Bachelorarbeit

Vorhersage des Privatbesitzes von Straßen mit Hilfe von Machine Learning

Christoph Röhrl

Gutachterin: Prof. Dr. Hannah Bast
Betreuer: Dr. Bruno Becker,
Dipl.-Inf. Michael Schlenk

Albert-Ludwigs-Universität Freiburg
Technische Fakultät
Institut für Informatik
Professur für Algorithmen und Datenstrukturen

02. Februar, 2023

Bearbeitungszeitraum

29. 01. 2023 – 02. 05. 2023

Gutachterin

Prof. Dr. Hannah Bast

Betreuer

Dr. Bruno Becker, Dipl.-Inf. Michel Schlenk

Erklärung

Hiermit erkläre ich, dass ich diese Abschlussarbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen/Hilfsmittel verwendet habe und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe. Darüber hinaus erkläre ich, dass diese Abschlussarbeit nicht, auch nicht auszugsweise, bereits für eine andere Prüfung angefertigt wurde.

Freiburg, 30.03.2023

Ort, Datum

C. R

Unterschrift

Zusammenfassung

Im Bereich der Planung von Glasfaserkabelverlegung spielt es eine große Rolle, ob sich eine Straße in Privatbesitz befindet oder nicht, da die Verlegung in privaten Straßen oftmals mit höheren Kosten einhergeht als in öffentlichen Straßen. Diese Information, also wem eine Straße gehört, kann oftmals nicht oder nur mit großem Aufwand herausgefunden werden. In dieser Arbeit untersuchen wir, inwiefern sich glaubwürdige Vorhersagen über den Privatbesitz einer Straße machen lassen. Hierfür werden wir öffentlich zugängliche Straßendaten von OpenStreetMap sowie verschiedene Machine Learning Methoden benutzen. Für das Trainieren und Testen der Machine Learning Modelle erstellen wir einen Referenzdatensatz mit Straßen der Stadt Heidelberg. Diesen Datensatz ergänzen wir mithilfe von offiziellen Straßendaten des städtischen Tiefbauamtes um die Information des Privatbesitzes. Mit diesen Daten trainieren wir zwei Machine Learning Modelle und evaluieren die Ergebnisse anschließend ausführlich, wobei beide Modelle miteinander verglichen und ihre Leistungen begutachtet werden. Zuletzt arbeiten wir noch die Aussagekraft der verschiedenen Straßenattribute heraus.

Inhaltsverzeichnis

1 Einleitung

1.1 Problembeschreibung

Mit einem Anteil von nur 7,1 % landesweit im Jahr 2021 [1] befindet sich Deutschland bei dem Ausbau von Glasfaserkabelanschlüssen noch in einem frühen Entwicklungsstadium. Dennoch steigt der Bedarf nach schnellen und verlässlichen Internetanschlüssen immer mehr an [2], weshalb ein möglichst rascher Ausbau der Glasfaserinfrastruktur wünschenswert ist. Dem Anschluss von Gebäuden an das Glasfasernetz geht ein aufwendiger Planungsprozess vorher. In diesem Prozess werden Abwägungen gemacht, ob eine Erschließung bestimmter Gebiete und Gebäude überhaupt durchführbar ist und, wenn ja, zu welchen Kosten.

Ein Faktor, welcher die Kosten und Komplexität eines Ausbauprojektes signifikant erhöht, ist der Privatbesitz von Straßen. Der Privatbesitz einer Straße hat oftmals die Folge, dass für ihre Verwaltung und Instandhaltung eine private Person oder ein privates Unternehmen verantwortlich ist. Für den Ausbau von Glasfaserinfrastruktur bedeuten solche Straßen deshalb einen erhöhten Bürokratie- und Planungsaufwand sowie zusätzlich entstehende Kosten, welche aus den individuellen Vereinbarungen mit den Eigentümer:innen hervorgehen. Es ist bei einem Glasfaser-Ausbauprojekt daher sinnvoll, Geräte wie z.B. Verteilerkästen nicht in Straßen, welche sich im Privatbesitz befinden, zu Planen bzw. zu Platzieren.

Die meisten der sich im Privatbesitz befindenden Straßen sind der Öffentlichkeit gewidmet. Öffentlich gewidmete Straßen sind solche, die für die Öffentlichkeit zugänglich sind und auf denen die Straßenverkehrsordnung gilt [3]. Wenn im Laufe dieser Arbeit also von 'privaten' Straßen die Rede ist, bezieht sich 'privat' lediglich auf die Besitztumsverhältnisse und nicht auf eventuell eingeschränkte Zugangsberechtigungen. Strukturell und vom Erscheinungsbild her unterscheiden sich private Straßen also meist wenig von solchen, die nicht privat sind. Dies hat zur Folge, dass man einer Straße durch einfache Betrachtung meist nicht ansieht, ob sie in Privatbesitz ist oder nicht.

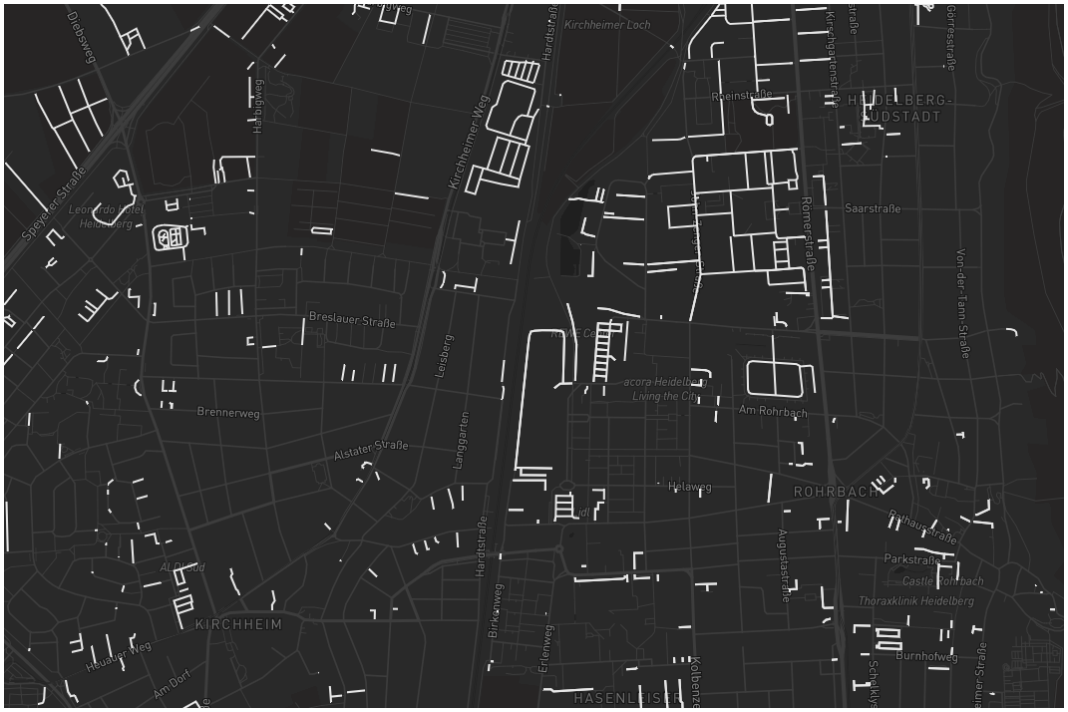

Abbildung 1: Auszug von privaten Straßen in Heidelberg

Abbildung 1 verdeutlicht diese Problematik. Das ermitteln der Besitztumsverhältnisse aller Straßen in einem bestimmten Gebiet durch Kontakt mit Behörden ist jedoch sehr aufwendig und zeitintensiv. Im Rahmen einer frühen Machbarkeits- bzw. Kostenabschätzung ist es daher nicht wirtschaftlich und praktikabel, diese Informationen schon im Vorhinein einzuholen.

1.2 Ziel der Arbeit

In dieser Arbeit untersuchen wir, inwiefern sich der Privatbesitz von Straßen vorhersagen lässt, um so den Planungsprozess von Glasfaser-Ausbauprojekten einfacher und effizienter zu gestalten. Hierfür nutzen wir unterschiedlichste Straßenattribute (wie Länge, Breite, Oberfläche etc.) sowie zwei verschiedene Machine Learning Methoden. Grundlage des Datensatzes bilden die Straßen der Stadt Heidelberg. Mittels offizieller Daten der Stadt wird der Datensatz um eine Grundwahrheit, also der Information über die tatsächlichen Besitztumsverhältnisse der Straßen, ergänzt. Diesen Datensatz verwenden wir anschließend, um ein neuronales Netzwerk sowie ein Gradient-Boosting Modell zu trainieren. Die Trainings Ergebnisse unterziehen wir daraufhin einer um-

fangreichen Evaluation, in der die Methoden miteinander verglichen und ihre Leistung inspiziert werden. Des Weiteren arbeiten wir noch heraus, welche Straßenattribute im Bezug auf Privatbesitz die meiste Aussagekraft haben. Zuletzt geben wir noch einen Ausblick sowie Verbesserungsvorschläge, um die hier gezeigte Methodik für die Praxis zu optimieren.

1.3 Bestehende Lösungen

Eine Lösung, welche genau das oben beschriebene Problem abdeckt, konnten wir nicht finden. Dennoch gibt es einige andere Arbeiten, welche mittels Machine Learning OpenStreetMap Daten verarbeiten und verbessern:

In einem Artikel [4] beschreiben Sajjad Hassany Pazoky und Parham Pahlavani ein multi-klassifizierer System, um OpenStreetMap Straßentypen vorherzusagen. Dafür trainierten sie gängige Machine Learning Modelle anhand von 14 geometrischen Parametern. Die besten Ergebnisse erzielten sie dabei mit einer Kombination aus einem Random Forest und einer Support Vector Machine. Mit einer Genauigkeit von 97 % konnten sie damit qualitativ hochwertige Vorhersagen generieren.

In einem weiteren Artikel [5] nutzten Daniel Feldmeyer, Claude Meisch, Holger Sauter und Joern Birkmann OpenStreetMap Daten und Machine Learning, um sozio-ökonomische Indikatoren zu generieren. So konnten sie zum Beispiel die Zahl von Einwohner:innen von Gemeinden in Baden-Württemberg mit einer Genauigkeit von bis zu $+-$ 200 vorhersagen. Auch hier kamen leistungsstarke Machine Learning Methoden wie Neuronale Netzwerke zum Einsatz.

Zudem untersuchten Sabine Storandt und Stefan Funke in einem weiteren Artikel [6], wie sich relevante Orte durch Dekonstruktion und Interpretation ihres Namens automatisch taggen lassen.

2 Datenquellen

Für diese Arbeit kommen zwei Datenquellen zum Einsatz: Daten von dem Open-Source Projekt 'OpenStreetMap' sowie amtliche Daten der Stadt Heidelberg. Im Folgenden werden wir die zwei Quellen genauer betrachten und dabei ihre Stärken und Schwächen herausarbeiten.

2.1 Straßendaten von OpenStreetMap

OpenStreetMap (kurz OSM) ist ein seit dem Jahr 2004 bestehendes internationales Projekt mit dem Ziel, eine für jeden Menschen frei zugängliche Weltkarte zu erschaffen [7]. Die Karte und die dazugehörigen Daten können von jedem/jeder lizenzkostenfrei genutzt und beliebig weiterverarbeitet werden. Im Gegensatz zu anderen Kartenanbietern wie beispielsweise Google Maps, stellt OSM den Nutzer:innen auch die zugrunde liegenden Rohdaten zur Verfügung. Dies erlaubt es uns, eine Straße nicht nur auf einer Karte zu betrachten, sondern auch die mit ihr assoziierten Eigenschaften zu untersuchen.

In OSM ist eine Straße ein Objekt mit einer eindeutigen ID und weiteren Attributen, sogenannte 'Tags'. Diese Tags werden unter anderem dazu verwendet, die Eigenschaften der Straße zu beschreiben. Welche Tags bzw. Eigenschaften wir in dieser Arbeit untersuchen und nutzen, ist in Tabelle 1 aufgelistet.

2.1.1 Qualität der Daten

Die Daten für das OpenStreetMap Projekt werden ehrenamtlich von Mitgliedern der OSM-Community (sog. 'Mapper') gesammelt und eingetragen. Das hat zur Folge, dass die Qualität und Abdeckung der eingetragenen Daten von Region zu Region und manchmal sogar von Straße zu Straße starken Schwankungen unterliegt. Vor allem Tags wie Höchstgeschwindigkeit, Oberfläche oder Breite sind desöfteren fehlerhaft oder meist gar nicht eingetragen.

Eigenschaft	Datentyp	Bemerkung
OSM-ID	Int	Eindeutige Identifikationsnummer für jedes OSM-Objekt
Geometrie	WKT	Geometrie der Straße im 'Well Known Text ' Format
Straßentyp	Kategorie	Art der Straße, also zb. "Autobahn" oder "Feldweg"
Oberfläche	Kategorie	Oberflächenart bzw. Beschafenheit, z.B. "Asphalt"
Einbahnstraße	Bool	Gibt an, ob eine Straße eine Einbahnstraße ist
Länge	Float	Die Länge der Straße in Meter
Höchstgeschwindigkeit	Float	Die erlaubte Höchstgeschwindigkeit für die Straße
Zufahrtsstraße	Kategorie	Art von Zufahrtsstraße, z.B. "Feuerwehrzufahrt"
Fahrstreifen	Int	Anzahl an Fahrstreifen von der Straße
Zugang	Kategorie	Mögliche Zugangsbeschränkung, z.B. 'Für Autos gesperrt'
Gehweg	Int	Gibt an wie viele Gehwege die Straße besitzt
Breite	Float	Gibt an wie Breit die Straße ist

Tabelle 1: Die im Rahmen dieser Arbeit untersuchten Straßeneigenschaften

Eine weitere Eigenschaft der Daten, welche wir beachten müssen, ist die Unklarheit bei fehlenden Werten. Tags wie z.B. "oneway" (Einbahnstraße) werden von den meisten Mappern nur dann gesetzt, wenn die vorliegende Straße auch eine Einbahnstraße ist. Eine Zuweisung "oneway=false" findet nur sehr selten statt. Diese Art des Mappings hat Auswirkungen auf die Annahmen, die wir im Bezug auf die Daten treffen können. Betrachten wir also eine Straße, welche keinen expliziten oneway-Tag gesetzt hat, können wir nicht wissen, ob die Straße keinen Wert hat, weil sie keine Einbahnstraße ist oder weil sie beim Mapping vergessen wurde.

Diese Unklarheit bei fehlenden Werten ist auch der Grund, warum der in OSM schon existierende Tag "ownership=private" für diese Arbeit nicht zum Einsatz kommt. Zumal ist es unwahrscheinlich, dass Community-Mapper Zugang zu verlässlichen Informationen über die Besitztumsverhältnisse von Straßen haben, was die Qualität des Tags zusätzlich infrage stellt.

2.1.2 Bezug der Daten

Es gibt verschiedene Arten und Wege, die Daten von OpenStreetMap zu beziehen. So ist es beispielsweise möglich, den kompletten Datensatz (also alle Kartendaten des gesamten Planeten) von der OpenStreetMap Website als XML Datei herunterzuladen [8]. Zusätzlich gibt es auch die Möglichkeit, über die sog. 'Overpass'-API selektiv und durch eine Anfrage spezifiziert, Daten von OSM herunterzuladen [9].

Für unseren Zweck machen wir Gebrauch von QLever, einer SPARQL Engine, welche vom Lehrstuhl Algorithmen und Datenstrukturen der Universität Freiburg entwickelt und zur Verfügung gestellt wird [10]. SPARQL (SPARQL Protocol And RDF Query Language) ist eine Abfragesprache für das Abfragen von Daten im Resource Description Framework (RDF) Format [11]. Daten im RDF Format werden als Triple der Form (Subjekt, Prädikat, Objekt) abgebildet. Zwei Beispiele hierfür im Kontext von Straßen sind:

[**Kaiser-Joseph-Straße** *liegt in* 'Freiburg']
oder
[**OSM-Objekt-ID** *besitzt Tag* 'Einbahnstraße']

Mit QLever können wir nun einen in RDF konvertierten OSM-Datensatz abfragen. Aus Effizienzgründen beschränken wir uns dabei auf einen Datensatz, welcher lediglich die OSM Daten von Deutschland enthält. Die SPARQL Anfrage für QLever sieht wie folgt aus:

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX ogc: <http://www.opengis.net/rdf#>
PREFIX osm: <https://www.openstreetmap.org/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX osmkey: <https://www.openstreetmap.org/wiki/Key:>
PREFIX osmrel: <https://www.openstreetmap.org/relation/>
SELECT ?osm_id ?geometry ?highway ?surface ?oneway ?length ?maxspeed ?service
       ?lanes ?sidewalk ?access ?width WHERE {
  ?osm_id osmkey:highway ?highway .
  OPTIONAL {?osm_id osmkey:surface ?surface} .
  OPTIONAL {?osm_id osmkey:oneway ?oneway} .
  OPTIONAL {?osm_id osmkey:length ?length} .
  OPTIONAL {?osm_id osmkey:maxspeed ?maxspeed} .
  OPTIONAL {?osm_id osmkey:service ?service} .
  OPTIONAL {?osm_id osmkey:lanes ?lanes} .
```

```
  OPTIONAL {?osm_id osmkey:sidewalk ?sidewalk} .
  OPTIONAL {?osm_id osmkey:access ?access} .
  OPTIONAL {?osm_id osmkey:width ?width} .
  ?osm_id rdf:type osm:way .
  ?osm_id geo:hasGeometry ?geometry .
  osmrel:285864 ogc:contains ?osm_id .
}
```

Mit dieser Anfrage geben wir uns Objekt ID's sowie die dazugehörigen Geometrien aus, die vom Typ Straße sind und in der Stadt Heidelberg liegen. Heidelberg wird hierbei durch die Relation 285864 spezifiziert, welche die Gebietsgrenzen der Stadt darstellt. Alle weiteren Tags werden als optional abgefragt, so dass ein Objekt, das einen oder mehrere dieser Tags nicht enthält, trotzdem im Resultat vorkommt. Anstelle eines Wertes wird dann bei den betroffenen Tags ein leerer String eingetragen.

Das Verarbeiten der Anfrage dauert nur wenige Sekunden. Das Resultat können wir nun als CSV-Datei herunterladen und weiterverarbeiten.

2.2 Straßendaten der Stadt Heidelberg

Neben den öffentlich zugänglichen Daten von OpenStreetMap benutzen wir in dieser Arbeit auch noch amtliche Daten der Stadt Heidelberg. Zur Verfügung gestellt werden die Daten vom Tiefbauamt Heidelberg, welches für Neubau, Unterhaltung und Instandsetzung von öffentlichen Verkehrsflächen der Stadt zuständig ist [12]. Die Daten liegen in Form einer Shape Datei vor und beinhalten alle Straßen-Verkehrsflächen, welche zum Grundvermögen der Stadt gehören. Mithilfe von diesen Informationen, also welche Straßen zum Grundvermögen der Stadt gehören und welche nicht, können wir später unseren Datensatz um die Grundwahrheit über den Privatbesitz von Straßen ergänzen. Die klare Stärke dieser Datenquelle liegt also in ihrer Amtlichkeit, wodurch glaubhafte Aussagen über die Besitztumsverhältnisse von Straßen in Heidelberg überhaupt erst möglich werden. Als Schwäche wäre an dieser Stelle lediglich die Genauigkeit der Geometrien in der Shape Datei zu nennen, was jedoch keine Problem für unser Vorhaben darstellt.

3 Bearbeiten der Daten

Um mit den OpenStreetMap Straßendaten Machine Learning Modelle trainieren zu können, müssen wir die Daten zuvor noch anpassen und ergänzen. Hierfür erarbeiten wir eine Grundwahrheit über den Privatbesitz und berechnen aus den Geometrien der Straßen ein weiteres boolsches Attribut "Sackgasse". Des Weiteren bereinigen wir die Daten und entfernen einzelne Werte, welche für den Machine Learning Algorithmus ungeeignet sind.

3.1 Erstellen einer Grundwahrheit

Unser Ziel ist es, einen Datensatz zu erstellen, welcher eine Grundwahrheit in Bezug auf den Privatbesitz von Straßen enthält. Die in OSM vorhandenen Straßendaten ergänzen wir also mithilfe der Daten des Tiefbauamt Heidelberg um ein neues Attribut "private property" (Privateigentum).

Für dieses Herausarbeiten der Grundwahrheit verwenden wir die Geometrien der Straßen. Einen Großteil der Arbeit erledigen wir dabei mit QGis, einer Open-Source Geosoftware zum Erstellen, Bearbeiten, Anzeigen und Analysieren von räumlichen Daten [13]. Das Grundprinzip ist hierbei wie folgt:
Wir legen die Straßengeometrien von OpenStreetMap und die Straßengeometrien des Tiefbauamt Heidelberg übereinander und entfernen all diejenigen Straßen, welche sich überlappen. Die übrigbleibenden OSM Straßen können wir dann in unserem Trainingsdatensatz als privat markieren.

Um die über QLever bezogenen OSM Daten in QGis zu laden, müssen diese zuvor noch in eine zusammenhängende Geojson Datei konvertiert werden. Hierfür iterieren wir mit einem Pythonskript über alle Geometrieeinträge der CSV-Datei und fügen diese in ein Geojson-Template ein.

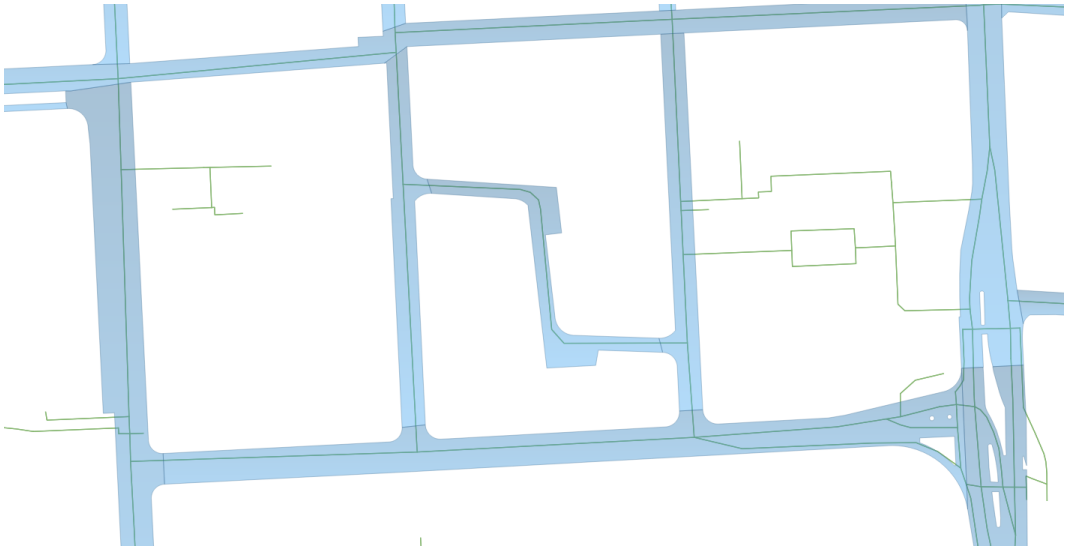

Abbildung 2: Auszug von OSM Straßen (Grün) überlagert mit der Shape Datei (Blau) des Tiefbauamtes Heidelberg

Anschließend laden wir die OSM Geojson-Datei und die Shape-Datei des Tiefbauamtes als zwei separate Layer in QGis. Wichtig hierbei ist, dass wir den beiden Layern das gleiche Koordinatenbezugssystem (KBS) zuweisen, damit beide auch auf die gleiche Weise an dieselbe Stelle der Karte projiziert werden. Wir verwenden für diesen Zweck das weitverbreitete WGS84 System, welches unter anderem auch die vermessungstechnische Grundlage für GPS bildet [14]. Abbildung 2 zeigt einen Ausschnitt der sich überlagernden Layer.

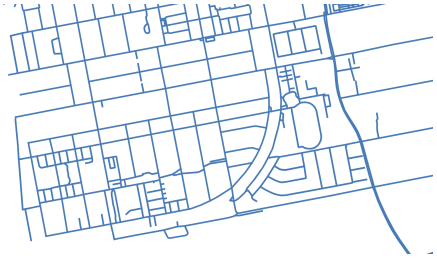

Abbildung 3: Vor dem Entfernen

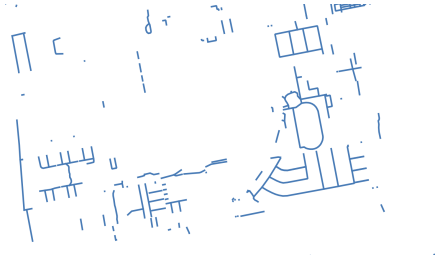

Abbildung 4: Nach dem Entfernen

In einem nächsten Schritt entfernen wir aus dem OSM Layer alle Straßen, welche von dem Tiefbauamt Layer überlagert werden. Übrig bleiben diejenigen Straßen, welche sich in Privatbesitz befinden. Abbildungen 3 und 4 zeigen diesen Schritt.

Es gibt jedoch noch eine Reihe an Randfällen, welche wir beachten müssen. So müssen wir Waldwege der Stadt separat entfernen, da sie in der Straßen-Shape Datei des Tiefbauamtes nicht enthalten sind. Hierfür nutzen wir zwei weitere Flächen-Shape Dateien, welche die Forstflächen der Stadt darstellen. Wie schon zuvor entfernen wir alle Straßen des OSM Layers, welche von diesen Shape Dateien überlagert werden. Weitere kleine Randfälle, welche z.B. durch den Cut-Off an Gebietsgrenzen zustande kommen, entfernen wir händisch über das ”Selection”-Tool von QGis. Ein weiterer bemerkenswerter Randfall sind Straßen, welche weder der Stadt Heidelberg gehören noch in Privatbesitz sind. Dazu zählen vor allem Autobahnen und Bundesstraßen, welche durch das Stadtgebiet von Heidelberg verlaufen. Damit diese nicht fälschlicherweise als privat markiert werden, müssen wir auch sie noch zusätzlich entfernen. Das Entfernen dieser Straßen erfolgt jedoch nicht mittels sich überlagernder Layer oder dem Selection Tool, sondern anhand ihres Typs. Da auch noch andere Straßentypen aus dem Datensatz entfernt werden müssen, erfolgt dieser Schritt später in ”Bereinigen der Daten”.

3.2 Berechnen von Sackgassen

Beim Betrachten der sich im Privatbesitz befindenden Straßen fällt auf, dass viele von ihnen eine Sackgasse sind. Wie schon für den Privatbesitz existiert auch für Sackgassen in OSM ein Tag ”noexit”, welcher aber ebenfalls nur selten benutzt wird und daher für uns unbrauchbar ist. Aus diesem Grund berechnen wir dieses Attribut erneut aus den Geometrien der Straßen. Das Grundprinzip der Sackgassenerkennung ist dabei wie folgt: Jede Straßengeometrie besteht aus einer Menge an Koordinatenpunkten,

Abbildung 5: Straßen im Privatbesitz

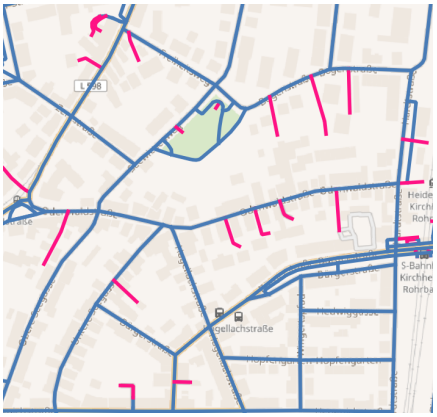

Abbildung 6: Errechnete Sackgassen

hier genannt 'Position'. Manche Straßengeometrien teilen sich solche Positionen mit anderen, z. B. wenn sie einen zusammenhängenden Pfad bilden oder sich kreuzen. Wenn wir also eine Position haben, welche der Start- oder Endpunkt einer Geometrie ist und gleichzeitig in keiner anderen vorkommt, wissen wir, dass die Geometrie an dieser stelle eine Sackgasse darstellt.

Realisiert wird die Sackgassenerkennung durch ein Pythonskript, welches durch alle Geometrien iteriert und die einzelnen Positionen in einem Dictionary abspeichert. Die Positionen selbst werden als Schlüssel verwendet und der Wert ist eine Liste von Index Nummern derjenigen Geometrien, welche den Schlüssel (also die Position) beinhalten. So speichern wir für jede Position die Geometrien, die sie enthalten. Im nächsten Schritt iterieren wir durch das Dictionary und schauen uns alle Schlüssel-Wert Paare an, für die die Länge der Liste 1 beträgt. Durch einfache If-else Abfragen überprüfen wir, dass es sich bei der Position um einen Start- oder Endpunkt handelt und dass die Geometrie kein Kreis ist. Zuletzt stellen wir noch sicher, dass sich die Position nicht in einem von uns zuvor erstellten Puffer-Bereich rund um die Gebietsgrenzen befindet, um abgeschnittene Straßen nicht fälschlicherweise als Sackgasse zu markieren. Sind all diese Bedingungen erfüllt, setzen wir mithilfe der zuvor gesammelten Indexnummer den Wert "simple deadend" in unserem Datensatz auf True. Abbildungen 5 und 6 zeigen im selben Ausschnitt jeweils private Straßen und die errechneten Sackgassen.

3.3 Bereinigen der Daten

Um die OSM Straßendaten für Machine Learning verwenden zu können, müssen wir diese noch von Fehlern und Inkonsistenzen bereinigen. Da, wie oben schon erwähnt, die Daten von ehrenamtlich arbeitenden Community-Mitgliedern gesammelt werden, können sie teilweise falsche und unbrauchbare Tag-Werte enthalten. Ein Beispiel hierfür ist der Wert ”asphalt;compacted” für den Tag Surface, welcher im gesamten Gebiet von Heidelberg nur einmal vorkommt. Er ist aufgrund seiner Seltenheit für das Training unbrauchbar und wird daher von uns mit einem Platzhalter Wert 'unknown' ersetzt. Ebenso kommen in den Daten auf ihren Kontext bezogen, sinnfreie Werte, wie z.B. 'excelent' für ”surface” vor, obwohl der Tag die Art der Oberfläche beschreiben soll, nicht ihre subjektive Qualität. Werte wie diese ersetzen wir ebenfalls durch 'unknown'.

Für das Bereinigen der Daten verwenden wir Pandas [15], eine Open Source Bibliothek für Python, welche es uns ermöglicht, Daten in tabellarischer Form einfach zu manipulieren und zu analysieren. Mithilfe von Pandas gehen wir also durch jedes Attribut und ersetzen oder transformieren problematische Werte. Im folgenden Abschnitt sehen wir die Schritte für die Attribute ”length” und ”maxspeed”:

```
# Set length attribute
# This length does not represent the "real length" because its measured
# on a flat plane. For the purpose of training, this is fine because
# only the relative distances matter.
df['length'] = df['geometry'].apply(
                    lambda x: shapely.wkt.loads(x).length * 1000
                )
df["length"] = df["length"].astype(float)

# Maxspeed
df["maxspeed"].replace("walk", 5, inplace=True)
df["maxspeed"].replace("none", 50, inplace=True)
df["maxspeed"].replace("", 50, inplace=True)
df["maxspeed"] = df["maxspeed"].astype(int)
```

Für das Attribut 'length' ersetzen wir die sehr lückenhaft getaggten Werte komplett durch eine anhand der Geometrie berechneten Länge. Dabei kommt nur die euklidische

Distanz zwischen den Stützpunkten der Straße zum Einsatz, die Krümmung der Erde sowie die Unterschiede der Abstände von Längen und Breitengrade werden ignoriert. Der errechnete Wert stellt also nur eine Annäherung an die tatsächliche Länge dar, welche für unser Vorhaben jedoch ausreichend ist. Dieser Schritt ist sinnvoll, da zum einen die tatsächliche Länge der Straße gar nicht so relevant ist, sondern eher die relativen Längenunterschiede zwischen den Straßen, und zum anderen da wir so eine Werte Abdeckung von 100% erreichen. Anschließend konvertieren wir noch alle Werte von 'length' zum Typ float.

Für das Attribut 'maxspeed' existieren in den Daten neben einer Reihe an integer Werten auch die Werte "walk", "none" und "" (leerer String). Diese ersetzen wir durch sinnvolle integer Werte. Dabei treffen wir die Annahme, dass innerhalb des Stadtgebiets höchstens 50 km/h gefahren werden darf (Abgesehen Bundesstraßen und Autobahnen, welche später noch aus den Daten entfernt werden). Anschließend konvertieren wir noch alle Werte von 'maxspeed' zum Typ int.

Neben den Attributen der Straßen müssen wir auch noch die Straßendaten selbst bereinigen: In OSM wird grundsätzlich jeder "Weg" als Straße (bzw. Highway) bezeichnet. Das schließt auch z. B. Trampelpfade und Forstwege mit ein. Im Kontext der Glasfaserplanung sind nur eine bestimmte Art von Straßen von Interesse, und zwar solche welche in der Nähe von Häusern sind und auch mit einem Auto befahren werden können.

Typische Beispiele hierfür sind:

- Eine Straße in einem Wohngebiet
- Ein Feldweg, an dessen Rand Häuser stehen
- Eine innerstädtische Hauptverkehrsstraße, an deren Rand Häuser stehen

Typische Beispiele von Straßen/Wegen welche explizit keine Rolle spielen sind:

- Ein Trampelpfad im Wald
- Nur zu Fuß begehbare Wege/Pfade im Allgemeinen
- Autobahnen, Schnellstraßen
- ... sowie andere begehbare Strukturen, welche nicht im Sinne der oben genannten Einschränkungen "Straßen" sind.

Auf genau solche Arten von Straßen (”innerstädtische Kraftfahrtstraßen”) bezieht sich auch der Datensatz der Stadt Heidelberg, was bedeutet, dass wir für andere Straßenarten auch keine Grundwahrheit erstellen können. Alle Straßen/Pfade/Wege, welche keine ”innerstädtische Kraftfahrtstraßen” sind, spielen für uns also keine Rolle. Wir entfernen sie mit Pandas anhand ihres Typs. Dies schließt auch die schon zuvor erwähnten Autobahnen und Bundesstraßen mit ein.

Eine wichtige Annahme, die wir dabei machen, ist folgende: Da Straßen einer der wichtigsten Grundbausteine von OpenStreetMap sind, hat hier der Tag ”Highway=” eine deutlich bessere Qualität als andere. Sprich, während Tags wie ”width", ”access” oder ”length” oftmals unvollständig und gelegentlich sogar falsch getaggt sind, ist die Abdeckung von Straßen und die Kategorie, in die sie fallen, deutlich besser. Wir gehen also davon aus, dass z.B. ’living streets’ oder ’residential streets’ im Sinne unserer obigen Definition (”innerstädtische Kraftfahrtstraßen”) nicht im größeren Stil fälschlicherweise als ’motorway’ oder ’footway’ getaggt sind.

4 Trainieren eines Neuronalen Netzwerks

Als erste Machine Learning Methode wählen wir ein künstliches neuronales Netzwerk (KNN). Motivation hierfür sind die vielversprechenden Fortschritte und Erfolge, welche mit dieser Methode in den vergangenen Jahren erzielt werden konnten [16].

Ein künstliches neuronales Netzwerk ist eine Machine Learning Methode, welche die Funktionsweise von Neuronen und ihren Vernetzungen im menschlichen Gehirns nachahmt. Es besteht aus vielen künstlichen Neuronen und Verbindungen, welche in 'Schichten' eingeteilt sind. So gibt es immer eine Eingabeschicht, welche die Input Daten enthält, eine Ausgabeschicht welche die Vorhersage macht und Zwischenschichten, so genannte 'Hidden Layers'. Die Anzahl an Neuronen ist in jeder Schicht unterschiedlich. In der Eingabeschicht hängt sie von den Input Daten ab und in der Ausgabeschicht von den möglichen Ergebnissen (im Falle von Klassifikation). Die Anzahl und Größe der Zwischenschichten ist frei wählbar. Die optimalen Werte hierfür stellen sich meist erst im Laufe des Trainings heraus. Abbildung 7 zeigt ein vereinfachtes Schema eines KNN.

Beim Trainieren eines KNN werden die Gewichte der einzelnen Verbindungen zwischen den Neuronen in einem iterativen Prozess so angepasst, dass die Vorhersagen des KNN nach und nach genauer werden. Hierfür werden dem KNN Beispieleingaben gezeigt, zu welchen die korrekte Ausgabe/Klassifizierung schon bekannt ist. Anhand der Unterschiede zwischen dem korrekten "soll" Ausgabewert und der tatsächlichen Ausgabe des Netzwerks werden die Gewichte so angepasst, dass in der nächsten Iteration der Unterschied kleiner ausfällt.

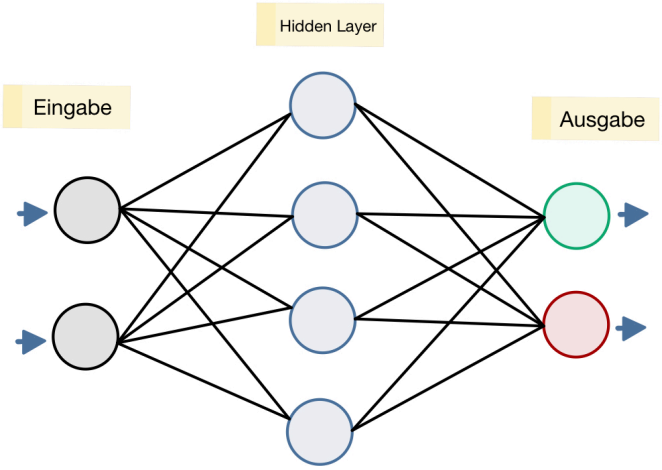

Abbildung 7: Vereinfachtes Beispiel von einem KNN

Da unsere Daten in tabellarischer Form vorliegen, benutzen wir für das Erstellen und Trainieren eines KNN ein Framework namens PyTorch Tabular [17]. PyTorch Tabular erlaubt es uns, einfach und effizient KNN's zu trainieren und zu konfigurieren. Der Dateninput hat hierbei wieder die Form eines Pandas Dataframes, wie wir es auch schon zum Bearbeiten/Bereinigen der Straßendaten verwendet haben. Im Folgenden sind die Konfigurationen unseres Netzwerks zu sehen, eingeteilt in die Bereiche Daten, Training und Model:

```
num_col_names = ['length', 'maxspeed', 'lanes', 'width']
cat_col_names = ['type', 'surface', 'oneway', 'service',
                 'sidewalk', 'access', 'simple_deadend']

data_config = DataConfig(
    target=['private_property'],
    continuous_cols=num_col_names,
    categorical_cols=cat_col_names
)
```

```
trainer_config = TrainerConfig(
    auto_lr_find=True,
    batch_size=128,
    max_epochs=10,
    gpus=None
)

model_config = CategoryEmbeddingModelConfig(
    task="classification",
    layers= "40-40-40",
    activation="LeakyReLU",
    loss="CrossEntropyLoss"
)
```

Im Bereich Daten teilen wir die Attribute in numerische und kategorische Datentypen ein und legen das zu vorhersagende Target fest. Für das Trainieren aktivieren wir die 'auto learning rate finder' Funktion, durch welche die Schrittweite, mit der die Gewichte im KNN angepasst werden, automatisch bestimmt wird. Des Weiteren legen wir die Batchgröße und die maximale Anzahl an Epochen fest, was bestimmt, wie viele Daten das KNN am Stück verarbeitet und wie oft der ganze Datensatz maximal durchlaufen werden soll. Für das Modell konfigurieren wir noch die Art der Vorhersage als Klassifikation, geben die Struktur der Schichten an und legen noch die Art der activation und der loss Funktion fest.

Für das Training teilen wir nun unseren Datensatz in 90 % Trainingsdaten und 10 % Testdaten auf. Welche Straße dabei in welchem Teildatensatz landen, wird durch eine pseudo Zufallsfunktion von Pandas bestimmt.

Das anschließende Training dauert nur wenige Minuten. Mit dem Testdatensatz können wir das trainierte Modell und seine Fähigkeiten testen:

- Accuracy: 0.82 (Anteil an korrekt klassifizierten Datenpunkten)
- Precision: 0.86 (Anteil an positiven Vorhersagen, welche korrekt waren)
- Recall: 0.85 (Anteil an erkannten tatsächlich positiven Datenpunkten)

Die Ergebnisse analysieren wir ausführlicher im Kapitel 'Evaluation'.

5 Trainieren eines Gradient Boost Models

Als zweite Machine Learning Methode wählen wir ein Gradient Boosting Verfahren implementiert durch die Open Source Softwarebibliothek XGBoost (Extreme Gradient Boost) [18]. Die Motivation für diese Methode sind auch hier wieder die vielen Erfolge, welche mit ihr in den vergangenen Jahren erzielt werden konnten [19].

Das Konzept von Gradient Boosting basiert darauf, schwache Modelle (z. B. kleine Entscheidungsbäume) sequenziell so zu kombinieren, dass daraus ein starkes Modell resultiert. Dabei versucht jedes schwache Modell die Fehler des vorherigen Modells zu korrigieren, wodurch das Gesamtmodell nach und nach genauer wird.

In Abbildung 8 ist eine vereinfachte Version für binäre Klassifikation mit Bäumen der Tiefe eins visualisiert. Zu Beginn wird ein Initialwert für die Wahrscheinlichkeit, dass ein gegebener Datenpunkt 'positiv' ist, festgelegt. Anschließend folgen beliebig viele "schwache" Entscheidungsbäume, welche den Wahrscheinlichkeitswert je nach Eingabe erhöhen oder verkleinern. Dabei wird der Einfluss von jedem Baum noch mit einer Lernrate skaliert, wodurch kleinere Korrekturschritte gemacht werden. Kleine Schritte sind wichtig, da sich das Modell sonst zu sehr an die Trainingsdaten anpasst (overfitting) und nicht mehr für generalisierende Aussagen über neue, ungesehene Datenpunkte geeignet ist. XGBoost berechnet also für jeden Datenpunkt einen Wahrscheinlichkeitswert zwischen null und eins. Ist dieser $>= 0.5$, so wird ein Datenpunkt als positiv klassifiziert, andernfalls als negativ.

Der Dateninput für unser XGBoost Modell ist wieder ein Pandas Dataframe. Da XGBoost mit numerischen Entscheidungsbäumen arbeitet, müssen die kategorischen Attribute zusätzlich noch one-hot codiert werden. Bei einer one-hot Codierung werden alle möglichen Werte eines kategorischen Attributs zu einzelnen boolschen Attributen umgewandelt (siehe Abbildung 9). Pandas hat hierfür eine Funktion 'get_dummies', mit welcher wir diese Umwandlung durchführen.

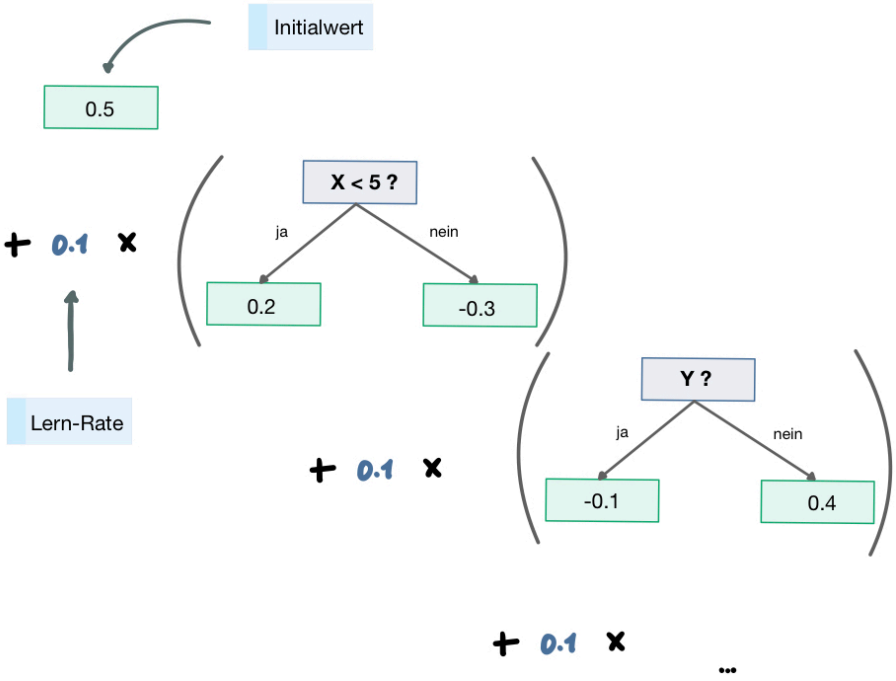

Abbildung 8: Vereinfachtes Beispiel von einem Gradient Boost Verfahren

ID	Farbe
#1	Grün
#2	Blau
#3	Rot
#4	Schwarz

ID	Rot	Grün	Blau	Schwarz
#1	0	1	0	0
#2	0	0	1	0
#3	1	0	0	0
#4	0	0	0	1

Abbildung 9: Prinzip der one-hot Kodierung

Im folgenden ist die Konfiguration für unser XGBoost Modell zu sehen:

```
model = xgb.XGBClassifier(
    objective='binary:logistic',
    scale_pos_weight=2,
    learning_rate=0.1,
    max_depth=20,
    reg_lambda=5,
    gamma=0,
    seed=42
)

model.fit(
    X_train,
    y_train,
    verbose=True,
    early_stopping_rounds=10,
    eval_metric='aucpr',
    eval_set=[(X_test, y_test)]
)
```

Für das Modell legen wir die Aufgabe als binäre Klassifikation fest. Da beim Erstellen der Bäume mitunter Pseudozufall zum Einsatz kommt (die Variation verhindert overfitting), legen wir auch einen Seed fest, damit die Ergebnisse konstant und reproduzierbar sind. Mit 'scale_pos_weight' legen wir fest, wie stark positive Datenpunkte beim Training gewichtet werden sollen. 'Learning_rate' bestimmt, wie oben schon erwähnt, die Schrittweite während 'max_depth' die Tiefe der Bäume limitiert. Mit 'reg_lambda' setzen wir einen Parameter, welcher bestimmt wie sensitiv das Modell auf ungewöhnliche Werte, sog. 'outliers' reagiert. Mit 'gamma' regulieren wir, ab wann es sich mit Hinblick auf die Fehlerquote lohnt, einen Baum um eine zusätzliche Ebene zu erweitern.

Mit 'early_stopping_rounds' legen wir fest, nach wie vielen Iterationen ohne signifikante Verbesserung das Training beendet werden soll. Des Weiteren legen wir fest, mit welchen Daten trainiert und mit welchen getestet werden soll.

Die oben gezeigte Konfiguration wurde durch ”Grid Search” bestimmt, einem Verfahren, bei dem alle durch ein Raster festgelegte Kombinationen an Werten systematisch und im Schnelldurchlauf ausprobiert werden.

Das Training dauert auch hier nur wenige Minuten. Die Testergebnisse sind wie folgt:

- Accuracy: 0.78 (Anteil an korrekt klassifizierten Datenpunkten)
- Precision: 0.78 (Anteil an positiven Vorhersagen, welche korrekt waren)
- Recall: 0.88 (Anteil an erkannten tatsächlich positiven Datenpunkten)

Auch diese Ergebnisse analysieren wir ausführlicher im Kapitel 'Evaluation'.

6 Evaluation

Gradient Boosting und KNN sind zwei von Grund auf verschiedene Verfahren. Während bei Gradient Boosting viele 'schwache' Modelle wie Entscheidungsbäume zu einem 'starken' Modell zusammengefasst werden, simulieren KNN die Neuronen und ihre Vernetzungen des menschlichen Gehirns. Gradient Boosting 'lernt' also durch das sequenzielle Aneinanderhängen von gewichteten Bäumen, während bei KNN iterativ die Gewichte der Neuronenverbindungen angepasst werden. Im Folgenden vergleichen wir die beiden Modelle miteinander, untersuchen welche Attribute den größten Einfluss auf die Vorhersagen haben und geben einen Ausblick, wie sich die Modelle weiter optimieren lassen.

6.1 Leistungsvergleich

In den vorangegangenen Kapiteln haben wir sowohl ein künstliches neuronales Netzwerk (implementiert mit PyTorch Tabular [17]) als auch ein Gradient Boosting Modell (implementiert mit XGBoost [18]) trainiert. Die Ergebnisse hiervon sind in Tabelle 2 aufgelistet.

Die Accuracy Metrik beschreibt, wie oft ein Modell anteilig mit seinen Vorhersagen richtig liegt. Hohe Werte sind also erstrebenswert, jedoch reicht diese Metrik alleine nicht aus, um die Leistung eines Modells im Kontext einer bestimmten Problemstellung ausreichend bewerten zu können. Beispielsweise bei einem unausgeglichenen Datensatz, in welchem nur 5% der Datenpunkte positiv sind, ist es für einen Machine

	Pytorch Tabular	XGBoost
Accuracy	0.82	0.78
Precision	0.86	0.78
Recall	0.85	0.88

Tabelle 2: Ergebnisse des Trainings beider Methoden im Vergleich

Learning Algorithmus ein Leichtes eine Accuracy von 95% zu erreichen, indem er konstant eine negative Antwort vorhersagt. In unserem Trainingsdatensatz ist das Verhältnis von positiven zu negativen Datenpunkten 62% zu 38%. Mit Accuracy Werten von je 82% und 78% haben also unsere Modelle auf jeden Fall etwas gelernt, was über die stumpfe Vorhersage von ein und derselben Antwort hinaus geht.

Die Precision Metrik beschreibt, wie oft die positiven Vorhersagen eines Modells richtig sind. Hohe Werte bedeuten also, dass das Modell wenige falsch-positive Vorhersagen macht. Auch hier gilt wieder, dass die Metrik meist nur in Verbindung mit anderen Metriken wirklich nützlich ist. So könnte ein Modell beispielsweise nur sehr selten und nur wenn es sich ganz sicher ist, positive Vorhersagen machen. Dies würde zwar in einer hohen Precision resultieren, aber auch für eine sehr schlechte Accuracy sorgen. Mit einem Precision Wert von 86% schneidet das KNN hier deutlich besser ab als das XGBoost Modell mit 78%.

Die Recall Metrik gibt an, wie viele der tatsächlich positiven Datenpunkte das Modell auch als solche klassifiziert hat. Ein hoher Wert bedeutet demnach, dass das Modell positive Datenpunkte gut erkennt. Auch hier gilt es wieder das Ganze in den Kontext von anderen Metriken zu setzen, da auch das konstante Vorhersagen einer positiven Antwort zu einem guten Recall führt (jedoch zu einer schlechten Accuracy). Mit Recall Werten von 85% und 88% schneiden beide Modelle hier fast gleich gut ab.

Sowohl für das künstliche neuronale Netzwerk als auch für das Gradient-Boosting Modell liegen alle drei Metriken grob um die 80%. Daraus können wir schließen, dass beide Modelle zumindest bis zu einem gewissen Grad komplexere Zusammenhänge gelernt haben. Insgesamt scheint das KNN, vor allem was die Gesamtqualität der Vorhersagen angeht, besser abzuschneiden als das Gradient-Boosting Modell.

Eine gute Möglichkeit, um viele Eigenschaften eines Modells auf einmal betrachten zu können, ist die sog. 'Konfusionsmatrix'. Sie gibt die Anteile der richtig-positiv, falsch-positiv, richtig-negativ und falsch-negativ Vorhersagen an. Aus diesen Werten lassen sich unter anderem auch alle eben genannten Metriken berechnen. So lässt sich zum Beispiel die Accuracy als Summe der richtig-positiv und richtig-negativ Werte angeben. Die Abbildungen 10 und 11 zeigen diese Matrix für beide Modelle.

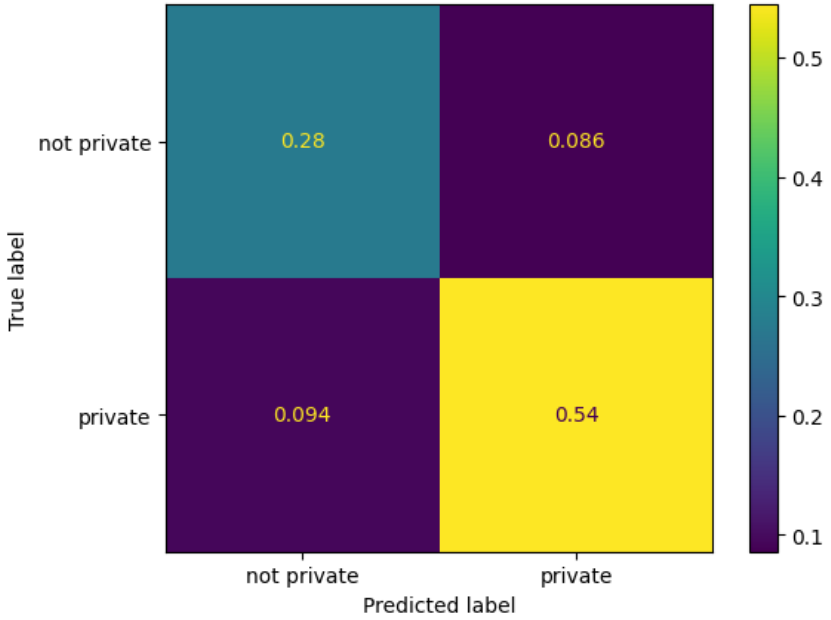

Abbildung 10: Konfusionsmatrix KNN

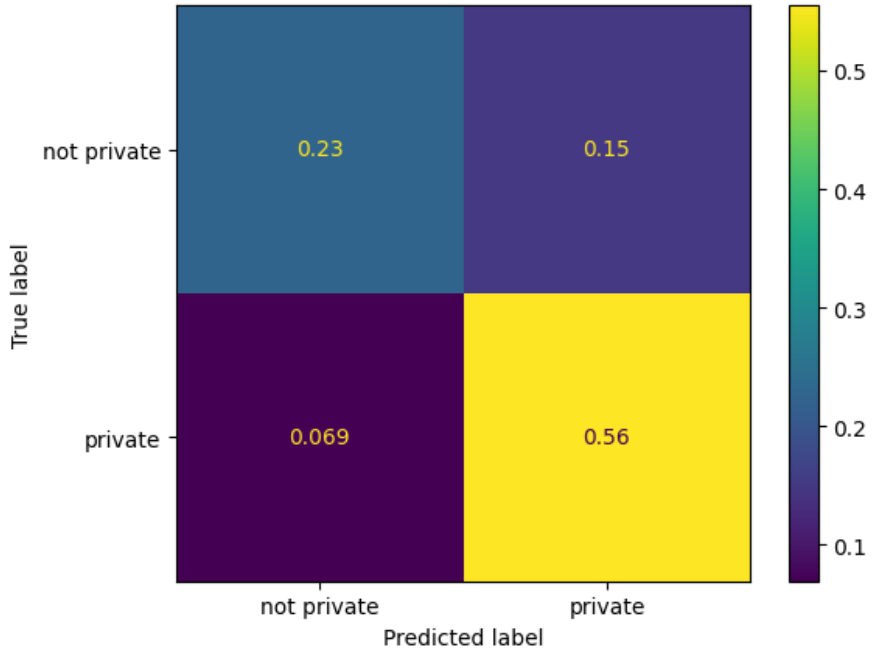

Abbildung 11: Konfusionsmatrix Gradient Boost Modell

6.2 Aussagekraft der verschiedenen Attribute

Um herauszufinden, welche Attribute für unsere Modelle von besonders großer Bedeutung sind, verwenden wir die sog. 'Permutation Feature Importance' [20]. Diese Methode misst den Verlust der Genauigkeit eines Modells, nachdem die Werte eines Attributs in den Testdaten zufällig permutiert wurden. Durch die Permutation wird die Verbindung zwischen dem Attribut und der Grundwahrheit zerstört, wodurch das Attribut für das Modell keinen echten Mehrwert mehr liefert. Ist die Genauigkeit mit den modifizierten Testdaten im Vergleich zu den Originalen deutlich niedriger, so hat das permutierte Attribut eine starke Verbindung zur Grundwahrheit und ist auch wichtig für die Vorhersagen des Modells. Ändert sich die Genauigkeit durch ein permutiertes Attribut kaum, so ist es für das Modell wahrscheinlich nicht von ganz so großer Bedeutung. Wir wählen diese Methode, da sie einfach zu berechnen und auch ohne großen Aufwand nachvollziehbar ist. Hinzu kommt, dass sie auf verschiedenste Arten von Machine Learning Modellen anwendbar ist, was uns auch einen einfachen Vergleich unserer eigenen Modelle erlaubt.

Es ist jedoch auch wichtig zu Erwähnen, dass Permutation Feature Importance einige Schwächen hat: Durch das zufällige Permutieren der einzelnen Attribute hängen auch die Messergebnisse zu einem gewissen Grad vom Zufall ab. Das bedeutet, dass die Ergebnisse von Testlauf zu Testlauf leicht unterschiedlich ausfallen können. Durch mehrmaliges Permutieren und Testen von ein und demselben Attribut und dem Mitteln der Ergebnisse, können wir dem jedoch effektiv entgegenwirken. Eine weitere Schwäche dieser Methode ist ihre Unfähigkeit, mit korrelierten Attributen umzugehen. Wenn ein Attribut permutiert wird, welches stark mit einem anderen korreliert, werden so unrealistische Datenpunkte erzeugt, welche das Messergebnis verzerren. Das bedeutet auch, dass korrelierende Attribute insgesamt eine niedrigere Permutation Feature Importance haben, da das Modell zur korrekten Vorhersage immer auf das nicht permutierte Attribut zurückgreifen kann.

Mit Hinblick auf die eben genannten Nachteile muss man sich also im Klaren sein, dass niedrige Permutation Feature Importance Werte nicht unbedingt die Nutzlosigkeit eines Attributs belegen. Vielmehr wird durch diese Metrik deutlich, welche Attribute auf jeden Fall von besonderer Bedeutung sind.
Tabelle 3 zeigt jeweils die drei Attribute mit der höchsten Permutation Feature Importance beider Modelle.

PyTorch Tabular	XGBoost
type (-0.194)	type_service (-0.129)
service (-0.013)	access_unknown (-0.015)
access(-0.011)	maxspeed (-0.015)

Tabelle 3: Top drei Permutation Feature Importance Scores beider Modelle

Zuerst sticht heraus, dass das Attribut 'type', welches die Art einer Straße angibt, für beide Modelle das Wichtigste ist. Mit jeweils Knapp 20 und 13 Prozent ist bei ihrer Permutation der größte Genauigkeitsverlust festzustellen. Hierbei ist es jedoch wichtig zu verstehen, dass type_service Teil der one-hot Codierung von dem Attribut type und nicht dasselbe wie 'type' ist. Während bei PyTorch Tabular 'type' ein kategorisches Attribut mit mehreren möglichen Werten ist, ist 'type_service' in XGBoost ein boolsches Attribut, welches angibt, ob eine Straße vom Typ 'service' ist oder nicht. Das Attribut 'service', welches bei PyTorch Tabular auf Platz zwei steht, ist wiederum ein eigenes Attribut für sich, welches angibt, ob und wenn ja welche Art von service Straße eine Straße ist. Hieraus lässt sich schließen, dass vor allem der Typ 'service' in beiden Modellen ein wichtiger Indikator den Privatbesitz von Straßen ist. Als service Straßen werden in OSM Zufahrtsstraßen verschiedenster Art bezeichnet. In unseren Daten kommen folgende Arten vor: 'parking_aisle', 'driveway', 'emergency_access' und 'alley'. Mit 'parking_aisle' werden Zufahrtsstraßen zu einem Parkplatz, mit 'driveway' Auffahrten zu Gebäuden und mit 'alley' schmale Gassen im Generellen bezeichnet. Diese Arten von Straßen sind also oftmals privat und können von unseren Modellen auch gut als solche erkannt werden. Ein weiteres wichtiges Attribut scheint 'access' bzw. 'access_unknown' zu sein. Dass bei XGBoost das Attribut 'access_unknown' und nicht 'access_private' oder 'access_no' so weit oben landet, wirkt auf den ersten Blick nicht sehr intuitiv. Eine mögliche Erklärung hierfür wäre jedoch, dass ein Wert von False für dieses Attribut eine erhöhte Aufmerksamkeit des Mappers auf den Zugang zur Straße bedeutet. Wie bei so vielen anderen Tags in OSM, wird der Tag 'access' oftmals nur dann aktiv gesetzt, wenn etwas vom Standard (also uneingeschränkt öffentlich zugänglich) abweicht. Es ist also plausibel, dass es eine Korrelation zwischen dem Attribut 'access_unknown' und dem Privatbesitz von Straßen gibt. Das Attribut 'maxspeed' ist für XGBoost ebenfalls noch wichtig, möglicherweise da private Straßen oftmals in der Nähe von Wohngebäuden vorkommen, wo die erlaubte Höchstgeschwindigkeit meist niedriger ist als im Rest der Stadt.

Alle weiteren Attribute weisen für beide Modelle eine Permutation Feature Importance von < 1 % auf. Dies bedeutet wie zuvor erwähnt jedoch nicht, dass sie für die Modelle nutzlos sind. Es bedeutet lediglich, dass sie nicht isoliert aus der Menge herausstechen. Tatsächlich führt auch das Trainieren von Modellen, welche die Attribute mit geringer Permutation Feature Importance nicht benutzen, zu deutlich schlechteren Ergebnissen. So lässt sich also festhalten, dass einzig der Straßentyp 'Service' ein klares Indiz für private Straßen bildet.

Um dennoch etwas über die Korrelationen in den Daten zu erfahren, können wir Gebrauch von der Pandas Funktion "corr()"machen. Angewendet auf unseren Datensatz gibt sie uns eine Korrelationsmatrix zurück, welche alle möglichen Paare von Attributen enthält. Die Werte sind sog. Korrelationskoeffizienten. Sie geben im Bereich von -1 bis +1 an, ob zwischen den Attributen ein negativer, keiner oder ein positiver linearer Zusammenhang besteht [21]. Tabelle 4 gibt einen Überblick der Werte für die Korrelationen von allen Attributen mit der Grundwahrheit.

Attribut	Korrelationskoeffizient
type_residential	-0.49
surface_asphalt	-0.38
service_no	-0.34
sidewalk_both	-0.29
access_unknown	-0.25
.	.
.	.
.	.
simple_deadend	0.22
access_private	0.23
service_driveway	0.24
surface_unknown	0.35
sidewalk_none	0.35
max_speed	0.38
type_service	0.59

Tabelle 4: Korrelationskoeffizienten der Grundwahrheit mit allen anderen Attributen

Tabelle 4 liefert uns keine großen Überraschungen, sondern bestätigt unsere Permutation Feature Importance Ergebnisse. Der Straßentyp 'residential' hat die größte negative Korrelation. Er bildet sozusagen das Gegenstück zu dem Straßentyp 'service', welcher die größte positive Korrelation besitzt. Des Weiteren sticht noch das Attribut 'max_speed' heraus. Warum ein Ansteigen der erlaubten Maximalgeschwindigkeit positiv mit privaten Straßen korreliert, scheint zunächst fragwürdig. Eine mögliche Erklärung hierfür wäre jedoch, dass die erlaubte Maximalgeschwindigkeit von privaten Straßen oftmals nicht explizit ausgeschildert bzw. für Mapper unbekannt ist. Lücken im 'max_speed' Attribut wurden von uns bei dem Aufbereiten der Daten auf den Standardwert von 50 km/h gesetzt. Dies ist auch die höchste in unseren Daten vorkommende Maximalgeschwindigkeit. Es kann also durchaus sein, dass viele private Straßen eine Maximalgeschwindigkeit von 50 km/h zugewiesen bekommen haben, wodurch dann auch 'max_speed' mit der Grundwahrheit korreliert.

6.3 Ausblick

Die vorangegangene Evaluation hat gezeigt, dass das Vorhersagen des Privatbesitzes von Straßen anhand von OSM Daten grundsätzlich möglich ist. Um die Genauigkeit des Modells und seine Robustheit zu erhöhen, könnten jedoch noch einige Verbesserungen vorgenommen werden: Beispielsweise eine deutlich größere Datenmenge für das Training wäre sicherlich von Vorteil, da so die Modelle mehr Muster erlernen und so auch ein besseres Verständnis für die Eigenschaften von privaten Straßen erlangen könnten. Ein Indiz dafür, dass wir mit der bisherigen Datenmenge noch kein Optimum erreicht haben, sind auch die Ergebnisse von Testtrainings mit weniger Daten. So führt das Training mit nur 90 oder 80% der Ursprungsdaten auch zu einer schlechteren Genauigkeit (80 bzw. 77 %).

Des Weiteren würden mehr Daten aus verschiedenen Städten auch zu einer besseren Generalisierbarkeit der Modelle führen. Die Modelle in unserer Arbeit haben nur gelernt, wie Privatbesitz von Straßen in Heidelberg aussieht. In einer anderen Stadt mit einer möglicherweise anderen Straßenbaupolitik bzw. Kultur könnten die gelernten Parameter also vielleicht nicht mehr passend sein. Ein Datensatz mit Straßen aus vielen verschiedenen Städten könnte diesem Problem entgegenwirken. Eine zusätzliche Verbesserungsmöglichkeit wäre das Berechnen von weiteren Geometrie Attributen. So gibt es auch viele private Straßen, welche keine Sackgasse, sondern in U-Form an eine nicht-private Straße angeschlossen sind. Man könnte also ein Attribut berechnen, welches angibt, ob eine Straße diese Eigenschaft hat, wodurch die Genauigkeit der Modelle potentiell gesteigert werden würde.

Um die hier getesteten Methoden tatsächlich im Planungsprozess von Glasfaserkabelverlegung einzusetzen, müsste noch eine weitere Anpassung vorgenommen werden: Da das Vermeiden von vielen Straßen unter Umständen teurer ausfällt als das gelegentliche Nutzen von privaten Straßen, sollte ein größerer Fokus auf das vermeiden von falsch-positiven Vorhersagen gemacht werden. Gerade das Gradient-Boosting Modell hat hier mit 15% (siehe Abbildung 11) eine ziemlich hohe Fehlerquote. Ein Fokus auf eine geringe falsch-positiv Rate könnte zwar die allgemeine Genauigkeit des Modells senken, sich jedoch im Kontext von einer Kostenoptimierung trotzdem lohnen.

7 Fazit

In dieser Arbeit haben wir gezeigt, dass das Vorhersagen des Privatbesitzes von Straßen mittels OpenStreetMap Daten und Machine Learning grundsätzlich möglich ist. Die Analyse unserer Datenquellen hat ergeben, dass vor allem die OSM Daten einige Lücken und Ungenauigkeiten aufweisen. Dennoch konnten wir diese, nicht zuletzt auch durch unsere aufwendige Aufbereitung, gut für das Trainieren von zwei Machine Learning Modellen verwenden. Im Zuge dieser Aufbereitung haben wir, neben dem Berechnen eines zusätzlichen Attributs, auch die Grundwahrheit über den Privatbesitz von Straßen herausgearbeitet. Hierfür kamen amtliche Daten der Stadt Heidelberg zum Einsatz, was die Glaubwürdigkeit unserer Modelle und Vorhersagen unterstreicht.

Beide trainierten Modelle haben eine zufriedenstellende Genauigkeit erzielt. Dabei schnitt das neuronale Netzwerk in seiner Gesamtheit etwas besser ab als das Gradient-Boosting Model. Mithilfe der Permutation Feature Importance haben wir die Aussagekraft und Wichtigkeit der verschiedenen verwendeten Attribute für beide Modelle analysiert. Dabei sind vor allem Straßen vom Typ 'service' herausgestochen, welche wohl ein sehr guter Indikator dafür sind, dass sich eine Straße im Privatbesitz befindet. Um die von uns vorgestellte Methode in der Praxis einzusetzen und zu optimieren, würde es sich anbieten, beide Modelle erneut und mit deutlich mehr Straßendaten aus verschiedenen Städten zu trainieren.

8 Danksagungen

Zuallererst möchte ich mich bei meinen Eltern bedanken, die mir all das hier ermöglicht haben.

Dann möchte ich noch Fabian Koch danken, der mich durch Mathe eins bis vier gebracht hat und meine Denkknoten in diesem Bereich zumindest etwas gelockert hat.

Und ich möchte Hannah Bast, Bruno Becker und Michael Schlenk danken, die meine Arbeit betreut und unterstützt haben.

Literaturverzeichnis

[1] "Oecd broadband statistics update." https://www.oecd.org/sti/broadband/1.10-PctFibreToTotalBroadband-2021-12.xls. Abgerufen: 20.02.2023.

[2] "Die privatkundennachfrage nach hochbitratigem breitbandinternet im jahr 2025." https://www.wik.org/fileadmin/Studien/2017/Die_Privatkundennachfrage_nach_hochbitratigem_Breitbandinternet_im_Jahr_2025_FINAL.pdf. Abgerufen: 20.02.2023.

[3] "Widmung (straßen- und wegerecht)." https://de.wikipedia.org/wiki/Widmung_(Stra%C3%9Fen-_und_Wegerecht). Abgerufen: 20.02.2023.

[4] S. H. Pazoky and P. Pahlavani, "Developing a multi-classifier system to classify osm tags based on centrality parameters," *International Journal of Applied Earth Observation and Geoinformation*, vol. 104, p. 102595, 2021.

[5] D. Feldmeyer, C. Meisch, H. Sauter, and J. Birkmann, "Using openstreetmap data and machine learning to generate socio-economic indicators," *ISPRS International Journal of Geo-Information*, vol. 9, no. 9, 2020.

[6] S. Storandt and S. Funke, "Automatic improvement of point-of-interest tags for openstreetmap data," *Free and Open Source Software for Geospatial (FOSS4G) Conference*, vol. 15, 2015.

[7] "Openstreetmap - deutschland." https://www.openstreetmap.de/. Abgerufen: 20.02.2023.

[8] "Downloading data." https://wiki.openstreetmap.org/wiki/Downloading_data. Abgerufen: 20.02.2023.

[9] "Overpass api." https://wiki.openstreetmap.org/wiki/Overpass_API. Abgerufen: 20.02.2023.

[10] "Qlever." https://qlever.cs.uni-freiburg.de/osm-germany. Abgerufen: 20.02.2023.

[11] "Sparql." https://de.wikipedia.org/wiki/SPARQL. Abgerufen: 20.02.2023.

[12] "Tiefbauamt heidelberg." https://www.heidelberg.de/hd/HD/Rathaus/Tiefbauamt.html. Abgerufen: 20.02.2023.

[13] "Qgis." https://www.qgis.org/de/site/. Abgerufen: 20.02.2023.

[14] "Wgs84." http://giswiki.org/wiki/WGS84. Abgerufen: 20.02.2023.

[15] T. pandas development team, "pandas-dev/pandas: Pandas."

[16] "Fortschritte bei künstlichen neuronalen netzwerken." https://www.ip-insider.de/die-treiber-des-technologie-fortschritts-entwickeln-sich-a-890522/. Abgerufen: 20.02.2023.

[17] manujosephv, "manujosephv/pytorch_tabular: v0.5.0-alpha," May 2021.

[18] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, (New York, NY, USA), pp. 785–794, ACM, 2016.

[19] "Machine learning challenge winning solutions." https://github.com/dmlc/xgboost/blob/master/demo/README.md#machine-learning-challenge-winning-solutions. Abgerufen: 20.02.2023.

[20] C. Molnar, *Interpretable Machine Learning*. 2 ed., 2022.

[21] "Korrelationskoeffizient / pearson-korrelation." https://de.wikipedia.org/wiki/Korrelationskoeffizient. Abgerufen: 20.02.2023.