chen et al. (2019) vor, mehr Syntax und Semantik aus den kurzen Texten zu erfassen, in dem man mit Hilfe einer bekannten Wissensbasis mit Merkmalen anreichert. Die Informationen aus dem Text alleine, reichen nicht aus um die Bedeutung des Textes in einer Textrepräsentation gut darzustellen. Bie (2019) nutzt unter anderem Wikipedia2Vec, um eine Textrepräsentation der Kurzttexte zu bekommen. Durch den mitgegeben Kontext über das Wikipedia2Vec Modell wird so dem Wort seine Bedeutung der Textklassifizierung mitgegeben.

Problemstellung 4. *In den Materialangaben kommen viele Fachbegriffe aus der Baubranche vor.*

Ein weiteres Problem ist Problemstellung 4. Laut Nooralahzadeh et al. (2018) gibt es viele vortrainierte Embedding Modelle, die sich aber auf den allgemeinen Bereich der Sprache fokussieren. Diese können Fachbegriffe oft schlecht verarbeiten, da die zum Trainieren genutzten Texte, diese Wörter nicht beinhalten.

Wie diese Probleme gelöst beziehungsweise umgangen werden, wird in den jeweiligen Abschnitten der verschiedenen analysierten Algorithmen in Abschnitt 4.2 und Abschnitt 4.3 erläutert.

4.1.3 Konzept für das Auslesen von Materialien aus einer IFC Datei

Bevor jegliche Klassifizierung und Strukturierung stattfinden kann müssen die Materialien erst aus der IFC-Datei extrahiert werden. In Unterabschnitt 2.2.4 wurden bereits die theoretischen Möglichkeiten der Materialangabe in der IFC-Spezifikation aufgezählt.

Über schon vorhandenen Abstraktion des *xbim-toolkit* (siehe Unterabschnitt 2.2.3) kann eine IFC-Datei eingelesen werden. Über das instanziierte Objekt `IfcProject` kann dann auf alle Daten der Datei zugegriffen werden. Hier wird ein neuer `MaterialsManager` als Attribut des `IfcProject` implementiert. Über die Methode `Enumerable<Material> GetMergedMaterials()` im `MaterialsManager` können alle Materialangaben aus einer IFC-Datei ausgelesen werden. Die Methode iteriert über jeden bekannten Materiallink mit seinem jeweiligen Objekt, teilt diese Links nach ihrem jeweiligen Materialtyp (z.B. `LayerSet`) auf und liest die Materialbezeichnungen entsprechend dem Materialtypen aus. Am Ende liefert die Methode also alle Materialbezeichnungen in einer List mit der jeweiligen Globally Unique Identifier (GUID) des Bauteiles. Das Extrahieren der Materialien passiert Clientseitig in der ORCA AVA. So muss dem Service nicht eine ganze Datei, sondern lediglich die Liste der Materialstrings übergeben werden. Zusätzlich spart man sich die Zeit des Einlesen der Datei, wenn diese schon im IFC-Manager geöffnet ist.

4.1.4 Preprocessing der Materialien

Nach dem Einlesen beginnt das Preprocessing der Materialien. Das Preprocessing ist ein wichtiger Schritt vor einer Klassifizierung. Es hat Auswirkungen auf die Genauigkeit, Interpretierbarkeit und Robustheit des Gesamtalgorithmus aus. (vgl. Gonzalez and Carlos, 2019) Materialbezeichnungen aus IFC-Dateien können verrauscht und uneinheitlich sein, da es sich um ein Freitextfeld handelt. Die Vorverarbeitung der Daten hilft, die Daten zu säubern und auf das wesentliche reduzieren. (vgl. Chandrasekar and Qian, 2016) Die Vorverarbeitungsstufe besteht normalerweise aus Aufgaben wie Tokenisierung, Entfernung von Stoppwörtern, Umwandlung in Kleinbuchstaben und Stemming. (vgl. Uysal and Gunal, 2014) Für die Material-Strukturierung wird das Preprocessing als erster Schritt aus der Textklassifizierung extrahiert. Somit wirkt sich das Preprocessing auf beide weiteren Schritte (Grobe Textklassifizierung und Feinstrukturierung) aus. Außerdem wird das Preprocessing zusätzlich für die Zuweisung der Materialkostengliederung bei der Mengenübernahme als erster Schritt gebraucht. Konkret wird beim Preprocessing folgende Schritte durchgeführt. Als erstes werden alle Sonderzeichen bis auf „/“ entfernt. Oft befinden sich in der Bezeichnung auch Größen- und Farbangaben, welche irrelevant

für die Materialbeschaffenheit sind. Mithilfe von Regex werden Farben, RGB-Farbcodes (z.B. *60 - 60 - 60*) und Größenangaben (z.B. *400 x 300*) herausgefiltert. Anschließend wird der Text in Worttokens, jeder Token in Kleinbuchstaben umgewandelt und unnötige Leerzeichen entfernt. Aus „*Kunststoff - grau 80-80-80*“ wird z.B. „*'kunststoff'*“ und aus „*Beton- C30/37 Verputzt*“ wird „*'beton', 'c30/37', 'verputzt'*“

4.2 Algorithmen für die Textklassifizierung

Die Textklassifizierung ist nach dem Preprocessing der zweite Schritt der Materialstrukturierung. Hier werden die Materialien, den in Abbildung A.3 aufgelisteten Überkategorien, zugeordnet. Technisch wird dieser Schritt mit der .NET-Bibliothek *ML.NET* implementiert. Das Nuget-Package ist für die Erstellung benutzerdefinierter Modelle für maschinelles Lernen. (vgl. Microsoft, 2022b) Da die Bibliothek die Anforderungen für die Textklassifizierung gut erfüllen kann, kann die technische Anforderung 4 für diesen Schritt eingehalten werden. Die Bibliothek kann direkt in den ASP.NET-Service integriert werden. In diesem Kapitel wird zuerst der Ablauf einer normalen Textklassifizierung erläutert, die Kriterien der Messbarkeit festgelegt und dann die verschiedenen Algorithmen von *ML.NET* analysiert.

4.2.1 Ablauf einer Textklassifizierung

Das Klassifizieren von Texten teilt sich oft in die folgende Stufen. Alle diese Stufen wirken sich auf das Ergebnis der Klassifizierung aus:

Preprocessing

Das in Unterabschnitt 4.1.4 beschriebene Preprocessing findet als erster Schritt der ganzen Materialstrukturierung statt. Ein zusätzliches Preprocessing für die Textklassifizierung gibt es nicht.

Feature Selection

Beim Schritt der Feature Selection werden unrelevante Informationen für die Klassifizierung aus den Ausgangsdaten entfernt. Da es bei der Materialstrukturierung nur die Materialbezeichnung als Feature gibt, muss keine Auswahl aus einem Feature-Set getroffen werden. Es werden lediglich Größen- und Farbangaben aus dem Material-String im Preprocessing entfernt.

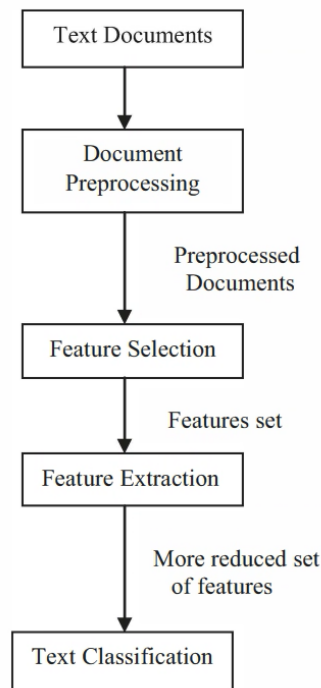


Abbildung 4.4: Stufen der Textklassifizierung (Quelle: Shah and Patel (2016))

Feature Extraction

Definition 3 (Feature extraction). „*Feature extraction addresses the problem of finding the most compact and informative set of features, to improve the efficiency or data storage and processing.*“ (Guyon and Elisseeff, 2006)

Bei der Feature Extraction geht es darum, die rohen Daten in eine numerische Darstellung zu bringen, die möglichst kompakt ist, ohne aber relevanten Informationen zu verlieren. Auch die im Preprocessing entstandenen Worttokens sind noch Zeichenketten die in eine numerische Darstellung umgewandelt werden müssen.

Classification stages

Nachdem die vorherigen Schritte auf die Daten ausgeführt wurde, kann das Training für die Klassifizierung durchgeführt werden.

4.2.2 Kriterien der Messbarkeit

Für die Textklassifizierung sind folgende Kriterien der Messbarkeit wichtig:

Genauigkeit

Die Genauigkeit bei maschinellem Lernen misst die Wirksamkeit eines Modells. Klassifizierte Daten werden in einen Testdatensatz und einen Trainingsdatensatz geteilt. Mit dem Testdatensatz wird das trainierte Modell getestet und als Ergebnis der Genauigkeit hat man eine prozentuale Angabe der Übereinstimmung des Testdatensatzes. (vgl. Microsoft, 2022a)

Interpretierbarkeit

Definition 4 (Interpretierbarkeit). „*Interpretability is the degree to which a human can understand the cause of a decision.*“ (Miller, 2017)

Hier geht es darum, die Ergebnisse des Algorithmus nachzuvollziehen zu können. Alleine die Genauigkeit reicht nicht dazu aus, einem Machine-Learning-Algorithmus vertrauen zu können. Ein Modell ist besser interpretierbar als ein anderes Modell, wenn seine Entscheidungen für einen Menschen leichter nachvollziehbar sind als die Entscheidungen des anderen Modells. Die Interpretierbarkeit kann man nicht direkt messen. Sie entsteht aus der Erfahrung über den Algorithmus und die Nachvollziehbarkeit der vorhergesagten Ergebnisse durch Testen. (vgl. Molnar, 2022) Das Endergebnis der Textklassifizierung aus verschiedenen IFC-Dateien, werden fachlich vom Produktmanagement bewertet. Diese haben das fachliche Wissen, um das Ergebnis bewerten zu können.

Robustheit

Von Deep-Learning-Algorithmen erwartet man, dass sie robust gegen kleine Störungen in der Eingabe sind. Es wurde allerdings schon festgestellt, dass kleine Störungen teilweise das Ergebnis ändern können. (vgl. Szegedy et al., 2013) Im Bezug mit auf die Materialeingabe sollen zum Beispiel verschiedene Satzzeichen keinen Einfluss auf das Ergebnis haben. Die Materialbezeichnung „Holz, Birke“ und „Holz - Birke“ sollen mit dem Algorithmus auf das gleiche Ergebnis führen.

Performance

Aspekte wie Geschwindigkeit und Ressourcenbedarf spielen eine sekundäre Rolle. Wenn es den Anwendungsfluss allerdings sehr beeinträchtigt und verlangsamt, scheidet er als Option aus. Zum Punkt Performance zählt auch die Trainingsdauer des Algorithmus. Diese kann beim Vergleich von verschiedenen Algorithmen helfen.

Konkret für die Textklassifizierung gilt als Kriterien der Messbarkeit die Genauigkeit, und sekundär die Ausführungsgeschwindigkeit und Trainingsdauer. Der Trainingsdatensatz wird aufgeteilt und 80% werden für das Trainieren und 20% für die Evaluation genutzt. Die benötigten Werte werden von *ML.NET* geliefert und können für die Auswertung verwendet werden. Bei der Genauigkeit ist vor allem die *MicroAccuracy* entscheidend, da sie den Durchschnitt der Datensätze aller Klassen berechnet. Aufgrund des Klassenungleichgewichts gibt die *MacroAccuracy* eine leicht verzerrte Genauigkeit zurück, da sie nur die durchschnittliche Genauigkeit der einzelnen Klassen berechnet. Für die Genauigkeit wird auch der *LogLoss* in Betracht gezogen. Zusätzlich wird eine Konfusionsmatrix erstellt, welche die Anzahl der vorhergesagten Klassen im Vergleich zu den tatsächlichen Klassen wiedergibt. (vgl. Microsoft, n.d.c) Die Trainingsdauer wird mit Hilfe der *Stopwatch*-Klasse im Code gemessen. Der Service wird dabei lokal auf einem Rechner mit einem Intel Core i7-7700 Prozessor mit einer integrierten Intel HD Graphics 6300 Grafikeinheit im Releasemodus ausgeführt. Zusätzlich wird die Interpretierbarkeit und die Robustheit durch die gesammelten Erfahrungen mit den Algorithmen widerspiegelt, können allerdings nicht wissenschaftlich belegt werden.

4.2.3 ML.NET Multiclass Classification

ML.NET bietet verschiedene *Trainer*-Klassen für das trainieren von verschiedenen Machine-Learning Modellen. Diese setzen sich immer aus einem Algorithmus und einem „Task“ zusammen. Der Algorithmus ist die Mathematik, die ausgeführt wird, um ein Modell zu erstellen. Dieser Algorithmus kann für verschiedene Aufgaben angewendet werden. Die Klassifizierung der Materialien fällt unter die Multiclass Classification, da es mehr als zwei Überkategorien gibt. Je nach Aufgabe hat das vom *Trainer* trainierte Modell, eine andere Ausgabe. (vgl. Microsoft, 2021) Für die Multiclass Classification gibt es in ML.NET verschiedene Vorgehensweisen. Diese werden in den folgenden Abschnitten behandelt.

Maximum Entropy Model

Einige Algorithmen benutzen Maximum Entropy Modeling um ein Modell mit mehreren Klassen zu generieren. Die Entropie kann man als Maß für den Grad der Unsicherheit, bzw der Informationsdichte einer Information bezeichnen. (??). Berger et al. (1996) beschreiben die maximale Entropie mit folgendem Prinzip: Die maximale Entropie entsteht bei einem möglichst einheitlich und gleich-verteilten Modell, das alle bekannten Daten modelliert, allerdings nichts zusätzlich, über noch unbekannte Daten, annimmt.

Ein Maximum Entropy Model ist ein stochastisches Modell zur Schätzung der bedingten Wahrscheinlichkeit, dass der Prozess bei einem Kontext x , y ausgibt. Unter allen Modellen ist das Modell mit der größten Entropie zu wählen, das mit den Beschränkungen der Trainingsdaten vereinbar ist. Das Modell ist somit möglichst einheitlich und unvoreingenommen (vgl. Berger et al., 1996).

Definition 5 (Entropie). „the entropy, H , of a discrete random variable X is a measure of the amount of uncertainty associated with the value of X “ (Shannon (1948) cited in Adriaans (2020))

Um ein solches Modell zu bekommen wird eine Optimierungsalgorithmus benötigt. Für jede Nebenbedingung hat das Modell der maximalen Entropie einen einstellbaren Parameter. Die optimalen Werte dieser Parameter werden durch Maximierung der maximalen Entropie der Trainingsdaten ermittelt. (vgl. Berger et al., 1996)

Die Bibliothek *ML.NET* bietet zwei verschiedene Trainer mit zwei verschiedenen Optimierungsmethoden an, welche auf dem Maximum Entropy Modell aufbauen. Der `SdcaMaximumEntropyMulticlassTrainer` basiert auf dem Stochastic dual coordinated ascent (SDCA) Optimierungsalgorithmus, welcher von Shalev-Shwartz and Zhang (2013) veröffentlicht wurde. SDCA minimiert die *Maximum Entropy*-Verlustfunktion während des Trainings. Dabei iteriert, der Algorithmus über die Trainingsdatensätze und aktualisiert die Gewichte des Modells. SDCA zeichnet sich vor allem durch seine Performance aus, da er immer nur einzelne Gewichte des Modells anpasst. (vgl. Shalev-Shwartz and Zhang, 2013; Microsoft, n.d.e)

Der `LbfgsMaximumEntropyMulticlassTrainer` ist ein weiterer Trainer, der auf das Maximum Entropy Modell basiert. Hier wird der Optimierungsalgorithmus limited memory Broyden-Fletcher-Goldfarb-Shanno method (L-BFGS) verwendet. Dieser wurde 1989 von Liu and Nocedal veröffentlicht und gehört zur Familie der quasi-Newton Methoden. Durch den optimierten Speicherverbrauch ist der Optimierungsalgorithmus vor allem bei großer Anzahl von Merkmalen oder einem hochdimensionalen Merkmalsvektor geeignet. (vgl. Microsoft, n.d.a)

TextclassificationTrainer

2022 wurde mit der *ML.NET*-Version 2.0 eine Textklassifizierungs Application Programming Interface (API) veröffentlicht. Quintanilla (2022) Dieser Algorithmus benutzt Deep Learning Techniken und arbeitet demnach direkt mit Text als Eingabe. Der Algorithmus basiert auf einem Transformer (??) welche schon einen internen Encoder besitzt, der sich um das vektorisieren der Eingabe kümmert. Da Transformer aufwändig zu trainieren

sind, wird ein vortrainiertes NAS-BERT Modell, mit eigenen Trainingsdaten nur noch verfeinert. (vgl. Quintanilla, 2022) *ML.NET* abstrahiert mit dieser Textklassifizierung die Implementierung von *TorchSharp*, was eine .NET Bibliothek für das Trainieren von eigenen Neuronalen Netzen ist. (vgl. Quintanilla, 2022)

One Versus All Ansatz

Der `OneVersusAllTrainer` ist eine weitere von *ML.NET* bereitgestellte Vorgehensweise für die Multiclass Classification. Bei diesem wird für jede Klasse ein binärer Klassifizierungsalgorithmus trainiert, welcher die Klasse von allen anderen unterscheidet. Bei Vorhersagen werden dann alle binären Klassifizierer ausgeführt und der mit der höchsten Wertung ausgewählt. Der binäre Klassifizierungsalgorithmus kann frei gewählt werden und liefert einen Klassifizierer, der bei der Vorhersage einen Boolean als Label zurückgibt. (vgl. Microsoft, n.d.b) *ML.NET* bietet verschiedene binäre Klassifizierer an.

Im Rahmen dieser Arbeit, wird der One-Versus-All Ansatz nur mit dem `SdcaLogisticRegressionBinaryTrainer` analysiert. Es ist der passendste binäre Klassifizierer ausgewählt, analysiert und getestet. Dieser nutzt den schon beschriebenen Optimierungsalgorithmus SDCA und basiert auf logistischer Regression. (vgl. Microsoft, n.d.d)

4.3 Möglichkeiten für die Feinstrukturierung

Die Feinstrukturierung ist nach der Textklassifikation der dritte Schritt der Material-Strukturierung. Hier werden die in eine Überkategorie zugeordneten Materialien weiter hierarchisch strukturiert, um eine feinere und tiefere Baumstruktur zu erreichen. In diesem Kapitel wird das Vorgehen mit dem GPT-Modell von OpenAI und der Implementierung mit *fastText* Encoding und Density Based Clustering analysiert.

4.3.1 Spezifikation und Kriterien der Messbarkeit

Die Verteilung der Materialien auf die Überkategorien ist nicht gleich verteilt. Materialien der Kategorien *Mineralisch* oder *Metall* kommen viel häufiger vor als zum Beispiel *BitumenUndTeer* oder *Gebäudetechnik*. Nicht in jeder IFC-Datei ist eine Einfahrt aus Asphalt oder die Gebäudetechnik definiert. Die Verteilung der Materialien ist in Tabelle 4.1 zu sehen. Somit ist auch die Verteilung der Materialien aus einer IFC-Datei nach dem Schritt der Textklassifizierung in die Überkategorien nicht gleich verteilt. Eine Feinstrukturierung ist bei zwei Materialien pro Kategorie nicht sinnvoll und auch bei drei

soll noch keine tiefere Strukturierung stattfinden. Erst bei vier Materialien pro Kategorie wird feiner strukturiert. Die Feinstrukturierung soll keine weiteren selbst trainierten Klassifizierungsmodelle benötigen. Diese würden zusätzlich zu der Textklassifizierung hohen Wartungsaufwand verursachen, da sie regelmäßig aktualisiert werden müssten, um die Genauigkeit der Klassifizierung beizubehalten bzw. zu verbessern. In den folgenden Kapiteln wird dies einerseits mit dem Benutzen eines fertig trainierten Modells und Unsupervised Learning umgangen.

Somit kann die Feinstrukturierung nicht mit fest definierten Kriterien der Algorithmen evaluiert werden. Für die Auswertung sind 20 Beispiele zum Testen der Algorithmen definiert. Diese sind in Tabelle A.5 bis Tabelle A.26 zu finden. Die Beispiele wurden in Kooperation mit dem Produktmanagement erstellt und fachlich überprüft. Sie sind auf Basis echter IFC-Dateien entstanden. Eine Tabelle stellt alle Materialien einer IFC-Datei, die in die Überkategorie (in der Überschrift zu sehen) eingeordnet wurden, dar. In der Spalte *Klasse* sind die Begriffe einer Unterklasse über eine Nummer zugeordnet. Alle Begriffe mit der selben Nummer gehören zu einer Unterklasse. Die Verteilung der Beispiele über die Überkategorien ist aus dem selben Grund, wie auch die Verteilung der Materialklassifikationsdaten (Tabelle 4.1), nicht gleich verteilt. Die Ergebnisse der Evaluation sind in Abschnitt 5.2 aufgezeigt.

Kategorie	Anteil	Anzahl
Metall	25%	119
Glas	4%	17
Holz	9%	43
Kunststoff	8%	39
BitumenUndTeer	1%	3
Mineralisch	26%	127
Verbundbauteile	5%	22
Dämmungen	8%	39
GebäudeTechnik	1%	6
Sonstiges	14%	70
Gesamt	100%	485

Tabelle 4.1: Verteilung der Materialklassifikationsdaten

4.3.2 Generative Pretrained Transformer (GPT) Modell von OpenAI

OpenAI ist ein Unternehmen für KI-Forschung und -Einsatz. Sie bieten verschiedene fertig trainierte KI-Modelle über eine API an. Neben Speech-ToText und einem Bild-generierungsmodell gibt es das Sprachverarbeitungsmodell GPT. Die Nutzung eines

fertig trainierten Modells hat einige Vorteile. Vor allem werden so die Problemstellung 3 und Problemstellung 4 automatisch gelöst. OpenAI benutzen ihr eigenen ausgereiften Textrepräsentationsmodelle. Somit wird auch die Bedeutung von Worten bei den kurzen Materialangaben bei der Vektorisierung beibehalten. Außerdem kann das Modell durch die Masse an Trainingsdaten aus Büchern, Artikel und Webseiten auch mit vielen Fachbegriffen umgehen. Außerdem spart man sich die Wartung eines selbst trainierten Modells. Das Etikettieren von Daten ist sehr Zeit und auch Ressourcen-aufwändig, da es von einer Person gemacht werden muss, die passende fachliche Kenntnisse besitzt.

Der Begriff GPT steht für eine Reihe von vor trainierten Sprachmodellen, die von OpenAI entwickelt wurden. Diese sind populäre Transformer für die Generierung von natürlicher Sprache. (vgl. Zhu and Luo, 2022)

Definition 6 (Transformer). *„Transformers are a type of artificial neural network architecture that is used to solve the problem of transduction or transformation of input sequences into output sequences in deep learning applications.“ (Rogel-Salazar, 2022)*

GPT ist also ein Modell, das auf einem großen Datensatz an textuellen Informationen trainiert wurde und kann zur Bewältigung spezifischer sprachbezogener Aufgaben eingesetzt werden. Im Vergleich zu Modellen zum Verstehen natürlicher Sprache wie zum Beispiel BERT kann GPT natürliche Sprache generieren. Der Algorithmus liefert basierend auf allen eingegebenen Tokens neue Tokens, welche dann die Antwort darstellen. (vgl. Zhu and Luo, 2022)

OpenAI hat verschiedene Versionen des GPT Algorithmus. GPT 1 wurde 2018 veröffentlicht. Darauf folgte GPT 2 2019. Beide Modelle wurden in zwei Schritten trainiert. Zuerst wurde ein Model mit Hilfe von Unsupervised Learning mit einer sehr großen Datenmenge vor trainiert und konnte dann für bestimmte Aufgaben mit dem Trainieren von überwachten Datensätzen verfeinert werden. (vgl. Radford, 2018) GPT 3 wurde in einem Schritt trainiert und verzichtet auf die kontextbezogene Verfeinern des Modells. (vgl. Zhu and Luo, 2022) Stattdessen benutzt das Modell „Meta-Learning“. Das Modell entwickelt so beim Training ein breites Spektrum an Mustererkennungsfähigkeiten und weiteren Fertigkeiten, die sich beim Ausführen des Modells schnell an die gewünschte Aufgabe anpassen. Durch die Skalierung des Modells auf 175 Billionen Parameter wurden ähnliche Ergebnisse wie bei damaligen modernster fein abgestimmter Systeme erzielt. (vgl. Brown et al., 2020)

Die GPT-Modelle 2 und 3 sind nicht darauf trainiert, Benutzeranweisungen zu befolgen. Dafür wurde 2022 *InstructGPT* auf der Basis des GPT 3 Modell trainiert. Mit Supervised Learning und Reinforcement Learning wurde das Modell verfeinert, um hilfreichere

Ergebnisse bei Benutzeranweisungen zu bekommen. Zusätzlich wurde eine Reduktion von toxischen Antworten erzielt. (vgl. Ouyang et al., 2022)

Das für die Material-Feinstrukturierung verwendete Modell *ChatGPT* ist ein Zwillingsmodell von *InstructGPT* und ist eine Feinabstimmung eines Modells der GPT-3.5-Serie. (vgl. OpenAI, 2022) Das Modell wurde Ende 2022 veröffentlicht und ist über eine öffentliche API erreichbar und kann mit Hilfe des Python-Package *openai* leicht genutzt werden. Als Parameter wird zusätzlich die *temperature* mit dem Wert 0 angegeben. Das bedeutet, dass das Ergebnis möglichst deterministisch ist. (vgl. OpenAI, n.d.a) Die Nutzung kostet 0.002\$ pro 1000 Tokens. (vgl. OpenAI, n.d.b) Ein Request kostet also je nach Eingabe und Antwortlänge bei 150 bis 450 Tokens circa 0,0003\$ bis 0,0009\$.

4.3.3 fastText mit Density-Based Spatial Clustering of Applications with Noise (DBSCAN)

In diesem Ansatz werden mit Hilfe von Clustering, Materialien weiter strukturiert. Durch das Nutzen von Unsupervised Learning müssen so keine weiteren Klassifizierungs-Modelle trainiert und verwaltet werden. Um Problemstellung 3 zu lösen, wird das Vorgehen wie es auch Bie (2019) vorgeschlagen hat angewandt. Ein Textrepräsentationsmodell wird mit anderen Texten trainiert und mit diesem Modell die Vektorisierung auf die Materialbezeichnungen angewandt. Die „ORCA Software GmbH“ entwickelt neben der ORCA AVA das Portal ADE. Dieses hat über einer Millionen Ausschreibungstext von über 800 verschiedenen Herstellern. Diese Texte eignen sich optimal für das Trainieren des Worteinbettung Modells. Sie enthalten viele Materialnamen und weitere Fachbegriffe der Baubranche, welche essenziell für das Clustering der Materialien aus dem IFC-Dateien sind. Somit ist zusätzlich Problemstellung 4 gelöst. Auch Nooralahzadeh et al. (2018) kommt auf das Ergebnis, dass sich Domänen-Spezifische Word Embeddings auch schon mit limitierter Eingabedaten nützlich ist. Die Ausschreibungstexte werden mit einem schon existierendem Preprocessing verarbeitet und liefern am Ende ca. 360.000 Sätze mit über 300.000 verschiedene Wörtern. Im Preprocessing werden verschiedene Zeichenketten aus den Texten mit Hilfe von Regex herausgefiltert. Dazu gehören Uniform Resource Locator (URL)s, verschieden Normen wie DIN und Attribut-Listen wie zum Beispiel *Nettogewicht: 0,196 kg*. Zusätzlich werden unnötige Absätze entfernt. Auch wenn Ausschreibungstexte oft keine kompletten Sätze beinhalten, werden so viele nicht relevanten Daten entfernt.

Als Textrepräsentation und Wort-Einbettungs-Algorithmus bietet sich *fastText* an. *fastText* wurde 2016 von Facebook veröffentlicht (vgl. Bojanowski et al., 2016). Außerdem

wurde die dazu getätigte Forschung mit Bojanowski et al. (2017) und Joulin et al. (2016) veröffentlicht.

Das Modell basiert auf dem Wort-Einbettungs Algorithmus *Word2Vec* mit „Continuous Skip-Gram“ und Continuous Bag Of Words (CBOW) (vgl. Bojanowski et al., 2017). Das Modell nutzt ein Neuronales Netz bei dem die ausgewählten Gewichte als Vektor für ein Wort entstehen. Die Trainingsdaten für das neuronale Netz entstehen durch ein selbst gestelltes Problem. Mit CBOW versucht man mit nahegelegenen Wörtern ein Wort hervorzusagen. Bei Skip-Gram versucht man aus einem Wort den Kontext, also die nahegelegenden Wörter hervorzusagen. So kann beim Iterieren über den Trainingstext das Modell immer weiter trainiert werden. In Abbildung 4.5 sieht man ein CBOW Beispiel, in dem die Kontextwörter als Input Layer und das gesuchte Wort als Ergebnis zur Ermittlung des Verlustes genutzt werden. Die nach dem Trainieren entstandenen Gewichte $W1_{n \times k}$ sind dann die Repräsentation für das Wort *catches*. *fastText* erweitert

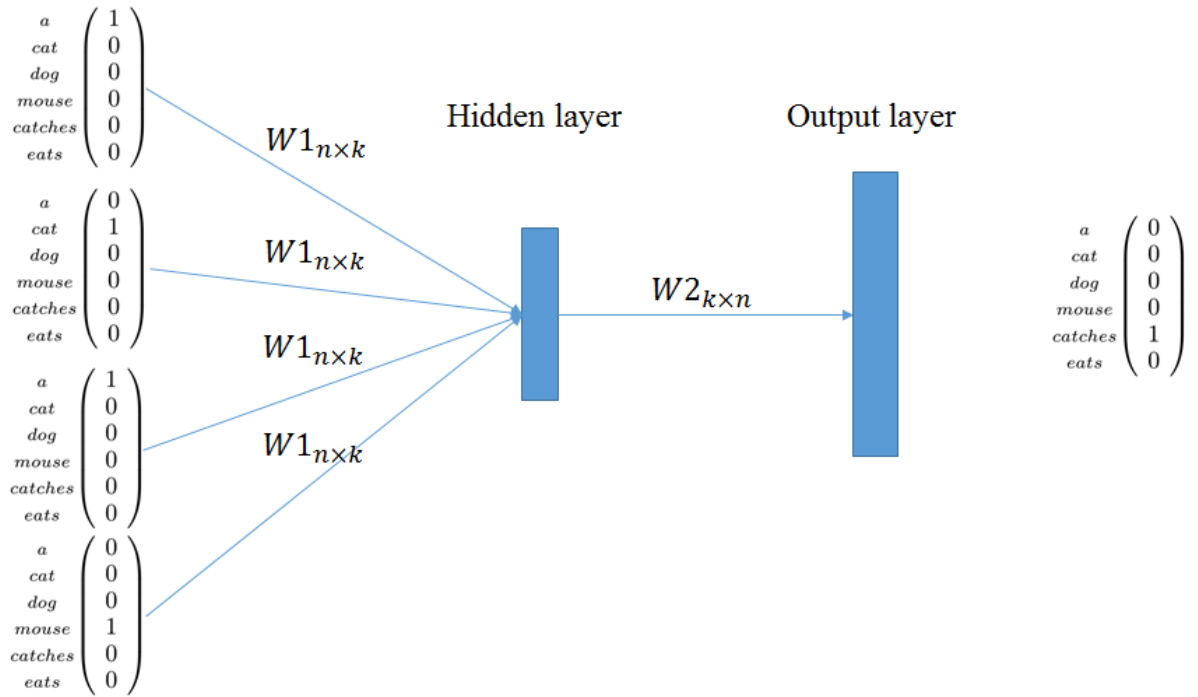


Abbildung 4.5: Beispiel des CBOW-Modells. (Quelle: Ferrone and Zanzotto (2019))

diese Vorgehensweise und, nutzt *fastText* Teilwortinformationen und ein Wort als Vektor darzustellen. Wörter werden in N-Grams dargestellt. (vgl. Bojanowski et al., 2017) Die Repräsentation für das Wort *Estrich* mit $n = 3$ ist zum Beispiel $\langle Es, est, str, tri, ric, ich, ch \rangle$ wobei die Zeichen \langle und \rangle als Begrenzungszeichen hinzugefügt werden, um die N-Gramms eines Wortes vom Wort selbst zu unterscheiden. Wäre also das Wort *ich* im Vokabular, wird es als $\langle ich \rangle$ dargestellt. Das Neuronale Netz wird dann mit der Liste

der N-Gramms inklusive dem kompletten Wort trainiert. Somit ermöglicht das Modell die gemeinsame Nutzung der Teilwortinformationen für alle Wörter (vgl. Bojanowski et al., 2017). Zusätzlich nutzt *fastText* eine hierarchische Softmax Verlustfunktion, um die Berechnung von unausgewogenen Verteilungen zu beschleunigen (vgl. Bojanowski et al., 2016).

fastText hat für den Anwendungsfall folgende Vorteile:

- *fastText* kann mit Wörtern außerhalb des Vokabulars umgehen. Bei unbekannten Wörtern, werden die Vektoren der einzelnen Char-Ngramms summiert. Es muss allerdings mindestens eines der char-ngrams in den Trainingsdaten vorhanden sein. (vgl. Gensim Documentation, n.d.; Le and Mikolov, 2014) Da sich bei diesem Konzept der Feinstrukturierung das Vokabular bei Ausführung vom Trainingsvokabular unterscheidet, ist das Verarbeiten von diesen Wörtern essentiell.
- *fastText* liefert Vektordarstellungen mit fester Länge welche für das Clustering mit zum Beispiel DBSCAN nötig ist. (vgl. Le and Mikolov, 2014)
- *fastText* liefert bessere Ergebnisse bei syntaktischen Aufgaben als zum Beispiel *Word2Vec* und *Bag of Words* (vgl. Gensim, n.d.; Le and Mikolov, 2014)
- *fastText* hat eine schnellere Trainingszeit als vergleichbar genaue Deep Learning Algorithmen (vgl. Bojanowski et al., 2016)

Mit dem fertige trainierten *fastText* Modell können dann Wörter in Vektoren umgewandelt werden. Bei der Vektorisierung der Materialien, die aus mehreren Wörtern bestehen, werden die Vektoren der einzelnen Wörter aufaddiert. Das ist die selbe Vorgehensweise, wie *fastText* bei neuen Wörter, die nicht im Vokabular vorkommen, vorgeht (vgl. Gensim Documentation, n.d.).

Wenn die Materialbezeichnungen vektorisiert wurden können sie geclustert werden. Aus einer großen Auswahl an Cluster-Algorithmen bietet sich DBSCAN an. „The DBSCAN algorithm views clusters as areas of high density separated by areas of low density.“ (vgl. scikit-learn developers, n.d.) Der Algorithmus wurde ? von ? vorgestellt und veröffentlicht. Ziel war es einen Algorithmus zu entwickeln, bei dem man wenig Fachwissen für die Bestimmung der Parameter braucht, Cluster mit beliebiger Form bekommen kann und eine gute Effizienz bei großen Datenbanken besitzt.

DBSCAN braucht die zwei Parameter *min_samples* und *eps*. Als erster Schritt werden alle Punkte, die inklusive sich selbst mindestens *min_samples* an Nachbarn innerhalb des Abstands von *eps* hat, als Kernpunkte definiert. Alle anderen Punkte werden erstmals

als Rauschen gekennzeichnet. Als nächstes werden alle Kernpunkte, die innerhalb des Radius eps beieinander liegen als ein Cluster definiert. Zum Schluss werden nochmal alle Rauschpunkte überprüft, ob sie nicht doch innerhalb des Radius eines Kernpunktes in einem Clusters liegen. (vgl. scikit-learn developers, n.d.; ?) Die Anzahl der Cluster ist also abhängig von den Daten (und den Parametern) und muss nicht vorgegeben werden. DBSCAN hat also für den Anwendungsfall folgende Vorteile:

- DBSCAN liefert eine passende Anzahl von Clustern, abhängig von den Materialbezeichnungen. Dies ist wichtig, da die Anzahl der Materialien pro Überkategorie und somit auch die der potentiellen Clustern immer unterschiedlich ist. Das Ergebnis des Clustering wird auch direkt dem Benutzer der ORCA AVA präsentiert. Somit ist es auch nicht möglich mit einem anderen Clustering-Algorithmus über die Anzahl von Clustern zu iterieren und das fachlich beste auszuwählen.
- Es müssen keine Trainingsdatensätze oder Modell verwaltet werden.
- Der Algorithmus setzt die mitgegebene Bedeutung des Wortes bei der Wort-Einbettung mit *fastText* um.

Am Ende liefert DBSCAN zu jedem übergebenem Material eine Zahl, die das Cluster repräsentiert. Rauschpunkte werden mit -1 zurückgegeben. Diese werden dann nicht weiter strukturiert und bleiben hierarchisch direkt unter ihrer zugewiesenen Überkategorie. Dieses Format kann für die Evaluationsbeispiele (Tabelle A.5 - Tabelle A.26) direkt genutzt werden. Bei diesem Ergebnis fehlen für die endgültige Material-Strukturierung noch die Begriffe der Gliederungspunkte für die entstandenen Cluster.

5 Gegenüberstellung der möglichen Konzepte

In diesem Kapitel werden die in Kapitel 4 vorgestellten Algorithmen verglichen. Dies wird gesondert für die Textklassifizierung und die Feinstrukturierung durchgeführt. Es wird jeweils das Ergebnis ausgewertet und eine Auswahl eines Algorithmus durchgeführt.

5.1 Vergleich der Textklassifizierung

Für die Textklassifizierung wird der beste Algorithmus der *ML.NET*-Bibliothek gesucht. Zur Auswahl stehen das *Maximum Entropy Model* mit SDCA oder L-BFGS als Optimierungsalgorithmus, die preview Textklassifizierungsschnittstelle basierend auf einem NAS-BERT Transformer und der One-Versus-All Ansatz mit einem binären Klassifizierer basierend auf logistischer Regression mit dem SDCA Optimierungsalgorithmus.

5.1.1 Auswertung

Bei der Textklassifizierung werden die Algorithmen nach den in Unterabschnitt 4.2.2 erläuterten Kriterien der Messbarkeit ausgewertet. Die Ergebnisse sind in Tabelle 5.1 zu sehen. Zusätzlich befinden sich in Tabelle A.1 bis Tabelle A.4 die Konfusionsmatrix für jeden Algorithmus. Da nur 500 Datensätze zum trainieren zur Verfügung stehen, konnten nur 20% und somit 94 Testdatensätze definiert werden. Da diese von *ML.NET* zufällig ausgewählt werden, kann es zu Variationen der Ergebnisse kommen. Die in dieser Arbeit verwendeten Messdaten repräsentieren eine durchschnittliche Auswertung der Algorithmen.

Algorithmus	MicroAccuracy	MacroAccuracy	LogLoss	TrainingTime
SdcaMaximumEntropy	0,882	0,910	0,477	0,245sec
LbfgsMaximumEntropy	0,670	0,591	1,131	0,243sec
TextClassification	0,479	0,292	5,994	20,229sec
OneVersusAll SdcaLogisticRegression	0,872	0,894	0,477	4,219sec

Tabelle 5.1: Auswertung der Textklassifizierungsalgorithmen

Die Klassifizierungsalgorithmen liefern verschiedene Ergebnisse. Bei der Betrachtung der Genauigkeit schneidet *SdcaMaximumEntropy* mit einer Genauigkeit von 88,2% am besten ab, dicht verfolgt von *OneVersusAll* mit 87,2%. *LbfgsMaximumEntropy* und die *TextClassification* haben dagegen nur eine Genauigkeit von 67% bzw. 48%. In den Konfusionsmatrizen dieser beiden erkennt man, dass vor allem die Klassen *Mineralisch* und *Metall* mit einer schlechten Genauigkeit von 43% bzw. 35% abschneiden. Mineralische Materialien wurden hier zwar meistens (96%/87%) richtig zugeordnet. Materialien vieler anderer Klassen wurden aber auch diesen beiden Klassen zugewiesen. Bei *SdcaMaximumEntropy* und *OneVersusAll* ist das ähnlich zu beobachten. Mineralische Materialien werden bei beiden Modellen zu 92% richtig klassifiziert. Hier ist die Genauigkeit aller Mineralisch zugeordneten Materialien bei 76%. Der LogLoss ist bei diesen beiden Modellen auch mit 0,477 deutlich niedriger als bei den anderen Algorithmen.

Die Maximum Entropy Modelle scheiden bei der Trainingsdauer am besten ab. Mit ca. 0,24 Sekunden ist die Trainingszeit bei beiden Modellen zu vernachlässigen und kann theoretisch sogar nach jedem neu klassifizierten Datensatz durchgeführt werden. Da der *OneVersusAll* Algorithmus mehrere binäre Modelle trainieren muss, braucht er mit 4,2 Sekunden deutlich länger. Das *TextClassification* Modell braucht ganze 20 Sekunden um 406 Datensätze zu trainieren.

5.1.2 Auswahl eines Algorithmus

Die Wahl des Textklassifizierungsalgorithmus fällt eindeutig auf *SdcaMaximumEntropy*. Mit einer Genauigkeit von 88,2% wurde ein nutzbares Ergebnis erzielt. Zusätzlich ist es der Algorithmus mit der kürzesten Trainingsdauer. Auch wenn der *OneVersusAll* Ansatz auf eine ähnliche Genauigkeit kommt, dauert das Training deutlich länger und das gespeicherte Modell braucht mehr Speicher, als das gespeicherte *SdcaMaximumEntropy* Modell. Zusätzlich wurde das *SdcaMaximumEntropy* Modell dem Projektmanagement vorgestellt und fachlich evaluiert. Das Ergebnis wurde als fachlich brauchbar eingestuft und kann somit für die ORCA AVA genutzt und integriert werden.

5.2 Vergleich der Feinstrukturierung

Bei der Feinstrukturierung wird zwischen dem fertig trainierten GPT-Modell von OpenAi und einem selbst implementieren Vorgehensweise mit *fastText* und DBSCAN-Clustering verglichen.

5.2.1 Auswertung

Die Algorithmen für die Feinstrukturierung werden zuerst nach den Kriterien der Messbarkeit aus Unterabschnitt 4.3.1 ausgewertet. Das Ergebnis ist in Tabelle A.5 bis Tabelle A.26 eingetragen. -1 bedeutet, dass das Ergebnis nicht nutzbar ist, da zum Beispiel der Begriff an sich verändert wurde. Zusätzlich ist in Tabelle 5.2 eine akkumulierte Auswertung. Hier sind die Evaluationsbeispiele eingeteilt. *Richtig* heißt, dass das Ergebnis mit dem erwarteten Ergebnis übereinstimmt. *Teilweise Richtig* bedeutet, dass manche Materialien zwar nicht zugewiesen wurde, aber es keine fachlich falsche Zuweisung enthält. Dieses ist laut dem Produktmanagement als ausreichendes Ergebnis für die Feinstrukturierung zu sehen. Bei *Falsch* wurde ein oder mehrere Materialien falsch zugewiesen, oder das Ergebnis war nicht für die Weiterverarbeitung nutzbar.

	fastText & DBSCAN	OpenAI
Richtig	9	5
Teilweise Richtig	6	10
Falsch	7	7

Tabelle 5.2: Auswertung der Feinstrukturierung in *Richtig*, *Teilweise Richtig*, und *Falsch*

Vorteile von *fastText* & *DBSCAN*:

- Besseres Ergebnis bei der Auswertung der Evaluationsbeispiele (siehe Tabelle 5.2)
- deterministischer Algorithmus liefert für gleiche Eingabe immer das gleiche Ergebnis
- keine zusätzlichen Kosten
- Gute Interpretierbarkeit des Algorithmus

Nachteile von *fastText* & *DBSCAN*:

- Erhöhter Aufwand durch das Pflegen und Verbessern des selbst trainierten *fastText*-Modells
- Clustert teilweise nach irrelevanten Wörtern. (z.B.: matt, verputzt)

Vorteile von *ChatGPT*:

- Kein Wartungsaufwand von selbst trainierten Modellen.
- Liefert fachlich zum Großteil ein fachlich richtiges Ergebnis.

Nachteile von *ChatGPT*:

- Verändert Materialbezeichnungen, was das Ergebnis für als Kostengliederung nicht brauchbar macht. Aus *Stahl matt* wird zum Beispiel *1.1 Stahl* und *1.1.1 matt*
- Die Ausgabe ist trotz *Temperature* von 0 nicht deterministisch. Das senkt in Kombination mit der Veränderung von Begriffen die Interpretierbarkeit des Algorithmus.
- Form der Ausgabe als Zeichenkette und mögliche Variation in der Struktur der Ausgabe kann zu Fehlern beim Abbilden in die Objektstruktur führen.
- Zusätzlicher Kostenaufwand von 0.002\$ pro 1000 Tokens

5.2.2 Auswahl eines Algorithmus

Für die Feinstrukturierung kommt nur *fastText* & *DBSCAN* als einzige Option in Frage. Das fachliche Ergebnis der jeweiligen Strukturierung lässt beide Optionen für die Feinstrukturierung zu. *fastText* & *DBSCAN* scheiden bei den Evaluationsbeispielen aber auch schon besser ab. Die beiden Algorithmen unterscheiden sich vor allem in der Interpretierbarkeit (Definition 4). Die genannten Nachteile von *ChatGPT* führen schnell zu fehlender Interpretierbarkeit und somit zu Misstrauen gegenüber dem Feature. Vor allem die Kombination der einzelnen Nachteile, lässt manche Ergebnisse willkürlich erscheinen. Zusätzlich können mit *ChatGPT* oft kein Ergebnis zur Verfügung gestellt werden, da nach der Veränderung von Materialbezeichnungen die ursprünglichen Materialien nicht mehr nachvollzogen werden können. *fastText* & *DBSCAN* schneiden in der Interpretierbarkeit gut ab. Das entsteht durch deterministische Ergebnisse und nachvollziehbares Clustering. Das Zusammenfassen von mehreren Materialien in ein Cluster kann immer, auch bei einem fachlich falschen Ergebnis, nachvollzogen und eine Logik hinter der Strukturierung gefunden werden. Durch die bessere Interpretierbarkeit wird *fastText* & *DBSCAN* als Algorithmus für die Feinstrukturierung festgelegt.

6 Praktische Umsetzung

In diesem Kapitel wird die Umsetzung der ausgewählten Algorithmen und Bibliotheken beschrieben. Zuerst werden Die Implementierung des Services und der Python-Interop gezeigt. Danach geht es um die konkrete Implementierung der Material-Strukturierung.

6.1 Implementierung des ASP.NET Services

Die Realisierung als ASP.NET Service ist eine Anforderung der „ORCA Software GmbH“. Für das Erstellen eines neuen Services gibt es ein fertiges Projekttemplate, welches nur umbenannt werden muss. Die Vorlage liefert eine einheitliche Projektstruktur und zeigt den Aufbau des Services anhand der Implementierung eines Beispiels. Die neu genannte *MaterialStructurizationApi* hat zwei verschiedene REST-Controller. Diese teilen sich wie im Anwendungsfalldiagramm (Abbildung 4.1) zu sehen in die Schnittstellen für die ORCA AVA Benutzer und die ORCA Internen Schnittstellen auf. In den Controllern werden die REST-Schnittstellen-Methoden definiert. Dazu zählt die Art der Anfragemethoden, die URL oder auch die möglichen Antwort-StatusCodes. Service interne Logik wird im *MaterialService* organisiert. Hier werden die Data Transfer Objects (DTOs) in Entitäten umgewandelt und die Aufrufe auf die entsprechenden weiteren themenspezifischen Services oder Datenzugriff-Repositories weitergeleitet. Am Ende werden Rückgabewerte wieder in entsprechende DTOs abgebildet und an den Controller zurückgegeben. Ein Beispiel ist in Codeauflistung A.1 zu sehen. Die Methode ist für die Material-Strukturierung zuständig. Sie bekommt die Liste von Manualbezeichnungen und eine `bool`, welcher bestimmt ob die Feingliederung durchgeführt werden soll, übergeben. Nach der Parameterprüfung werden Duplikate aus der Materialliste entfernt und danach in Zeile 8 über das *MaterialRepository* neue Materialien dem Trainingsdatensatz für die Klassifizierung hinzugefügt. Danach wird der *MaterialStructurizerService* aufgerufen, der sich dann um die Logik der Strukturierung kümmert.

Repositories kapseln den Datenzugriff über Interfaces. So ist die Logikimplementierung in den Services unabhängig von der Implementierung des Datenbankzugriffs. Der

Datenbankzugriff wird in den Repositories mit der Object-Relation-Mapper Bibliothek *EntityFramework* implementiert. Es wird der Model-First Ansatz verwendet, bei dem zuerst die Objektstruktur im C#-Code definiert wird und daraus automatisiert die Datenbank generiert wird.

6.2 Python-Interop

Für die Implementierung einiger Komponenten wird Python als Programmiersprache benötigt. Der Webservice wird allerdings mit ASP.NET implementiert. Um die Kommunikation zwischen .NET und Python zu vereinfachen wird das Nuget-Paket *pythonnet* verwendet. Die Bibliothek vereinfacht es Python-Code aus dem C#-Code auszuführen und verschiedene Python-Variablen auszulesen oder diesen Werte zuweisen. Um den Python-Interop nutzen zu können, muss eine lauffähige Python-Umgebung installiert sein. Der Python-Interop ist in mit dem Interface *IKiExecutor* gekapselt und lässt so eine eventuelle C# Ablösung der Python-Komponenten in der Zukunft einfach zu integrieren. In Codeauflistung 6.1 ist der Python-Code Aufruf für das Preprocessing zu sehen. Nachdem die Python-Engine initialisiert wird, kann der Python-Variable ein Wert zugewiesen werden. Dann wird das Skript *preprocess.py* (Codeauflistung A.3) über den Namen aufgerufen und danach die Python-Variable *material_preprocessed*, in welcher am Ende des Python-Skripts das Ergebnis steht, wieder ausgelesen.

```
1 PythonEngine.Initialize();
2
3 using (Py.GIL())
4 {
5     using (PyModule scope = Py.CreateScope())
6     {
7         scope.Set("material_raw", materials);
8         scope.Exec(File.ReadAllText(GetPythonScript("preprocess")));
9         string materialPreprocessed = scope.Get("material_preprocessed")
10             .As<string>();
11     }
```

Codeauflistung 6.1: Python-Interopt für das Preprocessing

Zusätzlich löst *pythonnet* das Problem, dass das Pythonskript zur Feinstrukturierung nicht immer wieder das *fastText*-Modell neu in den Speicher laden muss. Durch die Möglichkeit Python-Variablen zuzuweisen und auszulesen, ist es auch möglich das *fastText*-Modell initial beim Starten des Services in den Speicher zu laden. Das macht das

Ausführen der Feinstrukturierung sehr viel performanter und erst produktiv einsetzbar. Für die Entwicklung der Pyhtonskripte wird die Logik immer in eine Methode ausgelagert. Mit der Zeile `if __name__ == "__main__":` wird der darauf folgende Code beim initialen aufrufen des Skriptes ausgeführt. Mit dem Try-Except-Block wird überprüft, ob durch *pythonnet* die Eingabevariable schon initialisiert wurde. Ansonsten kann beim Debuggen ein Wert über die Kommandozeile eingetragen werden. Somit kann für alle Konstellationen immer nur ein Skript verwendet werden. Das ganze Skript sowohl die definierte Methode kann dann mit *pythonnet* im C#-Code ausgeführt und aber auch autark im Debug-Modus ausgeführt werden. (Beispiel: Codeauflistung A.3)

6.3 Implementieren des Auslesens aller Materialien aus einer IFC Datei

Für das Auslesen aller Materialien aus einer IFC-Datei besteht schon ein Grundgerüst in der Codebasis der ORCA AVA. Es bestehen bereits die Funktionen mit dem *xbim-toolkit* IFC-Dateien einzulesen und daraus das Business Model `IfcProject` zu erzeugen. Dazu reicht der Pfad zur gewünschten IFC-Datei.

6.4 Implementierung der Material-Strukturierung

Die Implementierung der Material-Strukturierung teilt sich ähnlich wie in Kapitel 4 auf. Zuerst geht es um das Erstellen einer Datengrundlage, dann um die Implementierung des Preprocessings. Anschließend wird das Textklassifizierungs-Modell vorgestellt. Für die Feinstrukturierung wird am Ende noch die Implementierung für *fastText* und DBSCAN erläutert.

6.4.1 Erstellen einer Datengrundlage

Die Datengrundlage entstand durch das Extrahieren der Materialangaben (wie in Abschnitt 6.3 beschrieben) aus verschiedenen IFC-Dateien. Hierzu standen über 500 verschiedene Dateien zur Verfügung. Die daraus entstandenen Materialbezeichnungen wurden über eine Hypertext Transfer Protocol (HTTP)-PUT Anfrage in einem Structured Query Language (SQL)-Server mit dem in Abbildung 4.2 abgebildetem Schema persistiert. Die Liste diese Materialien wurden anschließend manuell gefiltert. In vielen IFC-Dateien werden die Möglichkeiten der Materialangabe noch nicht gut genutzt. Deshalb waren viele Materialangaben nicht aussagekräftig um sie als Trainingsdaten

für die Textklassifizierung zu nutzen. Am Ende standen circa 500 unterschiedliche Materialien zur Verfügung. Um die Daten anschließend zu klassifizieren, wurde ein Klassifizierungs-Service mit Oberfläche implementiert (siehe Abbildung A.11). Dieser stand im internen Firmennetzwerk zur Verfügung, sodass ausgewählte Mitarbeiter die Materialien klassifizieren konnten. Die dafür benötigten Methoden standen über den `InhouseMaterialController` der *MaterialStructurizationApi*, welcher die internen ORCA-Schnittstellen bietet, zur Verfügung. Somit wurden über 480 klassifizierte Datensätze geschaffen. Die Verteilung der Klassifizierung ist in Abbildung A.7 zu sehen.

6.4.2 Implementierung des Preprocessings

Das Preprocessing für die Textklassifizierung wurde in Python implementiert. Das Vorgehen wurde schon in Unterabschnitt 4.1.4 erläutert. Die Implementierung des Pythonskripts ist in Codeauflistung A.2 zu sehen. Das Preprocessing basiert größtenteils auf der Textverarbeitung mit Regex. Mit dem Regex `[A-Za-z0-9üäöÜÄÖßóâéèÉÈÓ/]` werden alle Sonderzeichen bis auf „/“ aus dem String entfernt. Bei der Maßangabe mussten aufgrund der unterschiedlicher Verarbeitung des Python-Package `re` bei `re.findall(...)` `re.sub(...)` mehrere Regex definiert werden. Zuerst werden dreidimensionale Maße im Materialstring gesucht. Wenn keine Vorhanden sind, werden zweidimensionale Maße entfernt. Ansonsten wird mit einem zusätzlichen Regex die dreidimensionalen Maßangaben entfernt. Am Ende wird eine Liste von Strings aus dem Materialstring gefiltert und überschüssige Leerzeichen mit `strip()` entfernt.

Wie in Abbildung 4.2 zu sehen ist, werden die vorverarbeiteten Materialien in der Datenbank persistiert. Immer wenn ein neues Material in die Datenbank geschrieben wird, wird das Preprocessing gestartet. Das bietet die Möglichkeit direkt nach dem Klassifizieren eines Materials, das Textklassifizierungs-Modell ohne vorheriges Preprocessing neu trainieren zu können. Dadurch verringert sich die Ausführungszeit des Trainingsprozesses und es kann direkt auf das aktualisierte Modell zugegriffen werden.

6.4.3 Training des Textklassifizierungsmodells

ML.NET bietet einem für jedes Machine-Learning-Model eine ähnliche Codestruktur. Diese ist in Abbildung A.12 aufgezeigt. Parallel dazu ist in Codeauflistung 6.2 ein leicht vereinfachter Codeausschnitt für das Trainieren des Materialklassifizierungsmodells zu sehen. Im Trainingsprozess wird nach dem Laden der Trainingsdaten in den Trainingskontext die Pipeline definiert. Mit der `Append()`-Methode können verschiedene Schritte der Pipeline hinzugefügt werden. Im ersten Schritt werden die Texte normalisiert und mit

Bag-of-words und N-Grams in Vektoren umgewandelt. In diesem Schritt kann man auch einige Preprocessing Schritte wie das Entfernen von Stopwords konfigurieren. Mit der `Fit()`-Methode wird die Trainingspipeline ausgeführt und ein Modell erzeugt. Mit `Evaluate` kann das Modell dann analysiert und bewertet werden. Die Methode liefert einige Kennzahlen, wie die Genauigkeit sowie den `LogLoss`, welche in Unterabschnitt 5.1.1 schon für die Auswertung verwendet wurden. Um ein Modell zu persistieren kann es mit `Save()` als ZIP-Datei gespeichert werden. Um das Modell wieder nutzen zu können wird das Modell mit `Load()` geladen und eine `PredictionEngine` erzeugt. Mit diesem Objekt kann die `Predict()`-Methode ausgeführt werden welche die Vorhersage für die Materialüberkategorie liefert.

```

1 // Load your data
2 var materialsDv = mlContext.Data.LoadFromEnumerable(trainable);
3
4 //Define your training pipeline
5 var pipeline =
6     mlContext.Transforms.Conversion.MapValueToKey("Label", "
        MaterialCategory")
7     .Append(mlContext.Transforms.Text.FeaturizeText("
        MaterialBezeichnungFeaturized", options, "
        MaterialBezeichnung"))
8     .AppendCacheCheckpoint(mlContext)
9     .Append(mlContext.MulticlassClassification.Trainers.
        SdcaMaximumEntropy(featureColumnName: "
        MaterialBezeichnungFeaturized", labelColumnName: "Label"))
10    .Append(mlContext.Transforms.Conversion.MapKeyToValue("
        PredictedLabel"));
11
12 // Train and save the model
13 var model = pipeline3.Fit(split.TrainSet);
14 mlContext.Model.Save(model, materialsDv.Schema, MODEL_PATH);
15 //Evaluate the model
16 var testdata = model.Transform(split.TestSet);
17 var evaluation = mlContext.MulticlassClassification.Evaluate(testdata);

```

Codeauflistung 6.2: vereinfachter Code für das Trainieren des Materialklassifizierungsmodells mit *ML.NET*

6.4.4 Training des *fastText* Modells

Für das Trainieren des *fastText*-Modells wird die Python-Bibliothek *Gensim* (vgl. Řehůřek and Sojka, 2010) verwendet. Die Bibliothek implementiert verschiedene Vektorisierungsalgorithmen, wie zum Beispiel *Word2Vec*, *EnsembleLDA* aber auch *fastText*.

Über eine interne Schnittstelle wurden alle ADE-Texte in Textdateien gespeichert. Der Trainingsablauf wurde in drei Skripte aufgeteilt. Das Skript *fasttext_prepare.py* liest die Catalogtexte ein und führt das in Unterabschnitt 4.3.3 erläuterte Preprocessing aus. Das Skript *fasttext_train.py* liest die vorverarbeiteten Texte wieder ein teilt sie in Sätze ein und generiert Tokens aus jedem Satz. Bei der Aufteilung in Sätze werden Abkürzungen in denen ein Punkt vorkommt beachtet und führen zu keinem eigenen Satz. Anschließend wird ein *fastText*-Modell initialisiert, trainiert und am Ende gespeichert.

Mit dem dritten Skript *fasttext_test.py* kann man das trainierte Modell fachlich bewerten. Mit der Methode `most_similar(positive=[word], topn=100)` werden die 100 ähnlichsten Wörter zu einem Wort ausgegeben. Da der Algorithmus DBSCAN mit der Dichte von Punkten und somit der „Ähnlichkeit“ von Worten funktioniert, ist diese Methode eine gute Möglichkeit das Modell fachlich zu evaluieren und testen.

6.4.5 Ausführen der Feinstrukturierung

Die Feinstrukturierung wird im Pythonskript *dbscan_structurize.py* ausgeführt und kann parallel abgearbeitet werden. Nachdem das *fastText*-Modells geladen wurde, werden für jede Überkategorie mit mehr als drei Materialien ein Threat für die Feinstrukturierung gestartet. Das kann man in zwei Schritte einteilen. Da die Begriffe schon bei der Textklassifizierung durch das Preprocessing gelaufen sind, ist das nicht mehr nötig. Der erste Schritt ist das Vektorisieren der Materialbezeichnungen. Hier werden die Begriffe mit `tokenize()` in die einzelnen Wörter geteilt und dann der Vektor durch das Produkt aller Wortvektoren ermittelt. Hierzu werden die Wörter nacheinander mit dem *fastText*-Modell in Vektoren umgewandelt und dann aufmultipliziert. Danach beginnt der zweite Schritt des Clusterings. Der DBSCAN-Parameter *min_samples* wird auf 2 gesetzt, damit Cluster ab zwei Materialien entstehen können. Über den Parameter *eps* wird von 0,5 bis 5 in 0,1-Schritten iteriert. Da jede Ausführung von DBSCAN andere Eingabevektoren hat, muss auch der Parameter *eps* immer anders gewählt werden um das optimale Ergebnis zu bekommen. Das optimale Ergebnis ist, möglichst viele verschiedene Cluster zu erhalten und dabei den Median der Anzahl der Rauschpunkte bei dieser Anzahl von Clustern zu haben. Bei der Iteration über den Parameter kann das optimale Ergebnis, durch evaluieren der jeweiligen Clusteranzahl und Anzahl von

Rauschpunkten, ermittelt werden. Der Startwert und Endwert haben sich durch Testen als passende Werte ergeben, sodass das optimale Ergebnis immer innerhalb des Bereiches liegt. Am Ende wird das optimale Ergebnis an den ASP.NET Service zurückgegeben. Dort wird der Rückgabewert in eine passende Objektstruktur abgebildet und als DTO als HTTP-Response zurückgegeben.

7 Maßnahmen zur Qualitätssicherung

Dieses Kapitel beinhaltet Clean Code Prinzipien und technische Hilfsmittel sowie Unit-Tests um die Qualitätssicherung sicherzustellen.

7.1 Clean Code

Definition 7 (Clean Code). „Clean code is simple and direct. Clean code reads like well-written prose. Clean code never obscures the designer’s intent but rather is full of crisp abstractions and straightforward lines of control. - Grady Booch author of *Object Oriented Analysis and Design with Applications*“ (Martin and Coplien, 2009, p. 8)

Um die Qualitätssicherung bereits vor der Ausführung von Tests sicherzustellen, gilt es gewisse Clean Code Konventionen adäquat anzuwenden. Durch die Verwendung von SonarCloud und SonarLint wird die Anwendung der Code Konventionen sichergestellt. SonarLint führt Analysen während des Programmierens durch und gibt dementsprechend Warnungen bzw. Fehler aus. Welche Konventionen SonarLint analysiert, ist über eine Konfigurationsdatei steuerbar. So ist der Entwickler gezwungen schon beim Entwickeln die verschiedenen Aspekte von Clean Code zu beachten. Auch ein Security Scanner ist in die Build-Pipeline integriert um Sicherheitsrisiken zu erkennen. Zusätzlich läuft mit SonarCloud eine statische Codeanalyse in der Build-Pipeline. Diese verhindert, dass unkonventioneller Code eingecheckt werden kann. Für den entwickelten Service wurden am Ende der Entwicklung keine Code Smells gefunden. Zusätzlich werden Code Reviews bei einem Pull Request durchgeführt. Durch das Vier-Augen-Prinzip werden so zusätzlich Fehler und Code Smells reduziert.

7.2 Technische Hilfsmittel

Bei der Durchsetzung von Maßnahmen der Qualitätssicherung wurden technische Hilfsmittel verwendet. Diese sind Visual Studio, Visual Studio Code, Azure Data Studio und Azure DevOps. Auf diese wird in diesem Abschnitt eingegangen. Alle benötigten

Lizenzen werden von der „ORCA Software GmbH“ zur Verfügung gestellt. Die Implementierung des ASP.NET-Service wird mit Visual Studio Enterprise 2022 von Microsoft durchgeführt. Die Pythonskripte wurden mit Visual Studio Code implementiert. Mit der „Python“ Erweiterung kann der entwickelte Code direkt ausgeführt und gedebugged werden. Für das Verwalten der SQL-Datenbanken wird Azure Data Studio genutzt. Beim Entwickeln wird eine lokale Datenbank verwendet. Diese wird mit *EntityFramework* und dem Code-First Ansatz automatisch erzeugt. Mit SQL-Statements über Azure Data Studio können die Daten aber manuell eingesehen oder editiert werden.

Das Projektmanagement (siehe Abschnitt 2.1) wird mit Azure DevOps umgesetzt. In diesem werden theoretische Arbeiten durchgeführt, als auch alle praktischen Aufgaben, damit alle Daten an einem zentralen Ort sind. Die Plattform hilft dabei Maßnahmen zur Qualitätssicherung zu treffen und umzusetzen. Zu Beginn wurde alle bekannten Aufgaben in Pakete unterteilt. Diese Aufgaben wurden schriftlich formuliert und anschließend als User Stories und Tasks in Azure DevOps eingetragen. Im Sprint Planning wurden dann dementsprechende User Stories dem Sprint zugewiesen. Nachdem ein neuer Quellcode verfasst oder alter Quellcode geändert wurde, wurde dieser auf den Azure DevOps Server geladen und mit dem zugehörigen Arbeitselement verknüpft. Bevor der Stand in das Remote Repository integriert wird, startet eine Pipeline. Diese kompiliert den neu oder geänderten Quellcode, führt die genannten Analysen aus und testet ob alle Tests erfolgreich durchlaufen. Die Nutzung der beiden Tools ist lizenzpflichtig.

7.3 Tests und Abnahme

Um einen funktionsfähigen Service zur Verfügung stellen und sicher weiterentwickeln zu können, muss dieser Ausreichend getestet werden. Für den ASP.NET-Service werden dafür Unit-Tests implementiert. Diese werden auch in der Build-Pipeline geprüft. Getestet wird vor allem die Controllerlogik, Pythonlogik und Repositorylogik. Controller können mit Hilfe von Mocking der genutzten Repositories und Services realisiert werden. Die Pythonskripte wurden inklusive des Python-Interop über die Klasse *IPythonKiExcuter* implementiert. Beim Testen von Repositories wird mit Hilfe des *In-memory Provider* von *EntityFramework* die Datenbank im Arbeitsspeicher emuliert. So können Zugriffe während des Tests durchgeführt werden ohne eine echte Datenbank zu benutzen.

Da im Rahmen dieser Arbeit die Nutzung der Materialstrukturierung temporär nur in einem Draft-Projekt implementiert ist und noch keine komplette Integration in die ORCA AVA stattgefunden hat, gibt es nur eine fachliche Abnahme der Strukturierung der Materialien. In einem Meeting mit dem Entwicklungsleiter und dem Projektmanagement

wird das Ergebnis der Strukturierung durchgesprochen und die zukünftige Verwendung diskutiert. Die Implementierung als Kostengliederungsimport ist weiterhin vorgesehen und wird in Abschnitt 8.2 behandelt.

8 Abschluss

In diesem letzten Kapitel wird ein Fazit gezogen, die Integration in die ORCA AVA aufgezeigt und ein Ausblick gegeben.

8.1 Fazit

In diesem Abschnitt werden die Ergebnisse der Forschung und praktischen Arbeit zusammengefasst, interpretiert und überprüft, ob das Ziel der Arbeit erreicht wurde. Am Ende werden Beschränkungen der Arbeit aufgezeigt.

Zusammenfassung der Ergebnisse

Zu Beginn wurde die Arbeit motiviert und die Problemstellungen definiert. Dafür wurde das Konzept erarbeitet, einen Webservice bereitzustellen, der die Materialstrukturierung in den drei Schritten ausführt. Diese sind das Preprocessing, die Textklassifizierung und die Feinstrukturierung eingeteilt. Für die Materialstrukturierung wurden dazu zu den Anforderungen passende Algorithmen verglichen und ausgewählt. Diese wurden nach den jeweiligen Kriterien der Messbarkeit und in der fachlichen Abstimmung mit dem Produktmanagement evaluiert. Für die Textklassifizierung und die Feinstrukturierung wurde jeweils ein passender Algorithmus gefunden. Die funktionalen Anforderungen wurden mit einem RESTfull ASP.NET Service realisiert. Dieser führt mit Hilfe von Python-Interop die Materialstrukturierung durch. Die Textklassifizierung wird mit *ML.NET* und einem Maximum Entropy Modell mit dem SDCA-Optimierungsalgorithmus durchgeführt. Bei der Feinstrukturierung stellte sich das Nutzen von DBSCAN mit einer vorausliegenden *fastText*-Vektorisierung als passend heraus. Die Daten für die Strukturierung werden in einer SQL-Datenbank persistiert. Zusätzlich entstand ein Draftprojekt, um den Service visuell ansteuern zu können. Dafür wurde die Extraktion aller Materialien aus einer IFC-Datei für den IFC Manager implementiert. Die Maßnahmen der Qualitätssicherung wurden erläutert.

Zielerreichung

Zu Beginn der Arbeit wurde unter Abschnitt 1.3 ein Ziel mit der SMART Methode definiert. Die für das Ziel definierten Anforderungen konnten durch die theoretische Konzeption in Kapitel 4, die Gegenüberstellung der Algorithmen in Kapitel 5 und die Implementierung in Kapitel 6 umgesetzt werden. Auch die Maßnahmen zur Qualitätssicherung wurden umgesetzt und verifiziert. Bei der Evaluation wurden die Kriterien der Messbarkeit beachtet. Die Evaluation, Auswahl und Implementierung lieferten ein brauchbares Ergebnis für die Strukturierung der Materialien, welches von der Qualitätssicherung freigegeben wurde. Ein Draftprojekt wurde vollständig erstellt. Somit wurde das definierte Ziel der Arbeit vor Ablauf der Zeitbegrenzung erreicht.

Textklassifizierung

Die Textklassifizierung mit *ML.NET* und einem Maximum Entropy Modell mit dem SDCA-Optimierungsalgorithmus lieferte trotz Problemstellung 3 (Materialien sind nur Stichpunkte aus wenigen Worten) mit einer Genauigkeit von 88% ein gutes Ergebnis. Mit der Erweiterung der Trainingsdaten kann das Modell noch robuster und auch genauer trainiert werden. Die Implementierung in Python war hier nicht nötig und die Bibliothek *ML.NET* bietet ausreichende Möglichkeiten an, die Textklassifizierung durchzuführen.

Feinstrukturierung

Für die Feinstrukturierung beweist sich das Nutzen einer domänenspezifischen Lösung. Das Trainieren eines *fastText*-Modells mit Ausschreibungstexten aus der Baubranche bietet dem simplen Clustering-Algorithmus DBSCAN es, nach der Bedeutung eines Wortes zu strukturieren. Somit wurde Problemstellung 3 und Problemstellung 4 gelöst und ein fachlich nutzbares Ergebnis erzielt.

Beschränkungen der Arbeit

Neben den behandelten Algorithmen und Bibliotheken wurden weitere Möglichkeiten nur sporadisch untersucht und getestet. Eine erste Recherche hat auf die in der Arbeit untersuchten Lösungen hingewiesen. Alle Möglichkeiten zu analysieren würde über den Rahmen der Arbeit hinausgehen. Falls das Ergebnis der Gegenüberstellung aus Gründen in der Zukunft nicht mehr in Frage kommt, können weitere Möglichkeiten aufgegriffen und wieder in Betracht gezogen werden. Zusätzlich lag der Fokus auf dem Finden passender Algorithmen. Es wurde in der Codebasis der ORCA AVA nur die Möglichkeit bereitgestellt, alle Materialien aus einer IFC-Datei zu extrahieren. Das

Feature der Materialkostengliederung wurde nicht implementiert. In Abschnitt 8.2 wird diese Integration beschrieben und ein Vorschlag geliefert.

8.2 Integration in die ORCA AVA

Da der Service vollumfänglich implementiert ist, kann die Integration in die ORCA AVA beginnen. Neben dem Import einer Materialkostengliederung wird nebenbei ein Import einer Raumkostengliederung aus einer IFC-Datei implementiert. Dabei handelt es sich um die selbe Vorgehensweise. Die Implementierung kann somit verwendet werden und muss nur bei der Beschaffung der Daten aus der IFC-Datei auf den `MaterialManager` zugreifen. **Todo: In der Oberfläche ... Option Feinstrukturierung, Verbesserungen**

8.3 Ausblick

In diesem Abschnitt wird ein Ausblick auf das Feature der Materialstrukturierung gegeben. Nachdem die Integration in die ORCA AVA vorgeschlagen wurde, kann diese jetzt auch als nächstes realisiert werden. Die Erweiterung könnte in der Version 26 in der ORCA AVA Enterprise Edition erscheinen. In der Zukunft können dann auch weitere Schritte folgen, um den IFC-First Ansatz zu realisieren. Hier wird auch der Einsatz von Machine Learning sowie NLP sinnvoll sein. Es muss zum Beispiel anhand der Attribute eines IFC-Bauteils ein Ausschreibungstext erstellt werden.

Eine weitere Nutzungsmöglichkeit der Materialstrukturierung ist das Ermitteln der CO₂-Bilanz eines Gebäudes über eine IFC-Datei. Wenn man bei der Extraktion der Materialien die Mengenangabe inkludiert, kann mit Hilfe von Umweltdatenbanken und deren CO₂-Durchschnittswerte für verschiedene Materialien die CO₂-Bilanz der Baustoffe eines Gebäudes ermittelt werden. Dabei hilft die Strukturierung der Materialien, es müsste aber eine Zuweisung zu den entsprechenden Datenbankeinträgen passieren.

Abbildungen



Metall	<i>Eisen & Stahl</i>	Eisen und Stahl Baustahl · Betonstahlmatte · Betonstahl · Spannbetonstahl · Gusseisen · Profilstahl
	<i>Nichtmetalle</i>	Nichtmetalle Aluminium · Magnesium · Blei · Zinn · Kupfer
Glas	<i>Glas</i>	Glas Glasbaustein · Flachglas · Pressglas
Holz	<i>Holz</i>	Holz Bauholz · Furniersperrholz · Leimbinder · Sperrholz · Grobspanplatte · OSB-Platte · Bretttschichtholz · MDF-Platte · Flachpressplatte · Multiplex-Platte · Holzwole · Strangpressplatte
Kunststoff	<i>Kunststoffe</i>	Kunststoffe (Thermoplaste) Ethylen-Tetrafluorethylen (ETFE) · Polyvinylchlorid (PVC) · Polyethylen (PE) · Polypropylen (PP) · Polymethylmethacrylat (PMMA) · Polystyrol (PS) · Polyvinylacetat (PVAc) · Polycarbonat (PC)
	<i>Duroplaste</i>	Kunststoffe (Duroplaste) Phenol-Formaldehyd-Harz („Bakelit“, PF) · UF-Harz · Melamin-Formaldehyd-Harz (MF) · ungesättigtes Polyester-Harz (UP) Epoxidharz (EP)
Bitumen & Teer	<i>Elastomere</i>	Kunststoffe (Elastomere) Polyurethan (PUR) · Kautschuk · Silikone · Acryl
		Dichtstoffe Bitumen · Noppenbahn
Mineralisch	<i>Asphalt</i>	Asphalt Walzasphalt · Gussasphalt · Naturasphalt · Halbstarre Belag
		Natürliche und künstliche Gesteinskörnungen Sand · Kies · Bims · Hochofenschlacke · Hüttenbims · Blähton · Blähschiefer
	<i>Mörtel</i>	Mörtel Mauermörtel · Putzmörtel · Estrichmörtel · Fliesenkleber
	<i>Beton</i>	Beton Leichtbeton · Normalbeton · Schwerbeton · Stahlbeton · Spannbeton · Faserbeton · Stahlfaserbeton · Porenbeton
	<i>Naturwerkstein</i>	Natürliche Bausteine (Naturwerksteine) Sandstein · Kalkstein · Granit · Schiefer · Marmor · Tuff · Grauwacke · Rhynolith · Basalt und andere
		Künstliche Bausteine Ziegel · Klinker · Tonhohlplatte · Dachziegel · Kalksandstein · Hüttenstein · Leichtbetonstein · Schwerbetonstein · Porenbeton · Betonwerkstein · Betondachstein · Blähton
Verbundbauteile		Fenster, Fassaden, ...
Dämmungen (siehe Arbeitsblatt Dämmstoffe)		Dämmstoffe Flachfasern · Holzwole · Leichtbauplatte (z. B. Heraklith) · Holzweichfaserplatte · Gerreideschüttungen (Ceralith) · Glasfaserdämmstoff · Hanffasern · Mineralfaserdämmstoff · Sch
Gebäudetechnik (?)		
Sonstiges		

Abbildung A.3: Überkategorien für die Klassifizierung von Materialien mit Beispielen

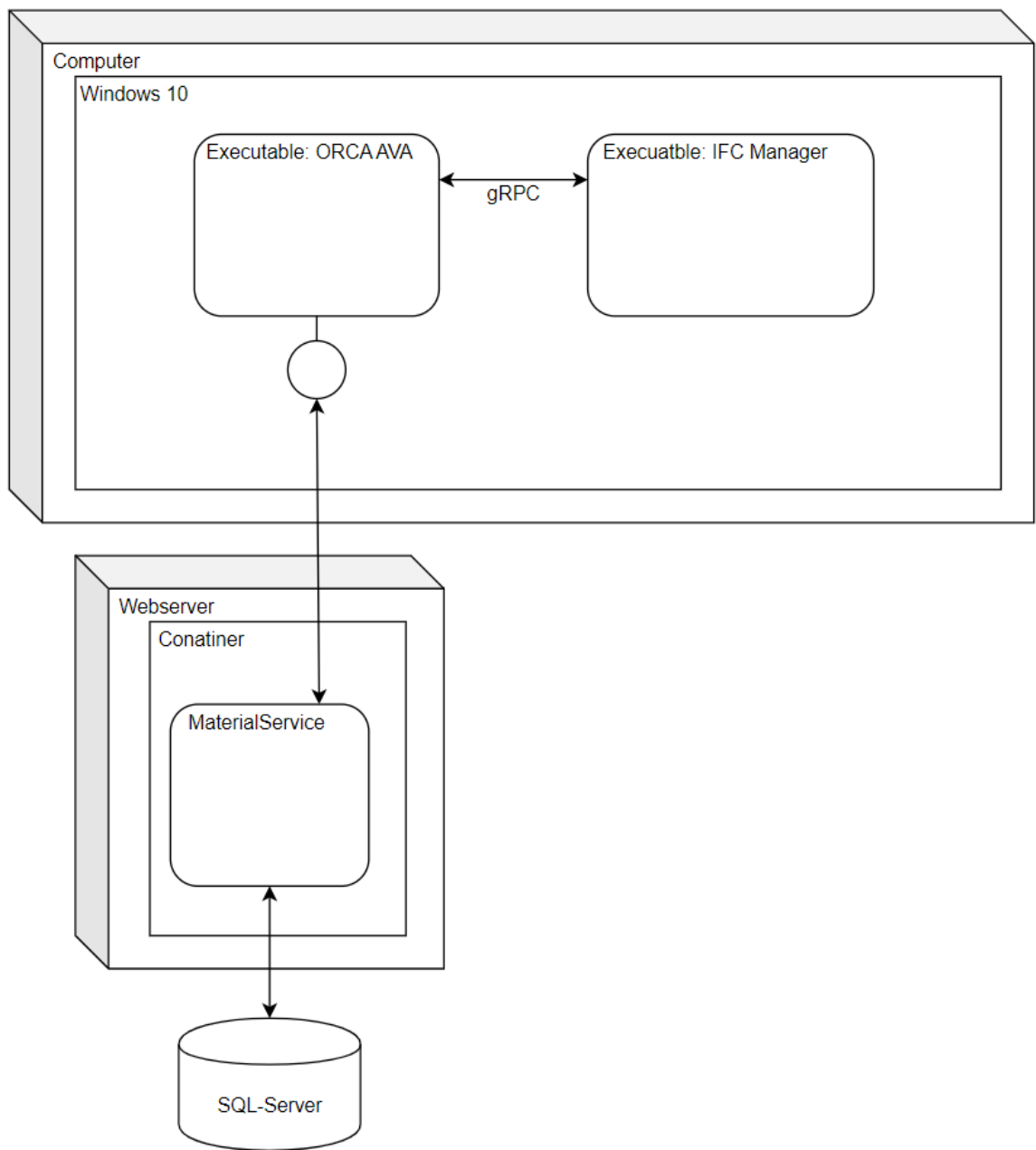


Abbildung A.7: Verteilungsdiagramm der Architektur

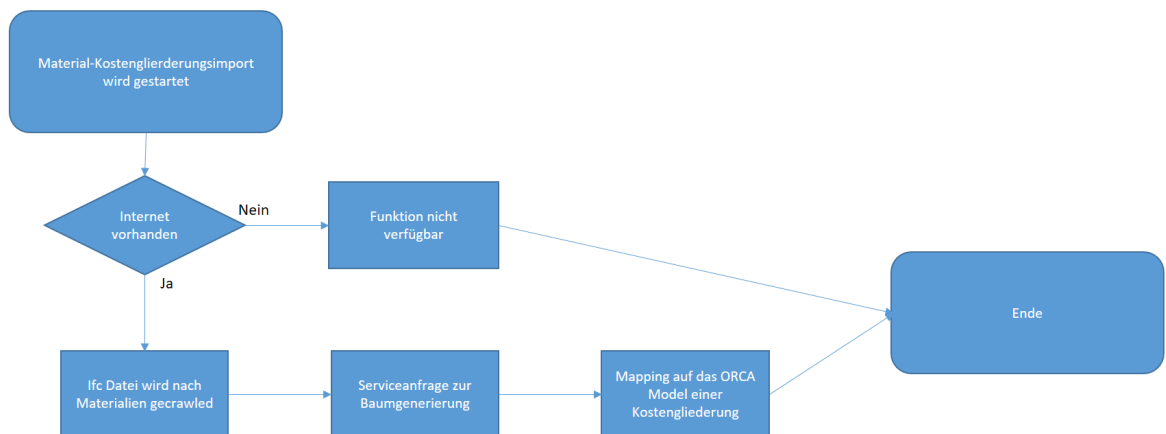


Abbildung A.8: Flussdiagramm Import der Material-Kostengliederung

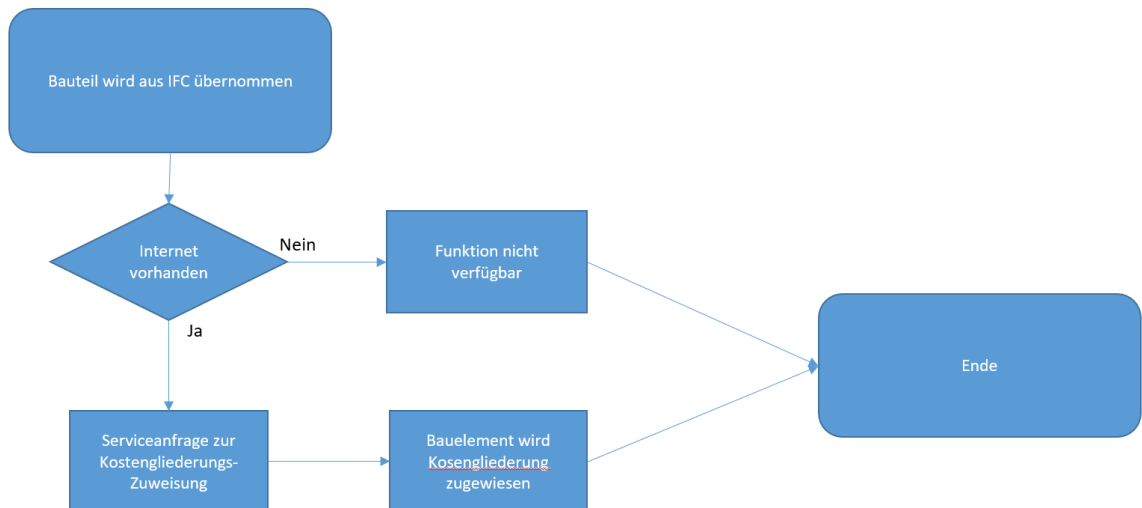


Abbildung A.9: Flussdiagramm Übernahme von Mengen aus dem IFC Manager

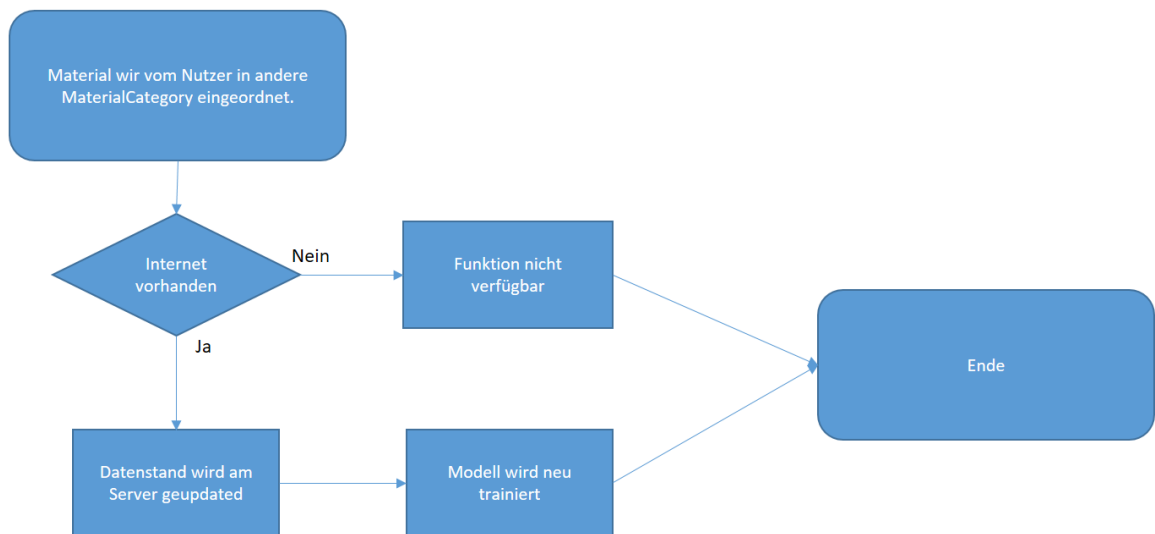


Abbildung A.10: Flussdiagramm Bewertung des Kostengliederungs-Import

Heizestrich -> Mineralisch

☐ Unknown

☐ Metall

☐ Glas

☐ Holz

☐ Kunststoff

☐ BitumenUndTeer

☒ Mineralisch

☐ Verbundbauteile

☐ Dämmungen

☐ GebäudeTechnik

☐ Sonstiges

Submit

Skip

Erklärung

Es werden dir immer Baustoffe angezeigt, die in eine Baustoffkategorie zugeordnet werden sollen.

Abbildung A.11: Oberfläche des Klassifizierungs-Tools für Materialien

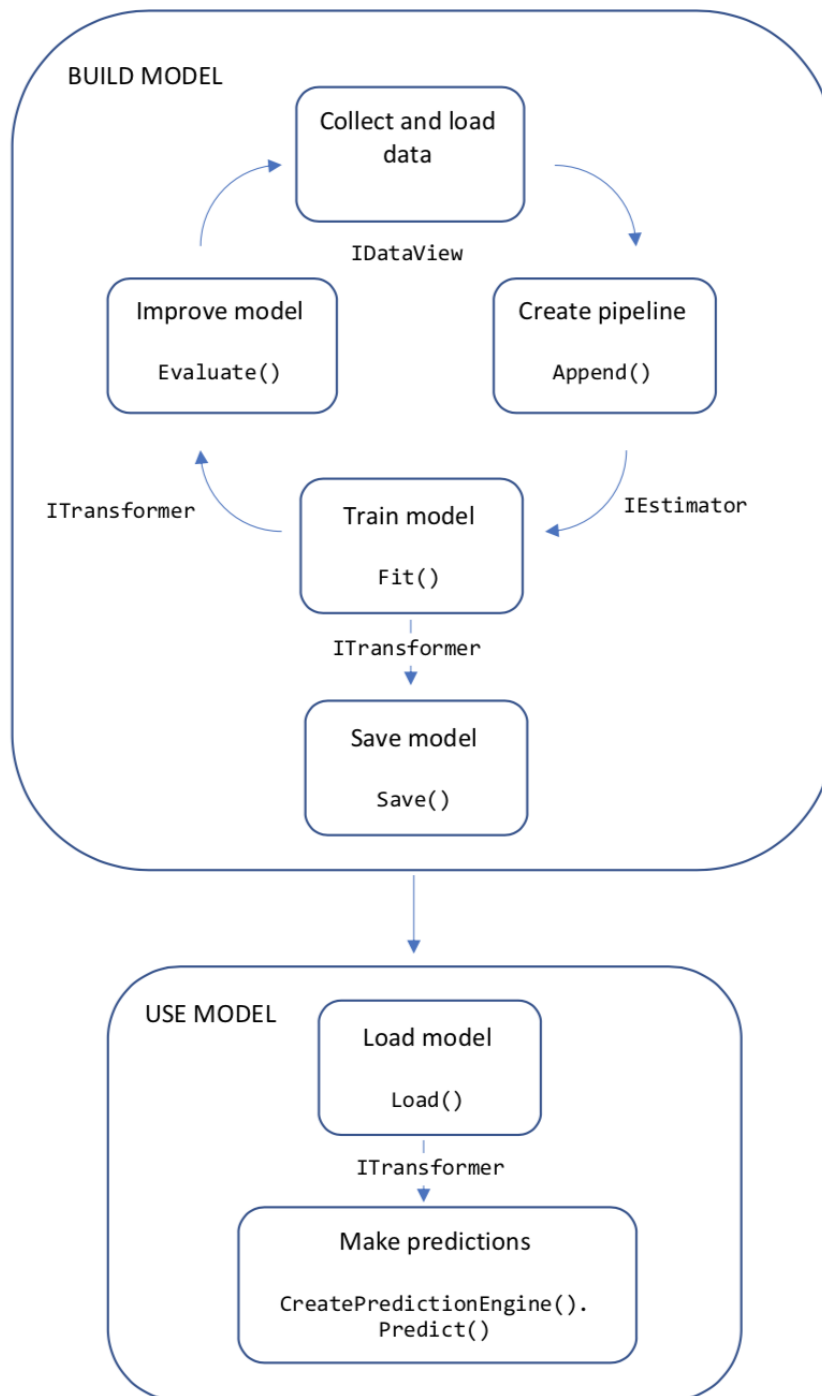


Abbildung A.12: Code Workflow von *ML.NET* (Quelle: Microsoft (2022c))

Listings

```

1 public async Task<CostStructureDto> GenerateMaterialCostStructureAsync(
    ICollection<string> materialNames, bool structurizeFine)
2 {
3     if (materialNames is null)
4         throw new ArgumentNullException(nameof(materialNames));
5
6     materialNames = materialNames.Distinct().ToArray();
7
8     List<Material> materials = await mMaterialRepository.CreateNewAsync
        (materialNames.Select(name => new Material() {
            MaterialBezeichnung = name, Appearances = 1 }));
9
10    ICollection<MaterialStructureNode> result = mMaterialStructorizer
11        .StructurizeMaterials(materials.Select(m => m.
            PreprocessedMaterialBezeichnung).ToArray(), structurizeFine);
12
13    var dto = new CostStructureDto();
14    dto.SetMaterials(result.Select(r => r.ToDto()).ToList());
15    return dto;
16 }

```

Codeaufistung A.1: Implementierung

der Methode GenerateMaterialCostStructureAsync der Klasse
MateialService

```

1 public static readonly string ModelPreprocess = "preprocess";
2 public static readonly string InputPreprocess = "material_raw";
3 public static readonly string OutputPreprocess = "material_preprocessed";
4
5 public string PreprocessMaterial(string material)
6 {
7     try
8     {
9         PythonEngine.Initialize();
10
11         using (Py.GIL())
12         {
13             using (PyModule scope = Py.CreateScope())
14             {
15                 dynamic preprocessScript = scope.Import(
                    PythonKiExecuterConstants.ModelPreprocess);
16                 string materialPreprocessed = preprocessScript.
                    preprocess(material);

```

```

17
18         return materialPreprocessed;
19     }
20 }
21 }
22 catch (Exception ex)
23 {
24     Console.WriteLine(ex.Message);
25     throw;
26 }
27 finally
28 {
29     PythonEngine.Shutdown();
30 }
31 }

```

Codeauflistung A.2: Python-Interopt für das Preprocessing

```

1 import re
2 words_to_remove = ['rot','blau','gelb','grün','gruen','weiß','weiss','
    schwarz','grau','rosa','pink','lila','braun','gold','silber','
    orange','dunkelgrau','beige','dunkel','hell']
3 material_raw:str
4 material_preprocessed:str
5
6 def preprocess(material_name):
7     old = material_name;
8     material_name = re.sub(r'[\d]{1,3}[\s_-]+[\d]{1,3}[\s_-]+[\d]{1,3}'
        ,'',material_name);
9     #remove special characters eg ,.-;:_
10    material_name = (" ".join(re.findall(r"[A-Za-z0-9üäöÜÄÖßóâéeÉÉÓ/]"
        , material_name))).replace(" ", " ");
11    #remove size and add afterwards as one
12    size_threedimensional = re.compile('[\d]{1,4}[\s]?[x][\s]?[\d]{1,4}([\s]?[x][\s]?[\d]{1,4})?');
13    size_twodimensional = re.compile('[\d]{1,4}[\s]?[x][\s]?[\d]{1,4}')
        ;
14    # size = re.findall('[\d]{1,4}[\s]?[x][\s]?[\d]{1,4}[\s]?[x][\s]?[\d]{1,4}', material_name);
15    size = re.findall('[\d]{1,4}[\s]?[x][\s]?[\d]{1,4}([\s]?[x][\s]?[\d]{1,4})?', material_name);
16    if len(size) <= 0:
17        size = re.findall(size_twodimensional, material_name);
18    material_name = re.sub(size_threedimensional,'',material_name);

```



```

19     for word in words_to_remove:
20         if word in material_name.lower():
21             index = material_name.lower().index(word);
22             if index >= 0:
23                 material_name = material_name[:index] + material_name[
24                     index+len(word):];
25     while(' ' in material_name):
26         material_name = material_name.replace(' ', ' ')
27     material_name = material_name.strip();
28     print(f"'{old}' -> '{material_name}'")
29     return material_name;
30
31 print(__name__);
32 if __name__ == "__main__":
33     try:
34         material_raw
35     except NameError:
36         material_raw = input("Text to preprocess: ")
37     material_preprocessed = preprocess(material_raw)
38     print(material_preprocessed)

```

Codeauflistung A.3: Python-Interopt für das Preprocessing

Tabellen

Predicted ->	0	1	2	3	4	5	6	7	8	9		Recall
Truth												
Mineralisch	22	0	0	0	0	2	0	0	0	0		0,9167
Sonstiges	4	8	0	0	0	0	0	0	0	0		0,6667
Dämmungen	0	0	7	0	0	0	0	0	0	0		1,0000
Holz	0	0	0	7	0	0	0	0	0	0		1,0000
Verbundbauteile	1	0	0	0	5	0	0	1	0	0		0,7143
Metall	2	1	0	0	0	25	0	0	0	0		0,8929
Kunststoff	0	0	0	0	0	0	4	0	0	0		1,0000
Glas	0	0	0	0	0	0	0	4	0	0		1,0000
GebäudeTechnik	0	0	0	0	0	0	0	0	1	0		1,0000
BitumenUndTeer	0	0	0	0	0	0	0	0	0	0		0,0000
Precision	0,7586	0,8889	1,0000	1,0000	1,0000	0,9259	1,0000	0,8000	1,0000	0,0000		

Tabelle A.1: Konfusionsmatrix: ScdaMaximumEntropy

Predicted ->	0	1	2	3	4	5	6	7	8	9		Recall
Truth												
0. Mineralisch	23	0	0	0	0	1	0	0	0	0		0,9583
1. Sonstiges	9	3	0	0	0	0	0	0	0	0		0,2500
2. Dämmungen	4	0	3	0	0	0	0	0	0	0		0,4286
3. Holz	1	0	0	6	0	0	0	0	0	0		0,8571
4. Verbundbauteile	4	0	0	0	3	0	0	0	0	0		0,4286
5. Metall	10	0	0	0	0	18	0	0	0	0		0,6429
6. Kunststoff	1	0	0	0	0	0	3	0	0	0		0,7500
7. Glas	0	0	0	0	0	0	0	4	0	0		1,0000
8. GebäudeTechnik	1	0	0	0	0	0	0	0	0	0		0,0000
9. BitumenUndTeer	0	0	0	0	0	0	0	0	0	0		0,0000
Precision	0,4340	1,0000	1,0000	1,0000	1,0000	0,9474	1,0000	1,0000	0,0000	0,0000		

Tabelle A.2: Konfusionsmatrix: LbfgsMaximumEntropy

Predicted ->	0	1	2	3	4	5	6	7	8	9		Recall
Truth												
0. Mineralisch	22	0	0	0	0	2	0	0	0	0		0,9167
1. Sonstiges	4	8	0	0	0	0	0	0	0	0		0,6667
2. Dämmungen	0	0	7	0	0	0	0	0	0	0		1,0000
3. Holz	0	0	0	7	0	0	0	0	0	0		1,0000
4. Verbundbauteile	1	0	0	0	4	1	0	1	0	0		0,5714
5. Metall	2	1	0	0	0	25	0	0	0	0		0,8929
6. Kunststoff	0	0	0	0	0	0	4	0	0	0		1,0000
7. Glas	0	0	0	0	0	0	0	4	0	0		1,0000
8. GebäudeTechnik	0	0	0	0	0	0	0	0	1	0		1,0000
9. BitumenUndTeer	0	0	0	0	0	0	0	0	0	0		0,0000
Precision	0,7586	0,8889	1,0000	1,0000	1,0000	0,8929	1,0000	0,8000	1,0000	0,0000		

Tabelle A.3: Konfusionsmatrix: OneVersusAll

Predicted ->	0	1	2	3	4	5	6	7	8	9		Recall
Truth												
0. Mineralisch	21	0	0	0	0	3	0	0	0	0		0,8750
1. Sonstiges	6	3	0	0	0	3	0	0	0	0		0,2500
2. Dämmungen	7	0	0	0	0	0	0	0	0	0		0,0000
3. Holz	5	0	0	1	0	1	0	0	0	0		0,1429
4. Verbundbauteile	6	0	0	0	0	1	0	0	0	0		0,0000
5. Metall	11	0	0	0	0	17	0	0	0	0		0,6071
6. Kunststoff	1	0	0	0	0	0	3	0	0	0		0,7500
7. Glas	2	0	0	0	0	2	0	0	0	0		0,0000
8. GebäudeTechnik	1	0	0	0	0	0	0	0	0	0		0,0000
9. BitumenUndTeer	0	0	0	0	0	0	0	0	0	0		0,0000
Precision	0,3500	1,0000	0,0000	1,0000	0,0000	0,6296	1,0000	0,0000	0,0000	0,0000		

Tabelle A.4: Konfusionsmatrix: TextclassificationersusAll

Begriff	Erwartet	DBSCAN	OpenAI
Ortbeton C30/37 Verputzt	1	1	
Trockenbau Gipsplatte			
Fußboden Heizestrich	3	3	3
Geländearbeiten Gebundene Schüttung	4	4	4
Ortbeton C30/37	1	1	
Mauerwerk mit Daemmeigenschaften	2	2	2
Fußboden Fußbodenaufbau	3	3	3
Fußboden Schüttung	3		3
Putz			
Geländearbeiten Rollierung Schuettung	4	4	4
Ortbeton bewehrt geschliffen	1		
Geländearbeiten Aufgeschüttet	4	4	4
Fußboden Estrich	3	3	3
Beton C 25/30	1	1	
Dachdeckung Ziegel			
Mauerwerk ohne Daemmeigenschaften	2	2	2
Auswertung	richtig	teilweise	teilweise

Tabelle A.5: Überkategorie: Mineralisch

Begriff	Erwartet	DBSCAN	OpenAI
Keramik			
Trockenbau Gipsplatte	1	1	1
Trockenbau Gipsplatte 2	1	1	1
Trockenbau Gipsplatte 3	1	1	1
Trockenbau Gipsplatte 4	1	1	1
Fußboden Teppich	2		2
Fußboden Estrich	2		2
Fußboden Trittschall	2	2	2
Fußboden Schüttung	2	2	2
Geschossdecke FB 200 Fliese	3	3	
Fußboden Fliese Travertin	2	3	2
Ortbeton bewehrt	3		
Mauerwerk Ziegel			
Auswertung	richtig	falsch	teilweise

Tabelle A.6: Überkategorie: Mineralisch

Begriff	Erwartet	DBSCAN	OpenAI
Gipskarton			
Keramik Fliesen			
Mauerwerk Betonblock	1	1	1
Beton Gegossen	2	2	2
Mauerwerk Ziegel	1	1	1
Beton	2	2	2
Auswertung	richtig	richtig	richtig

Tabelle A.7: Überkategorie: Mineralisch

Begriff	Erwartet	DBSCAN	OpenAI
Stütze Auflager			-1
Kalksandstein			
Kies			
Fliesen			2
Estrich			2
Gipsputz			
Volkernplatte			1
Ausbauplatte GKBI	1	1	1
Ausbauplatte GKB	1	1	1
Auswertung	richtig	richtig	falsch

Tabelle A.8: Überkategorie: Mineralisch

Begriff	Erwartet	DBSCAN	OpenAI
WI KS 17 5	1	1	
Kalksandstein	1		1
WI KS 11 5	1	1	1
DA Flachdach Kiesschicht	3		
Kies	3		
Fliesen			3
Estrich			3
Gipsputz			
Ausbauplatte GKBI	2	2	2
Ausbauplatte GKB	2	2	2
Auswertung	richtig	teilweise	falsch

Tabelle A.9: Überkategorie: Mineralisch

Begriff	Erwartet	DBSCAN	OpenAI
Mauerwerk Langlochziegel verputzt	1	1	1
Mauerwerk Ziegel verputzt	1	1	1
MW Kalksandstein verputzt	1	1	1
Knauf Bauplatte Typ A			
Decke Fliesen			
Beton Estrich			
Porenbeton verputzt	1	1	
Vorgabe Dach			
Auswertung	richtig	richtig	teilweise

Tabelle A.10: Überkategorie: Mineralisch

Begriff	Erwartet	DBSCAN	OpenAI
A W TR MW KS 17 5	4	4	
FAS PUTZ TYP1 0 2 0	1	1	
I BOD ESTRICH ZE 6	2	2	2
I BOD ESTRICH ZE 7	2	2	2
FAS PUTZ TYP1 0 2	1	1	
I BOD BELAG BETONSTEIN 3	3	3	3
I BOD ESTRICH ZE 4	2	2	2
I BOD BELAG FLIESE 1 5	3	3	3
I BOD ESTRICH ZE 5 5	2	2	2
I W TR MW KS 11 5	4	4	
A BOD KIES 5			
Gefällebeton	5		
Beton	5		
Concrete	5		
Auswertung	richtig	teilweise	teilweise

Tabelle A.11: Überkategorie: Mineralisch

Begriff	Erwartet	DBSCAN	OpenAI
Metall Edelstahl gebürstet	1	1	1
Fensterbank Aussen Blech			-1
Metall Edelstahl Satiniert	1	1	1
Metall Stahl matt		1	
Auswertung	richtig	falsch	falsch

Tabelle A.12: Überkategorie: Metall

Begriff	Erwartet	DBSCAN	OpenAI
Metall Titanzink		2	
Metall Oberfläche lackiert elfenbein Mit Glaz	1	1	1
Metall Oberfläche lackiert matt	1	1	1
Metall Aluminium		2	
Metall lackiert	1	1	
Metall verzinkt		2	
Auswertung	richtig	richtig	teilweise

Tabelle A.13: Überkategorie: Metall

Begriff	Erwartet	DBSCAN	OpenAI
Metall Edelstahl Satiniert	1	1	1
Metall Edelstahl gebürstet	1	1	1
Metall Stahl matt	2	1	
Metal Strugal Stainless Steel	1	2	
Aluminum Strugal RAL 9016		2	
Fensterbank Aussen Blech		2	
Auswertung	richtig	falsch	teilweise

Tabelle A.14: Überkategorie: Metall

Begriff	Erwartet	DBSCAN	OpenAI
StB C45	1	1	1
StB C55	1	1	1
StB C35	1	1	1
StB C30	1	1	1
StB C60	1	1	1
Fertigteil StB C35	1	1	1
StB FU	1		1
Auswertung	richtig	teilweise	richtig

Tabelle A.15: Überkategorie: Verbundbauteil

Begriff	Erwartet	DBSCAN	OpenAI
Aluminium			
Stahl verzinkt			
Edelstahl			
Auswertung	richtig	richtig	richtig

Tabelle A.16: Überkategorie: Metall

Begriff	Erwartet	DBSCAN	OpenAI
DA Flachdach Gefälledämmung	2	2	1
Mineralwolle 4	1	1	1
WA XPS Dämmung 10cm	3		1
XPS	3	2	1
DA Flachdach Dämmung 14cm	2	2	1
Mineralwolle 3	1	1	1
Brandriegel			
Trittschalldämmung			
Auswertung	richtig	falsch	falsch

Tabelle A.17: Überkategorie: Dämmungen

Begriff	Erwartet	DBSCAN	OpenAI
DA Flachdach Dämmung 14cm	3	2	1
Mineralwolle 3	1	1	1
WA XPS Dämmung 10cm	2	2	1
XPS	2		1
DA Flachdach Gefälledämmung	3	2	1
Mineralwolle 4	1	1	1
Trittschalldämmung			
Brandriegel			
Auswertung	richtig	falsch	falsch

Tabelle A.18: Überkategorie: Dämmungen

Begriff	Erwartet	DBSCAN	OpenAI
FAS WDVS WD EPS 8	1	1	1
FAS WDVS WD EPS 18	1	1	1
I BOD DAEM TD 3			2
I BOD DAEM WD 14	2	2	2
I BOD DAEM WD 15	2	2	2
A BOD DAEM XPS 11	4	4	4
A W NT DAEM PD XPS 12	3	3	3
A W NT DAEM PD XPS 6	3	3	3
A BOD DAEM XPS 12	4	4	4
A BOD DAEM XPS 10	4	4	4
Auswertung	richtig	richtig	teilweise

Tabelle A.19: Überkategorie: Dämmungen

Begriff	Erwartet	DBSCAN	OpenAI
Glas Isolierverglasung klar	1	1	1
Glas Isolierverglasung 3 fach	1	2	1
Glas Isolierverglasung 2 fach	1	2	1
Glas klar	1	1	
Glas matt			
Auswertung	richtig	falsch	teilweise

Tabelle A.20: Überkategorie: Glas

Begriff	Erwartet	DBSCAN	OpenAI
Glas matt			
Glas Isolierverglasung Bronze	1		1
Glass Strugal Clear Glazing	1	2	
Glas klar	1	1	
Glass Simonton Clear	1	2	
Glas Isolierverglasung klar	1	1	1
Auswertung	richtig	falsch	teilweise

Tabelle A.21: Überkategorie: Glas

Begriff	Erwartet	DBSCAN	OpenAI
Geschossdecke FB 200 Parkett	2	2	-1
Holz generisch		1	
Fußboden Parkett eiche	2	2	-1
Dachdeckung Holz	1	1	1
Dachdeckung Holz 2	1	1	1
Auswertung	richtig	teilweise	falsch

Tabelle A.22: Überkategorie: Holz

Begriff	Erwartet	DBSCAN	OpenAI
Holz Parkett			1
Holz Birke	1	1	1
Holz Eiche	1	1	1
Standard Treppe			
Auswertung	richtig	richtig	teilweise

Tabelle A.23: Überkategorie: Holz

Begriff	Erwartet	DBSCAN	OpenAI
Dachdeckung Holz			
Fußboden Terrasse Teakholz			
Holz HSB Steher			
Holz Astfichte senkrecht			
Auswertung	richtig	richtig	richtig

Tabelle A.24: Überkategorie: Holz

Begriff	Erwartet	DBSCAN	OpenAI
Fußboden Epoxidharzbeschichtung			
BSt 500 M A	1	1	1
BSt 500 M B	1	1	1
Auswertung	richtig	richtig	richtig

Tabelle A.25: Überkategorie: Verbundbauteile

Begriff	Erwartet	DBSCAN	OpenAI
A W TR STB OB 17 5	1	1	-1
I W TR STB OB 25	1	2	-1
A DE TR STB OB 30	3		-1
I W TR STB OB 12	1	2	-1
A W TR STB OB 25	1	1	-1
A W TR STB OB 30	1	1	-1
I W TR STB OB 22	1	2	-1
I DE TR STB OB 20	3	3	-1
I DE TR STB OB 45	3	3	-1
I DE TR STB OB 25	3	3	-1
Auswertung	richtig	teilweise	teilweise

Tabelle A.26: Überkategorie: Verbundbauteile

B Abkürzungsverzeichnis

IFC Industry Foundation Classes

IFC2x3 IFC Version 2.3

IFC4 IFC Version 4

BIM Building Information Modeling

KG Kostengruppe

DevOps Development Operations

NLP Natural Language Processing

AVA Ausschreibung, Vergabe und Abrechnung

ADE AUSSCHREIBEN.DE

BIM Building Information Modeling

DIN Deutsches Institut für Normung e. V.

SE Starter Edition

PE Professional Edition

EE Enterprise Edition

VB Visual Basic

GUI Graphical User Interface

WPF Windows Presentation Foundation

ISO International Organization for Standardization

STEP Standard for the exchange of product model data

SPF STEP Physical Format

XML Extensible Markup Language

CAD Computer-aided design

REST Representational State Transfer

KI Künstliche Intelligenz

IoT Internet of Things

RGB Rot,Grün,Blau

API Application Programming Interface

GPT Generative Pretrained Transformer

VB Visual Basic

GUID Globally Unique Identifier

DTO Data Transfer Object

URL Uniform Resource Locator

DBSCAN Density-Based Spatial Clustering of Applications with Noise

CBOW Continuous Bag Of Words

HTTP Hypertext Transfer Protocol

SQL Structured Query Language

SDCA Stochastic dual coordinated ascent

L-BFGS limited memory Broyden-Fletcher-Glodfarb-Shanno method

C Definitionsverzeichnis

1	Definition (Scrum)	6
2	Definition (Funktionale Anforderung)	13
3	Definition (Feature extraction)	21
4	Definition (Interpretierbarkeit)	22
5	Definition (Entropie)	24
6	Definition (Transformer)	27
7	Definition (Clean Code)	43

D Quellcodeverzeichnis

6.1	Python-Interopt für das Preprocessing	37
6.2	vereinfachter Code für das Trainieren des Materialklassifizierungsmodells mit <i>ML.NET</i>	40
A.1	Implementierung der Methode <code>GenerateMaterialCostStructure-</code> <code>Async</code> der Klasse <code>MaterialService</code>	55
A.2	Python-Interopt für das Preprocessing	56
A.3	Python-Interopt für das Preprocessing	57

Literaturverzeichnis

- Adriaans, P. (2020), Information, *in* E. N. Zalta, ed., ‘The Stanford Encyclopedia of Philosophy’, Fall 2020 edn, Metaphysics Research Lab, Stanford University.
- Beetz, F. and Harrer, S. (2021), ‘GitOps: The Evolution of DevOps?’, *IEEE Software* **39**.
- Berger, A. L., Pietra, V. J. D. and Pietra, S. A. D. (1996), ‘A maximum entropy approach to natural language processing’, *Comput. Linguist.* **22**(1), 39–71.
- Bie, Q. (2019), ‘Words are not enough! Short text classification using words as well as entities’.
- URL:** <https://www.fiz-karlsruhe.de/sites/default/files/FIZ/Dokumente/Forschung/ISE/Masterarbeiten/Thesis-Qingyuan-Bie.pdf>
- Bojanowski, P., Grave, E., Joulin, A. and Mikolov, T. (2017), ‘Enriching word vectors with subword information’.
- Bojanowski, P., Joulin, A. and Mikolo, T. (2016), ‘Fair open-sources fasttext’. Accessed: 2023-4-26.
- URL:** <https://research.facebook.com/blog/2016/8/fasttext/>
- Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I. and Amodei, D. (2020), ‘Language models are few-shot learners’.
- buildingSMART International Ltd. (2017*a*), ‘Industry Foundation Classes - IfcMaterial’. Accessed: 2023-1-31.
- URL:** http://standards.buildingsmart.org/IFC/RELEASE/IFC4_1/FINAL/HTML/link/ifcmaterial.htm

- buildingSMART International Ltd. (2017b), ‘Industry Foundation Classes 4x1 - Material Association’. Accessed: 2023-1-31.
URL: http://standards.buildingsmart.org/IFC/RELEASE/IFC4_1/FINAL/HTML/link/material-association.htm
- buildingSMART International Ltd. (2017c), ‘Industry Foundation Classes 4x3- IfcPropertySet’. Accessed: 2023-1-31.
URL: https://standards.buildingsmart.org/IFC/DEV/IFC4_3/RC1/HTML/schema/ifckernel/lexical/ifcpropertyset.htm
- Chandrasekar, P. and Qian, K. (2016), The impact of data preprocessing on the performance of a naive bayes classifier, *in* ‘2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)’, Vol. 2, pp. 618–619.
- Chen, J., Hu, Y., Liu, J., Xiao, Y. and Jiang, H. (2019), ‘Deep short text classification with knowledge powered attention’.
- Ferrone, L. and Zanzotto, F. M. (2019), ‘Symbolic, distributed, and distributional representations for natural language processing in the era of deep learning: A survey’.
- Gensim (n.d.), ‘Comparison of fasttext and word2vec’. Accessed: 2023-4-25.
URL: https://github.com/RaRe-Technologies/gensim/blob/develop/docs/notebooks/Word2Vec_FastText_Comparison.ipynb
- Gensim Documentation (n.d.), ‘Gensim documentation - FastText model’. Accessed: 2023-4-25.
URL: https://radimrehurek.com/gensim/auto_examples/tutorials/run_fasttext.html
- Glinz, M. (2007), On non-functional requirements, *in* ‘15th IEEE International Requirements Engineering Conference (RE 2007)’, pp. 21 – 26.
- Gonzalez, Z. and Carlos, V. (2019), Towards explaining the effects of data preprocessing on machine learning, *in* ‘2019 IEEE 35th International Conference on Data Engineering (ICDE)’, pp. 2086–2090.
- Grutters, B. (2022), ‘Industry Foundation Classes (IFC) – An Introduction’. Accessed: 2023-2-22.
URL: <https://technical.buildingsmart.org/standards/ifc/>

- Guyon, I. and Elisseeff, A. (2006), *Feature Extraction Preface*, Springer Berlin Heidelberg, Berlin, Heidelberg.
URL: https://doi.org/10.1007/978-3-540-35488-8_1
- Hölzing, J. A. (2008), *Die Kano-Theorie der Kundenzufriedenheitsmessung*, Gabler Verlag Wiesbaden.
- Industry Foundation Classes* (2017). Accessed: 2023-1-31.
URL: http://standards.buildingsmart.org/IFC/RELEASE/IFC4_1/FINAL/HTML
- Joulin, A., Grave, E., Bojanowski, P. and Mikolov, T. (2016), ‘Bag of tricks for efficient text classification’.
- Ken, S. and Jeff, S. (2020), ‘The scrum guide’.
URL: <https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf>
- Laakso, M. and Kiviniemi, A. O. (2012), ‘The IFC standard: A review of history, development, and standardization, information technology’, *J. Inf. Technol. Constr.* **17**(9), 134–161.
- Le, Q. V. and Mikolov, T. (2014), ‘Distributed representations of sentences and documents’.
- Liu, D. C. and Nocedal, J. (1989), ‘On the limited memory bfgs method for large scale optimization’, *Mathematical Programming* **45**, 503–528.
- Martin, R. C. and Coplien, J. O. (2009), *Clean code: a handbook of agile software craftsmanship*, Prentice Hall, Upper Saddle River, NJ [etc.].
URL: https://www.amazon.de/gp/product/0132350882/ref=oh_details_o00_s00_i00
- Microsoft (2021), ‘Auswählen eines Algorithmus für maschinelles Lernen - Azure Machine Learning’. Accessed: 2023-05-02.
URL: <https://learn.microsoft.com/de-de/azure/machine-learning/how-to-select-algorithms>
- Microsoft (2022a), ‘Auswählen eines Algorithmus für maschinelles Lernen - Azure Machine Learning’. Accessed: 2023-3-10.

URL: <https://learn.microsoft.com/de-de/azure/machine-learning/how-to-select-algorithms>

Microsoft (2022b), ‘What is ML.NET and how does it work?’. Accessed: 2023-5-02.

URL: <https://learn.microsoft.com/en-us/dotnet/machine-learning/how-does-ml-dotnet-work>

Microsoft (2022c), ‘What is ml.net and how does it work?’. Accessed: 2023-5-02.

URL: <https://learn.microsoft.com/en-us/dotnet/machine-learning/how-does-ml-dotnet-work>

Microsoft (2022d), ‘Windows presentation foundation (WPF)’. Accessed: 2023-2-10.

URL: <https://github.com/dotnet/wpf>

Microsoft (n.d.a), ‘Lbfgsmaximumentropy-multiclass-trainer class’. Accessed: 2023-5-02.

URL: <https://learn.microsoft.com/en-us/dotnet/api/microsoft.ml.trainers.lbfgsmaximumentropy-multiclass-trainer?view=ml-dotnet>

Microsoft (n.d.b), ‘Lbfgsmaximumentropy-multiclass-trainer class’. Accessed: 2023-5-02.

URL: <https://learn.microsoft.com/en-us/dotnet/api/microsoft.ml.trainers.lbfgsmaximumentropy-multiclass-trainer?view=ml-dotnet>

Microsoft (n.d.c), ‘Multiclassclassificationmetrics class’. Accessed: 2023-5-02.

URL: <https://learn.microsoft.com/en-us/dotnet/api/microsoft.ml.data.multiclassclassificationmetrics?view=ml-dotnet>

Microsoft (n.d.d), ‘Sdcalogisticregressionbinarytrainer class’. Accessed: 2023-5-02.

URL: <https://learn.microsoft.com/en-us/dotnet/api/microsoft.ml.trainers.sdcalogisticregressionbinarytrainer?view=ml-dotnet>

Microsoft (n.d.e), ‘Sdcamulticlasstrainerbase<tmodel> class’. Accessed: 2023-5-02.

URL: <https://learn.microsoft.com/en-us/dotnet/api/microsoft.ml.trainers.sdcamulticlasstrainerbase-1?view=ml-dotnet>

Miller, T. (2017), ‘Explanation in Artificial Intelligence: Insights from the Social Sciences’.

URL: <https://arxiv.org/abs/1706.07269>

Molnar, C. (2022), *Interpretable Machine Learning*, 2 edn.

URL: <https://christophm.github.io/interpretable-ml-book>

.NET Foundation and Contributors (n.d.), ‘ASP.NET core’. Accessed: 2023-3-20.

URL: <https://github.com/dotnet/aspnetcore/blob/main/README.md>

Nooralahzadeh, F., Øvrelid, L. and Lønning, J. T. (2018), Evaluation of domain-specific word embeddings using knowledge resources, in ‘Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)’, European Language Resources Association (ELRA), Miyazaki, Japan.

URL: <https://aclanthology.org/L18-1228>

OpenAI (2022), ‘Introducing ChatGPT’. Accessed: 2023-4-24.

URL: <https://openai.com/blog/chatgpt>

OpenAI (n.d.a), ‘Api reference’. Accessed: 2023-5-09.

URL: <https://platform.openai.com/docs/api-reference/introduction>

OpenAI (n.d.b), ‘Pricing’. Accessed: 2023-5-09.

URL: <https://openai.com/pricing>

ORCA Software GmbH (n.d.), ‘ORCA helpdesk - ORCA AVA 25 - Kostengliederungen als Basis für das Kostenmanagement’. Accessed: 2023-2-1.

URL: https://helpdesk.orca-software.com/Solution/AVA25/Content/Kontexthilfe-337_8277

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C. L., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., Schulman, J., Hilton, J., Kelton, F., Miller, L., Simens, M., Aspell, A., Welinder, P., Christiano, P., Leike, J. and Lowe, R. (2022), ‘Training language models to follow instructions with human feedback’.

Quintanilla, L. (2022), ‘Introducing the ml.net text classification api (preview)’. Accessed: 2023-5-02.

URL: <https://devblogs.microsoft.com/dotnet/introducing-the-ml-dotnet-text-classification-api-preview/>

Radford, A. (2018), ‘Improving language understanding with unsupervised learning’.

URL: <https://openai.com/research/language-unsupervised>

- Řehůřek, R. and Sojka, P. (2010), Software Framework for Topic Modelling with Large Corpora, *in* ‘Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks’, ELRA, Valletta, Malta, pp. 45–50. <http://is.muni.cz/publication/884893/en>.
- Rogel-Salazar, J. (2022), ‘Transformers models in machine learning: Self-attention to the rescue’. Accessed: 2023-4-19.
URL: <https://www.dominodatalab.com/blog/transformers-self-attention-to-the-rescue>
- scikit-learn developers (n.d.), ‘Clustering’. Accessed: 2023-4-27.
URL: <https://scikit-learn.org/stable/modules/clustering.html>
- Shah, F. P. and Patel, V. (2016), A review on feature selection and feature extraction for text classification, *in* ‘2016 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)’, pp. 2264–2268.
- Shalev-Shwartz, S. and Zhang, T. (2013), ‘Stochastic dual coordinate ascent methods for regularized loss minimization’.
- Shannon, C. E. (1948), ‘A mathematical theory of communication’, *Bell System Technical Journal* **27**(3), 379–423.
URL: <https://onlinelibrary.wiley.com/doi/abs/10.1002/j.1538-7305.1948.tb01338.x>
- Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I. and Fergus, R. (2013), ‘Intriguing properties of neural networks’.
URL: <https://arxiv.org/abs/1312.6199>
- Thomas, Baumanns and Dr. Philipp-Stephan, Freber and Dr. Kai-Stefan, Schober and Dr. Florian, Kirchner (2016), ‘Bauwirtschaft im Wandel - Trends und Potenziale bis 2020’.
- Thomas, L., Yoshinobu, A., James, F., Juha, H., Kari, K., Kent, R., Stefan, R. and buildingSMART International Ltd. (2007), ‘IfcMaterialList’. Accessed: 2023-1-31.
URL: <https://standards.buildingsmart.org/IFC/RELEASE/IFC2x3/TC1/HTML/ifcmaterialresource/lexical/ifcmateriallist.htm>
- Uysal, A. K. and Gunal, S. (2014), ‘The impact of preprocessing on text classification’, *Information Processing & Management* **50**(1), 104–112.
URL: <https://www.sciencedirect.com/science/article/pii/S0306457313000964>

van Berlo, L. (2022), ‘IFC Specifications Database’. Accessed: 2023-2-22.

URL: <https://technical.buildingsmart.org/standards/ifc/ifc-schema-specifications/>

Wang, J., Wang, Z., Zhang, D. and Yan, J. (2017), Combining knowledge with deep convolutional neural networks for short text classification, *in* ‘Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17’, pp. 2915–2921.

URL: <https://doi.org/10.24963/ijcai.2017/406>

Xbim Ltd. (n.d.), ‘Xbim toolkit’. Accessed: 2023-1-31.

URL: <https://docs.xbim.net/research/history.html>

Zhu, Q. and Luo, J. (2022), ‘Generative pre-trained transformer for design concept generation: An exploration’, *Proceedings of the Design Society* **2**, 1825–1834.