Fachbereich Informatik und Medien

BACHELORARBEIT

Matching und Visualisierung von Points of Interest aus sozialen Daten

Vorgelegt von: Paul Lehmann

am: 31.08.2011

zum

Erlangen des akademischen Grades

BACHELOR OF SCIENCE

(B.Sc.)

Erstbetreuer: Dipl.-Inf. Ingo Boersch

Zweitbetreuer: Dr. Tatjana Scheffler

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit zum Thema

Matching und Visualisierung von Points of Interest aus sozialen Daten

vollkommen selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt, sowie Zitate kenntlich gemacht habe. Die Arbeit wurde in dieser oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

Brandenburg/Havel, den 31.08.2011

Unterschrift

Danksagung

Hiermir möchte ich mich bei allen herzlich bedanken, die mich bei der Anfertigung meiner Arbeit unterstützt haben.

Mein Dank geht an meine beiden Betreuer Frau Tatjana Scheffler am DFKI und Herrn Ingo Boersch seitens der FH Brandenburg. Ferner möchte ich Rafael Schirru, ebenfalls vom DFKI, danken, der stets für Korrekturen bereit stand und meine Fragen immer bestmöglich beantwortet hat.

Meinen Freunden möchte ich für die Unterstützung während der Dauer der Arbeit danken.

Außerdem danke ich meiner Familie, die mich wie immer tatkräftig unterstützt hat.

Abkürzungsverzeichnis

App ..... Application
DFKI ..... Deutsches Forschungszentrum für künstliche Intelligenz
GPS ..... Global Positioning System
GSB ..... Geofilter, Stringpreprocessing, Bigram-Distance
GSE ..... Geofilter, Stringpreprocessing, Edit-Distance
GSL ..... Geofilter, Stringpreprocessing, longest common substring
GST ..... Geofilter, Stringpreprocessing, TF-IDF
HTTP ..... Hypertext Transfer Protocol
ID ..... Identifikator
IDF ..... inverse document frequency
JSON ..... JavaScript Object Notation
LCS ..... longest common substring
NP ..... nähester Point of Interest
OSM ..... OpenStreetMap
PoI ..... Point of Interest
REST ..... representational state transfer
TF ..... term frequency
URL ..... Uniform Resource Locator

Inhaltsverzeichnis

1 Einleitung

Weltwissen liegt immer häufiger in (sozialen) Netzwerken im *World-Wide-Web* vor. Es wird von vielen Personen genutzt und bearbeitet. Die Repräsentation jedes einzelnen Objekts kann sich je nach Netzwerk unterscheiden. Dadurch muss bei der Entwicklung von Anwendungen, die mit den Daten unterschiedlicher Netzwerke arbeiten, darauf geachtet werden, dass diese Objekte auf die jeweilige Repräsentation des selben Objekts in einem anderen Netzwerk abgebildet werden und so verknüpft werden können.

1.1 Thema und Motivation

Diese Thematik kam das erste Mal zum Ausdruck während meines Praktikums beim Deutschen Forschungszentrum für Künstliche Intelligenz im Projektbüro Berlin. Aufgabe des Praktikums war es, Points of Interest (im Folgenden PoIs) von verschiedenen sozialen Diensten zu beziehen und anschließend auf einer Karte darzustellen. Bei der Anzeige fiel dann auf, dass die PoIs der verschiedenen Netzwerke sich überlagerten, da viele der PoIs in jedem der Dienste vorhanden waren. Ferner muss man bei den zugrundeliegenden Daten auch immer noch den Faktor des *user generated content* mit in Betracht ziehen, da solche sozialen Metadaten potenziell fehlerbehaftet, unvollständig und widersprüchlich sind. Da die Lösung dieses Problems keinesfalls trivial ist, sollte es im Rahmen dieser Arbeit untersucht werden.

1.2 Zielsetzung

Die Daten, die für diese Arbeit benutzt wurden, stammen aus den Netzwerken OpenStreetMap[1], Qype[2], Facebook Places[3] und Panoramio[4]. Aus Gründen der Handhabbarkeit beschränkt sich die Arbeit auf Daten aus dem Raum Berlin. Das Primärziel der Arbeit ist,

[1] http://www.openstreetmap.org/
[2] http://www.qype.com/
[3] http://www.facebook.com/places
[4] http://www.panoramio.com/

die jeweiligen Entsprechungen eines PoIs aus jedem der vorhandenen Netzwerke zu einem zusammenzulegen, also das Matching. Dabei muss man vermutlich hier und da Kompromisse eingehen, was dazu führt, dass Fehler entstehen. Ein Ziel muss es dabei sein, die Fehler minimal zu halten. In dieser Arbeit werden verschiedene Algorithmen oder Kombinationen jener Algorithmen gegenüber gestellt und deren Vor- und Nachteile erörtert. Ferner wird auch darauf eingegangen, welche Fehler bei bestimmten Abläufen auftreten und durch was sie provoziert werden. Nach dem Matching muss bestimmt werden, wie der resultierende PoI am Ende repräsentiert werden soll, seine kanonische Repräsentation. Die Paare von PoIs sollen dann über einen Webservice verfügbar sein. Zur Veranschaulichung der Ergebnisse werden die PoIs in einer Android-App auf einer Google-Maps-Karte visualisiert, die die Daten von dem Webservice bekommt.

1.3 Aufbau der Arbeit

Zuerst werden in Kapitel 2 die genutzten Netzwerke vorgestellt, anschließend wird auf die grundlegende Problematik der unterschiedlichen Repräsentation des selben Objekts in verschiedenen Datenbanken eingegangen. Außerdem werden Maße vorgestellt, die dazu dienen herauszubekommen, wie ähnlich sich Einträge sind, um so gleiche Objekte zu finden und deren Daten zu aggregieren. Im dritten Kapitel wird auf die Anwendung dieser Algorithmen in der vorliegenden Arbeit eingegangen, was hier anders ist, was aufgrund der bestehenden Daten angepasst werden musste usw. Ferner wird hier die schrittweise Erweiterung des verwendeten Algorithmus erläutert. Das vierte Kapitel befasst sich mit der Implementierung, zunächst der einzelnen Module, und dann der jeweiligen Matching-Algorithmen, die auf die einzelnen sozialen Netzwerke abgestimmt sind. Außerdem wird noch auf den Webservice, der die Daten bereit stellt und die App eingegangen, in der dann die Visualisierung stattfinden soll. Im fünften Kapitel, der Evaluierung, werden die Ergebnisse der verschiedenen Erweiterungen des Matching-Algorithmus ausgewertet. Eine Zusammenfassung und ein kurzer Ausblick folgen in Kapitel 6.

2 Allgemeine Grundlagen

2.1 Datenquellen

Die Daten, die für diese Arbeit benutzt wurden, stammen aus den Netzwerken OpenStreetMap, Qype, Facebook Places und Panoramio. Die PoIs aller vier Netzwerke haben Länge und Breite im Gradnetz als Koordinaten und einen Titel. Jedes dieser Netzwerke stellt andere zusätzliche Metadaten zur Repräsentation von Objekten aus der realen Welt zur Verfügung.

2.1.1 OpenStreetMap

OpenStreetMap oder auch kurz OSM ist ein britisches Projekt, dass im Jahre 2004 durch Steve Coast ins Leben gerufen wurde. Es hat das Ziel, eine Weltkarte zu erschaffen, die für jedermann frei zugänglich ist. Die Daten für diese Weltkarte werden von den Nutzern eingetragen und sind für jeden verfügbar. Es sind mittlerweile eine Vielzahl an verschiedenen Karten entstanden, seien es normale Straßenkarten, Karten im Stil von *Google-Maps*, Fahrradkarten, Wanderkarten usw., die den kommerziellen Produkten ebenbürtig sind. Die verfügbaren Daten für PoIs aus OSM sind Koordinaten, Kategorien und Titel. Die Kategorien und Titel sind dabei allerdings nicht immer gut gewählt, da sie von den Nutzern stammen und potenziell falsch sind. Um das zu erläutern, ein kleines Beispiel wie bei OSM Kategorien vergeben werden. In OSM werden geographische Punkte mit Tags versehen. Jedes Tag besteht aus einem Schlüssel und einem dazugehörigen Wert. Die meisten PoIs werden unter dem Schlüssel „amenity“ verwaltet. Ab hier bestimmt allerdings der Nutzer, was der zugehörige Wert ist. Dabei kann es passieren, dass Leute „yes“, „no“ oder auch den Titel eintragen. Demzufolge kann die Richtigkeit dieser Daten nicht immer garantiert werden. Bei den Koordinaten ist es ähnlich. Sie bestehen aus mindestens sechs Stellen nach dem Komma, was sie genauer macht als die Daten der anderen Netzwerke (siehe Kapitel 3.1), allerdings können diese auch falsch eingetragen sein.

Abbildung 2.1: Visualisierung eines PoI und seiner Metadaten in der Qype-Plattform

2.1.2 Qype

Qype ist ein Dienst, in dem man seine Meinung über eine Lokalität[1] ausdrücken kann. Das Portal bietet die Möglichkeit Wertungen für jede eingetragene Lokalität zwischen eins und fünf abzugeben. Ferner kann jeder Nutzer Kommentare schreiben, die dann für andere lesbar sind. Qype wurde im November 2005 durch Stefan Uhrenbacher gegründet und ging im Jahr 2006 erstmals online. Anfangs nur für deutsche Städte vorhanden, wurde Qype mittlerweile in vielen Ländern eingeführt. Man erhält unter Anderem folgende Informationen zu PoIs: den Titel des Lokals, das durchschnittliche Rating (zwischen 0 -nicht bewertet- und 5 -sehr gut-), die Kommentare der Nutzer (siehe Abb. 2.2), ein Profilbild, die Koordinaten (nur vier Stellen nach dem Komma), die Kategorien, weitere Bilder, eine Beschreibung durch die Besitzer etc. In Abbildung 2.1 ist ein Beispiel zu sehen, wie eine Seite bei Qype aussieht.

Abbildung 2.2: Beispiel eines durchschnittlichen Ratings und Kommentars in Qype

[1] In dieser Arbeit werden nur Restaurants, Cafes, Bars, Läden, Sehenswürdigkeiten und Ähnliches verwendet, keine Ärzte, Anwälte, Schulen usw.

2.1.3 Facebook Places

Facebook Places ist ein Dienst, mithilfe dessen man seine Statusnachrichten mit georeferenzierten Informationen versehen kann (Check-In). Beispielsweise kann man mit seinen Freunden teilen, wo man sich gerade befindet, in welchem Lokal man gerade etwas trinkt oder isst. Die verfügbaren Daten sind die Koordinaten, die Titel der PoIs, die Anzahl der Check-Ins und eine Beschreibung der Lokalität.

2.1.4 Panoramio

Der Dienst Panoramio wurde im Jahr 2005 von den beiden Spaniern Joaquín Cuenca Abela und Eduardo Manchón Aguilar gegründet. Auf Panoramio kann man georeferenzierte Fotografien hochladen, das sind Fotos, zu denen man die geografische Lage abspeichert, von wo aus sie gemacht wurden. Seit der Übernahme durch Google im Jahr 2007 hat sich die Seite stark weiterentwickelt und vergrößert. Die Daten, die Panoramio zu einem PoI bereitstellt, sind das Bild, die Koordinaten, die Beschreibung des Bildes (Titel) und noch einige anderen Daten, die für diese Arbeit nicht berücksichtigt wurden.

2.2 Datenheterogenität

Bei dem Versuch Daten von verschiedenen Quellen zu integrieren, stellt man fest, dass die Daten je nach Dienst unterschiedlich repräsentiert werden. Desweiteren gelten andere Standards für die Nutzung der Daten. Solche Probleme fallen unter den Begriff der Datenheterogenität. Unterschieden werden zwei Typen: strukturelle und lexikalische Heterogenität (siehe auch [EIV07]).

2.2.1 Strukturell

Strukturelle Heterogenität tritt auf, wenn die Attribute, die sich in den verschiedenen Datenbanken eigentlich entsprechen, je nach Datenbank anders repräsentiert werden. So werden manche Attribute in der einen Datenbank vielleicht auf mehrere Felder aufgeteilt, und in einer anderen Datenbank werden sie nur in einem einzigen Feld gespeichert. Ein Beispiel wäre hierbei das Attribut Adresse. In einer Datenbank wurde es in seine verschiedenen Bestandteile, wie Straße, Hausnummer, Postleitzahl, Stadt, aufgeteilt. In einer anderen Datenbank könnte es aber auch durch zwei Felder repräsentiert werden, in das eine Feld speichert man Straße und Hausnummer und in das andere die Daten PLZ und Ort. In einer

dritten Datenbank wiederum wird die Adresse nur in ein Feld geschrieben. Als Lösung für das Problem müsste man die Struktur der Daten jeder Datenbank so anpassen, damit sie in eine gemeinsame Tabelle geschrieben werden können.

2.2.2 Lexikalisch

Da bei den hier verwendeten Daten, bzw. deren Quellen, das Problem der strukturellen Heterogenität keine Rolle spielte, wird auf diese im Weiteren nicht mehr eingegangen. Im Gegenteil dazu tritt die lexikalische Heterogenität, auch semantische Heterogenität genannt, auf, wenn die Tupel der Datenbanken identisch strukturiert sind, aber unterschiedliche Repräsentationen genutzt werden, um die gleichen Objekte zu referenzieren. Als Beispiel kommt die Zeichenkette „Café“ infrage. Bei der einen Quelle wird es mit dem Akzent abgebildet und bei einer anderen Quelle ohne. Dieses Problem besteht generell bei allen Sonderzeichen, weil man nie sicher sein kann, ob alle dieselbe Repräsentation nutzen. Wie an dieses Problem herangegangen wurde, dazu im Verlauf der Arbeit mehr. Ein weiteres Problem hätte sein können, dass die Geokoordinaten der verschiedenen Dienste unterschiedlich zur Verfügung gestellt werden, z.B. bei dem einen Dienst werden die Daten als Dezimalgrad angegeben und bei einem anderen Dienst als Grad mit Dezimalminuten. Dann müsste man die Daten erst in eine einheitliche Form bringen. Da in den gewählten sozialen Netzwerken dieses Problem nicht auftrat, war keine Transformation der Geokoordinaten notwendig.

2.3 Deduplizierung der Einträge

Im Gegenteil zu dem in [EIV07] vorgestellten Finden von doppelten Einträgen und dem Deduplizieren jeder dieser Dopplungen, ist es bei dieser Arbeit das Ziel, PoIs der einen Quelle auf PoIs einer anderen Quelle abzubilden und ihre Metadaten zu aggregieren. Sobald der passende Partner gefunden wurde, sollen die Daten der beiden PoIs vereinigt werden. Alles, was spezifisch für die jeweilige Quelle ist, soll so auch vorhanden bleiben, da es den PoI weiter charakterisiert. Ferner muss abgewogen werden, welche Repräsentation von Titel und Koordinaten benutzt wird, um den Punkt später zu beschreiben. Für die Koordinaten würde stets der OSM-Datensatz in Betracht kommen, da die Daten dort sechs Stellen nach dem Komma haben und somit vermutlich genauer sind, als die Daten von Qype, die maximal vier Stellen besitzen. Der Titel hingegen, wie in Kapitel 3.1 zu sehen, ist in Qype bzw. Facebook Places konkreter hinterlegt, in OSM ist er teilweise unvollständig und fehlerhaft.

2.4 Ähnlichkeitsmaße

Um zu bestimmen, welche Points of Interest sich ähnlicher sind als andere, muss man Maße finden, die PoIs sinnvoll aufeinander abbilden. Im Folgenden werden einige Verfahren vorgestellt, die man nutzen kann, um die Ähnlichkeit der Attribute von PoIs zu bestimmen.

2.4.1 Nähester Point of Interest

Es wird der PoI gesucht, der den geringsten Abstand zu dem Ausgangspunkt hat. Normalerweise würde in die Berechnung noch die Erdkrümmung mit einfließen. Da sich die Daten, die dieser Arbeit zugrundeliegen, nur auf den Bereich Berlin beschränken, wurde dieser Fakt aus der Berechnung heraus gelassen. Die Distanz (*Euklidischer Abstand*) zwischen zwei Punkten berechnet man, indem zuerst die Differenzen in Länge und Breite errechnet werden. Diese beiden Werte werden anschließend quadriert und addiert. Zum Schluss wird die Quadratwurzel gezogen und das Ergebnis ist die Distanz zwischen diesen beiden Punkten. Folgendes ist die Gleichung für die Distanz zweier Punkte p_1 und p_2 in der Ebene:

$$d(p_1, p_2) = \sqrt{(lat_{p_1} - lat_{p_2})^2 + (lon_{p_1} - lon_{p_2})^2} \tag{2.1}$$

Eine Fehlerquelle dieses Verfahrens ist die fehlerhafte Angabe der Koordinaten in manchen Netzwerken, die von den Nutzern eingetragen werden. Ein weitere potenzielle Fehlerquelle wäre die Ungenauigkeit von GPS-Geräten, deren Schwankung bei rund 20 Metern liegt ([kow07]). Dieses Verfahren ist nur dafür geeignet, die Koordinaten der beiden PoIs miteinander zu vergleichen, und lässt die anderen vorhandenen Daten unberücksichtigt.

2.4.2 Longest common substring

Bei diesem Maß handelt sich um einen reinen Zeichenkettenvergleich, in dem man herausbekommt, welche die längste Zeichenkette ist, die zwei oder mehr Zeichenketten gemeinsam haben. Die Kosten dieses Verfahrens liegen bei $O(nm)$, wobei n die Anzahl der Zeichen in der ersten Zeichenkette ist und m die Anzahl der Zeichen in der zweiten. Vorteile dieses Verfahrens sind die schnelle und einfache Implementierung und ein gutes Ergebnis, wenn die Quelldaten sehr ähnlich zueinander sind. Probleme treten sehr schnell auf, wenn die Quelldaten beispielsweise kurze Zeichenketten sind, bei denen die längste mögliche Zeichenkette nicht wirklich ausschlaggebend ist, oder Tippfehler vorhanden sind. Beispielsweise

wäre der *longest common substring* der beiden Zeichenketten „abcdefgh“ und „bacdef“ die Zeichenkette „cdef“.

2.4.3 Edit-Distance

Die Edit-Distance (auch Levenshtein-Distance) beschreibt erneut ein Verfahren zum Zeichenkettenvergleich. Sie bestimmt die minimale Anzahl an Operationen, die nötig sind, um eine Zeichenkette s_1 in eine andere Zeichenkette s_2 umzuwandeln (siehe dazu [MRS08] Seite 58 f.). Die verfügbaren Operationen sind dabei Einfügen, Löschen oder Substituieren eines Zeichens. Da die Wichtung jeder dieser Operationen standardmäßig bei 1 liegt, bekommt man als Ergebnis eine ganze Zahl. Die Zeichenkette s_2 mit dem niedrigsten Wert ist dann der Zeichenkette s_1 am ähnlichsten. Bei kurzen Zeichenketten treten hierbei im Vergleich zu längeren schnell Probleme auf. Gegeben seien die Zeichenketten s_1 mit „cat“ und s_2 mit „dog“. Die Edit-Distance wäre hierbei 3, weil jeder einzelne Buchstabe ausgetauscht werden muss. Wenn man längere Zeichenketten hat, wie zum Beispiel den Titel eines Restaurants, könnte es passieren, dass der Titel in dem einen Netzwerk den Zusatz „Restaurant“ mit im Namen trägt und in dem anderen Netzwerk nicht. Abgesehen davon, wie lang der eigentliche Titel des Restaurants ist, wäre das eine Distanz zwischen den beiden Zeichenketten von 10. Wenn man sich die beiden Distanzen anguckt, könnte man meinen, dass sich die beiden Wörter des ersten Beispiels ähnlicher sind, als die des zweiten. Die Schwelle für eine mögliche Ähnlichkeit darf hierbei demzufolge kein fester Wert sein, denn je nach Länge der Zeichenketten sind andere Herausforderungen gegeben. Demzufolge muss die maximale Distanz abhängig von der kürzeren der beiden Zeichenketten sein.

Im Gegensatz zu anderen Verfahren, wie zum Beispiel TF-IDF, gibt es bei der Edit-Distance Möglichkeiten mit Tippfehlern umzugehen. Eine Verbesserungsmöglichkeit des Verfahrens wäre, die Wichtungen der Operationen anzupassen. Man könnte allen Ersetzungen durch Buchstaben, die auf der Tastatur nebeneinander liegen, eine geringere Wichtung zuweisen. Je nach Zweck und Anwendungsfeld muss geguckt werden, auf welche Art und Weise dieses Verfahren sinnvoll ist oder nicht.

2.4.4 TF-IDF-Wichtung

TF-IDF (siehe [SJ72]) dient im *Information Retrieval* der Wichtung bestimmter Terme in den Dokumenten, in denen sie auftreten im Verhältnis zum Dokument-Korpus. Der Begriff bildet sich aus der *term frequency* tf, die Vorkommenshäufigkeit eines Terms in einem Dokument, und der *inverse document frequency* idf, der inversen Dokumenthäufigkeit. Mithilfe dieser

beiden Werte wird auch die Wichtung des Terms in seinem jeweiligen Dokument bestimmt. Es besteht folgender Ablauf:

1. Termliste aufstellen
2. Häufigkeitsmatrix bestimmen
3. Wichtungen berechnen

Um die Termliste zu bestimmen, muss man zuerst alle Zeichenketten in ihre Wörter aufteilen und diese Wörter anschließend, falls noch nicht vorhanden, dieser Liste hinzufügen. Für den zweiten Schritt müssen mehrere Werte bestimmt werden. Zuerst einmal die Anzahl an Vorkommen eines Terms t in einem Dokument d, die Vorkommenshäufigkeit $tf_{t,d}$. Die inverse Dokumenthäufigkeit idf_t berechnet sich aus dem Logarithmus des Quotienten aus der Anzahl der Dokumente N in der gesamten Datenbank und der Anzahl der Dokumente n_t, die diesen Term beinhalten. Mithilfe dieser Werte kann man im letzten Schritt die Wichtungen bestimmen. Die Wichtung $w_{t,d}$ eines Terms in einem Dokument ist das Produkt der Vorkommenshäufigkeit $tf_{t,d}$ und der inversen Dokumenthäufigkeit idf_t:

$$w_{t,d} = tf_{t,d} \cdot idf_t = tf_{t,d} \cdot \log \frac{N}{n_t} \tag{2.2}$$

Daraus ergibt sich, dass je häufiger ein Term in einem Dokument vorkommt, desto stärker wird er in diesem Dokument gewichtet. Aber er wird schwächer gewichtet, wenn er in vielen verschiedenen Dokumenten vorkommt. Also werden die Dokumente vor allem durch die Terme charakterisiert, die entweder oft in diesem Dokument auftreten oder durch die, die im Korpus selten vertreten sind.

Dadurch, dass dieses Verfahren wortbasiert funktioniert, sind Tippfehler ein großes Problem. Es wird immer nur mit den kompletten Wörtern gearbeitet. Das führt dazu, dass Terme, wie „resaurant“ mit in die Termliste hineingenommen werden, obwohl es sich eigentlich um den Term „restaurant“ handelt, bei dem ein Buchstabe fehlt. Dieser Term erhält bei der Wichtung ein sehr hohes Gewicht, da er aufgrund des Tippfehlers einzigartig ist.

2.4.5 NGram-Ähnlichkeit

Bei der NGram-Ähnlichkeit ([Kon05]) handelt es sich um ein weiteres Verfahren, um herauszufinden, wie ähnlich sich zwei Zeichenketten sind. Dabei bestimmt das „n“ im Titel, wie lang die Zeichenketten sind, in die die Ausgangszeichenkette geteilt wird. Wenn $n = 1$, dann bezeichnet man das als Unigram. Das bedeutet, dass dabei jeder Buchstabe einzeln

betrachtet wird. Die Unigram-Ähnlichkeit ist exakt das selbe wie das Verfahren der *longest common subsequence*. Im Gegensatz zum *longest common substring* ist hierbei das Ergebnis keine Zeichenkette, die ununterbrochen auch Teil der beiden ursprünglichen Zeichenketten ist, sondern es werden alle Buchstaben verwendet, die in gleicher Reihenfolge in beiden Zeichenketten vorkommen, auch mit Unterbrechungen. Ein Beispiel: Die längste Subsequenz der beiden Zeichenketten „abcxdtefmgs“ und „apbociduezfg“ wäre „abcdefg“. Um nun zu bestimmen, wie ähnlich sich diese beiden Zeichenketten sind, könnte man die Längen der entstandenen Sequenzen vergleichen. Die Zeichenkette mit der längsten gemeinsamen Sequenz ist dann vermutlich die Zeichenkette, die der anderen am ähnlichsten ist.

In dieser Arbeit wurden Bigramme benutzt. Dabei wird die Zeichenkette so in kleine N-Gramme der Länge zwei aufgeteilt, dass der letzte Buchstabe eines N-Grams stets auch der erste des nächsten N-Grams ist. Bei der Zeichenkette „abcdefgh“ würde das wie folgt aussehen: „ab bc cd de ef fg gh“. Im Gegensatz zum Unigram, bei dem die einzelnen Buchstaben verglichen werden, vergleicht man hierbei die N-Gramme. Da die Reihenfolge der N-Gramme dabei aber eingehalten wird, führen Fälle, wie zum Beispiel die beiden Zeichenketten „Starbucks Cafe“ und „Cafe Starbucks“ zu Problemen und werden als nicht ähnlich genug gedeutet. In diesem Punkt hat das TF-IDF-Verfahren seine Vorzüge, da dort die Reihenfolge der Wörter irrelevant ist. Der Vorteil der Bigramme ist allerdings, dass Tippfehler nicht so schlimm wie bei TF-IDF ins Gewicht fallen. Sollte an einer Stelle ein Buchstabe zu viel oder zu wenig sein, dann wird dieser N-Gram übersprungen bzw. diesen N-Gram gibt es nicht, aber das Ergebnis ist dann immernoch aussagekräftig, denn die anderen N-Gramme sind noch gleich.

3 Konzept

In diesem Kapitel geht es um die Darstellung der Konzeption zur Lösung des Matching-Problems. Im letzten Kapitel wurden bereits Maße vorgestellt mit deren Hilfe man bestimmen kann, wie nah geografische Punkte beieinander liegen und wie ähnlich sich Zeichenketten sind. In diesem Kapitel wird vorgestellt, inwiefern die benutzten Verfahren angepasst werden mussten, um die Merkmale der PoIs, hier nur Koordinaten und Namen, miteinander zu vergleichen.

3.1 Welche Daten sind vorhanden?

Tabelle 3.1: Verfügbare Daten

	Facebook Places	Qype	Panoramio	OpenStreetMap
Geokoordinaten	ja	ungenau	ja	ja
Titel	ja	ja	eingeschränkt	eingeschränkt
Kategorien	nein	ja	nein	eingeschränkt

Koordinaten

Bei jedem der Dienste (siehe Tabelle 3.1), bis auf Qype, liegt die Genauigkeit der beiden Geokoordinaten Länge und Breite bei sechs Stellen nach dem Komma. Bei der Breite entspricht das etwa einer Schwankung von rund 0,111 Metern. Die Genauigkeit des Längengrades hängt allerdings unmittelbar von der Breite ab, denn je höher die Breite, desto dichter liegen die Längengrade beieinander. Bei einer Breite von etwa 52° (Berlin) entspricht das also einer Genauigkeit von rund 0,08 m (siehe [OW11]). Da Qype nur vier Stellen nach dem Komma unterstützt, ist die Genauigkeit also um ein 100-faches kleiner, also etwa 11,1 m in Breite und 8 m in Länge.

Titel

Bei den Titeln der Points of Interest sind die Entsprechungen in den beiden Diensten Facebook Places und Qype meist komplett und fehlerfrei. Die Ursache dafür ist vermutlich, dass die Titel der PoIs nach dem Eintragen nochmal auf Tippfehler überprüft werden. Bei OSM bzw. Panoramio sieht das allerdings anders aus. Bei Panoramio besteht das Problem, dass die PoIs hier nur durch Bilder repräsentiert und die restlichen Metadaten nicht zuverlässig sind. Dabei sind vor allem die Titel zu nennen, die meist unbrauchbar sind, da sie oft durch Stadt oder Land in dem sie sich befinden, erweitert werden (Beispiel: „Hightex- Waldbühne Open Air Stage, Berlin, Germany“, zu viele Zusätze, die nicht den eigentlichen PoI charakterisieren). Zum Teil sind die Titel auch in anderen Sprachen hinterlegt, wie zum Beispiel Russisch[1], Englisch[2] usw. Bei OSM dagegen ist der Titel meist unvollständig. Als Beispiel ist hier das „Indische Restaurant Amar“ zu nennen, das mit seinem vollen Namen in Qype vorhanden ist, allerdings in OSM nur unter „amar“ zu finden ist. Aus diesem Grund wurde auch für die Visualisierung stets der Name von Qype bzw. Facebook Places benutzt.

Kategorien

Die Kategorien der PoIs wären ein hervorragendes Vergleichsmittel, um die Lokalitäten mit denen bei OSM zu vergleichen, das Problem dabei ist, dass die vorhandenen Daten der Netzwerke Facebook Places und Panoramio keine Kategorien enthalten. Desweiteren werden die Kategorien bei OSM durch die Nutzer eingetragen, was dazu führt, dass sie unter Umständen nicht genau genug sind, schlichtweg falsch sind oder keinen Sinn ergeben. Die einzig verlässliche Quelle wäre hierbei Qype, weshalb das Matching über dieses Attribut für die betrachteten Datensätze nicht durchgeführt werden konnte.

3.2 Erste Baseline - Longest common substring

Als Basis wurde das Verfahren des *longest common substring* benutzt. Es ist schnell und einfach implementiert und bildet eine solide Grundlage, um die anderen Verfahren damit zu vergleichen. Verwendet wird hierbei nur der Titel/Name der Lokalität. Es wird die längste Zeichenkette gesucht, die die Titel von zwei PoIs gemeinsam haben. Der PoI, dessen Titel die längste Teilzeichenkette hat, wird als Match gewertet. Allerdings muss die Länge der Teilzeichenkette eine gewisse Mindestlänge überschreiten. Sollte kein Kandidat eine

[1] http://www.panoramio.com/photo/17354384
[2] http://www.panoramio.com/photo/9444619

gemeinsame Zeichenkette mit dieser Länge haben, so wird für diesen PoI kein Match zugewiesen. Sollte der Fall eintreten, dass es zwei mögliche Matches gibt, die beide die selbe Länge der Teilzeichenkette haben, so wird der PoI ausgewählt, bei dem das Verhältnis von Länge der Teilzeichenkette zur Länge des Titels des PoIs, zu dem der Match gesucht wird, größer ist.

Probleme treten hierbei schnell auf, wenn die Zeichenketten nicht komplett korrekt sind oder sich Sonderzeichen darin befinden. Zum Beispiel ist in Qype ein PoI mit dem Namen „Mendoza Argentinisches Steakhaus Berlin“ hinterlegt (siehe Abb. 2.1). Wenn man die Zeichenketten in keiner Weise bearbeitet, wird es Probleme geben, die Entsprechung dieses PoIs in OSM zu finden, da dieser dort mit dem Namen „MENDOZA - Argentinisches Steakhaus“ eingetragen ist. Da sich die beiden Zeichenketten eigentlich nur minimal unterscheiden, könnte man annehmen, dass das kein Problem darstellen sollte. Die längste Zeichenkette, die beide gemeinsam haben, wäre jedoch „Argentinisches Steakhaus“. Da bei OSM der eigentliche Name „Mendoza“ in Großbuchstaben geschrieben ist, fällt er bei der Suche nach der Teilzeichenkette heraus. Da es aber vermutlich nicht das einzige argentinische Steakhaus in Berlin ist, wird hier eventuell ein falscher Match zugewiesen. Vielleicht schon deshalb, weil der Name eines anderen Steakhauses auf ein „a“ endet und die Teilzeichenkette damit länger wäre. Selbst wenn der Name in beiden gleich geschrieben worden wäre, bestünde immer noch das Problem mit dem Bindestrich zwischen dem Namen und der Art des Restaurants im OSM-Namen. Bei den Erweiterungen muss eine Normierung der Zeichenketten erfolgen, um solche Fehler zu vermeiden.

3.3 Alternative Baseline - Nähester Point of Interest

Alternativ zum *longest common substring* als Basis wurden die Punkte als Matches gewählt, die die geringste Entfernung voneinander haben. Im Gegenteil zur ersten Baseline hat der Titel des PoIs hierbei keinen Einfluss, es geht nur um die Koordinaten. Um die Distanz zu berechnen, wird der *Euklidische Abstand* von zwei Punkten in der Ebene bestimmt (siehe Kapitel 2.4.1). Durch ungenaue, wie bei Qype (nur vier Stellen nach dem Komma), und falsche Koordinaten werden unter Umständen andere PoIs als Kandidaten gefunden, die eigentlich nicht infrage kommen würden. Außerdem treten Probleme auf, wenn mehrere PoIs den gleichen Abstand haben, also mehrere für den Match infrage kommen würden.

3.4 Erweiterungen

In diesem Abschnitt geht es darum, mit welchen Mitteln man die Baselines erweitern könnte, um die Ergebnisse zu verbessern. Eine erhebliche Steigerung ist schon dadurch möglich, dass man beide Attribute miteinbezieht und Wege findet, wie beide Einfluss auf das Ergebnis nehmen.

3.4.1 Stringpreprocessing

In Kapitel 3.2 wurden die möglichen Probleme aufgezeigt, die auftreten, wenn man Zeichenketten aus verschiedenen Quellen miteinander vergleichen will. Es muss also ein Weg gefunden werden, diese Zeichenketten zu normieren. Dabei muss insbesondere an zwei Dinge gedacht werden. Zum einen, was macht man mit den Umlauten in der deutschen Sprache und zum anderen, was passiert mit den anderen Sonderzeichen. Da man nicht sicher sein kann, wie mit den Umlauten, also „ä“, „ö“, „ü“, in jedem einzelnen Netzwerk umgegangen wird, werden alle Vorkommen der Umlaute auf „ae“, „oe“ und „ue“ abgebildet. Im nächsten Schritt sollen alle nicht-alphanumerischen Zeichen durch ein Leerzeichen ersetzt werden. Die alphanumerischen Zeichen umfassen das lateinische Alphabet von a-z in Groß- und Kleinbuchstaben. Außerdem gehören die Ziffern von 0-9 und das Leerzeichen dazu. Dann werden Leerzeichen, die in der Zeichenkette hintereinander liegen zu einem reduziert. Und zum Schluss wird alles in Kleinbuchstaben umgewandelt. Die Zeichenkette „Ab ächd? efö !";.g)h“ würde durch diesen Ablauf zu „ab aecd efoe g h“ werden. Dies soll verwendet werden, bevor man die beiden Zeichenketten der PoIs miteinander vergleicht.

3.4.2 Geofilter

Um die Koordinaten in das Ergebnis einfließen zu lassen, soll ein Filter entworfen werden, der aus der Liste der PoIs Berlins Kandidaten auswählt, die dann durch andere Verfahren weiterverarbeitet werden. Die Auswahl erfolgt mithilfe eines Rechtecks, auch *Bounding Box* genannt, das die Grenzen festlegt (siehe Abb. 3.1). Der schwarze Punkt in der Mitte ist der PoI (von Qype, Facebook Places oder Panoramio), für den der passende Match gefunden werden soll. Die grauen Punkte sind PoIs aus OSM, die in der *Bounding Box* liegen und damit zur Liste der Kandidaten gehören. Alles, was außerhalb der Box liegt, wird nicht in Betracht gezogen. Die Größe des Rechtecks soll parametrisierbar sein. Wie in der Abbildung zu sehen, ist also jeder PoI ein potenzieller Kandidat, wenn er den maximalen Abstand d zum PoI in der Mitte in Länge und Breite nicht überschreitet.

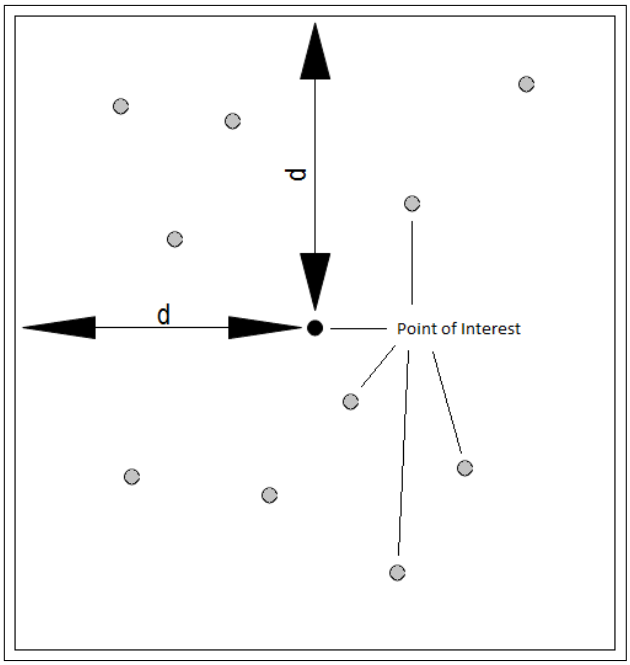


Abbildung 3.1: Bounding Box mit Matching-Kandidaten für Geofilterung von PoIs

Mit diesem Filter werden von vornherein PoIs ausgeschlossen, die von ihrer geografischen Lage her nicht infrage kommen. Dadurch werden unnötig falsche Matches vermieden.

3.4.3 TF-IDF-Wichtung mit Cosinus-Maß

Tabelle 3.2: Häufigkeitsmatrix

	restaurant	abc	klm	xyz
Restaurant Abc	1	1	0	0
Restaurant Abc	1	1	0	0
Restaurant Klm	1	0	1	0
Restaurant Xyz	1	0	0	1
	4	2	1	1

In Kapitel 2.4.4 wurde der generelle Ablauf bei der TF-IDF-Wichtung erläutert. Im Umgang mit PoIs muss man dabei bedenken, dass die Dokumente nur die Titel eines jeden PoIs umfassen. Dabei kann es sich um sehr kurze Zeichenketten handeln, die zum Teil nur aus einem Wort bestehen. Wie der Ablauf aussehen könnte, wird anhand des folgenden Beispiels erläutert. Die Liste der möglichen Kandidaten wurde auf die drei PoIs mit den Namen „Restaurant Abc“, „Restaurant Klm“ und „Restaurant Xyz“ eingegrenzt. Gesucht wird der Match zu dem PoI „Restaurant Abc“, wie er in einem anderen Netzwerk hinterlegt ist. Zunächst wird die Termliste erstellt. Diese sieht wie folgt aus: $\{restaurant, abc, klm, xyz\}$.

Als nächstes wird die Häufigkeitsmatrix aufgestellt, um die Häufigkeiten der Vorkommen für die spätere Berechnung verfügbar zu haben. Dabei wird das Anfragedokument als erste Zeile mit dazu geschrieben. Die letzte Zeile dient ebenfalls der vereinfachten Berechnung, denn in ihr steht die Anzahl an Dokumenten, die diesen Term beinhalten. Das Ergebnis für dieses Beispiel ist in Tabelle 3.2 zu sehen. Die Einträge entsprechen der Vorkommenshäufigkeit $tf_{t,d}$ des Terms t in dem Dokument d.

Tabelle 3.3: Dokument-Term-Matrix mit den Wichtungen

	restaurant	abc	klm	xyz
Restaurant Abc	0	0,301	0	0
Restaurant Abc	0	0,301	0	0
Restaurant Klm	0	0	0,602	0
Restaurant Xyz	0	0	0	0,602

Mithilfe der Gleichung 2.2 können jetzt die Wichtungen berechnet werden. Als Beispiel wird die Wichtung des Terms „restaurant“ in dem Dokument „Restaurant Abc“ bestimmt. Die Vorkommenshäufigkeit können wir der Tabelle 3.2 entnehmen, also $tf_{restaurant,RestaurantAbc} = 1$. Die inverse Dokumenthäufigkeit *idf* berechnet sich aus dem Logarithmus des Quotienten aus Anzahl aller Dokumente und Anzahl der Dokumente, die den Term „restaurant“ enthalten. Also ist $idf_{restaurant} = \log \frac{4}{4} = \log 1 = 0$. Die Wichtung ergibt sich aus dem Produkt von Vorkommenshäufigkeit und inverser Dokumenthäufigkeit, ist in diesem Fall also 0. Alle Wichtungen für dieses Beispiel sind in Tabelle 3.3 zu finden. Als erste Zeile wird neben den anderen Dokumenten noch das Anfragedokument mit einbezogen, da es die Berechnung der Ähnlichkeiten von jedem Dokument mit dem Anfragedokument, erheblich vereinfacht.

Für die Berechnung der Ähnlichkeiten wurde das Cosinus-Maß verwendet, das bestimmt wie ähnlich sich zwei Vektoren sind. Die Vektoren hierbei sind die Zeilen der Dokument-Term-Matrix mit den Wichtungen der verschiedenen Terme. Die Ähnlichkeit liegt im Intervall $[-1; 1]$. Das ist die Gleichung zur Berechnung des Cosinus-Maßes von einem Vektor/Dokument w_i mit dem Anfrage-(vektor/dokument) q (siehe [Fer03]):

$$cos\,(w_i, q) = \frac{\sum_{k=1}^{n} w_{k,i} q_k}{\sqrt{\left(\sum_{k=1}^{n} w_{k,i}^2\right)} \sqrt{\left(\sum_{k=1}^{n} q_k^2\right)}} \tag{3.1}$$

Als Beispiel wird die Ähnlichkeit des Dokuments „Restaurant Klm“ und des Dokuments

"Restaurant Abc“ mit dem Anfragedokument, berechnet:

$$cos\left(w_{RestaurantKlm}, q\right) = \frac{(0 \cdot 0) + (0 \cdot 0) + (0 \cdot 0) + (0 \cdot 0)}{\sqrt{0,602^2} \cdot \sqrt{0,301^2}} = 0 \tag{3.2}$$

$$cos\left(w_{RestaurantAbc}, q\right) = \frac{(0 \cdot 0) + (0,301 \cdot 0,301) + (0 \cdot 0) + (0 \cdot 0)}{\sqrt{0,301^2} \cdot \sqrt{0,301^2}} = 1 \tag{3.3}$$

Das Dokument „Restaurant Abc“ hat eine errechnete Ähnlichkeit von 1. Da 1 das Maximum ist, bedeutet das, dass das Dokument und das Anfragedokument sich sehr ähnlich sind und die dazu gehörenden PoIs vermutlich das selbe Objekt repräsentieren.

3.5 Hybridisierungen

Um eine bessere Performanz zu erreichen, ist es oft hilfreich mehrere Verfahren miteinander zu kombinieren, um so ein besseres Ergebnis zu erzielen. Burke hat in [Bur02] S. 6ff. mögliche Strategien vorgestellt, wie man Empfehlungssysteme miteinander verknüpfen kann. Drei, für dieses Problem adaptierte, mögliche Hybridisierungsmethoden werden an dieser Stelle kurz vorgestellt.

Wichtungen

Hierbei wird das Ergebnis aus den Vorschlägen aller Systeme berechnet. Dabei fließt jedes Ergebnis eines Systems mit einer bestimmten Wichtung ein. Die Wichtung kann statisch gewählt werden oder man lässt die Wichtungen während des Ablaufs dynamisch bestimmen. Für diese Methode eignen sich vor allem Maße, bei denen die Ergebnisse Werte bereitstellen, wie Edit-Distance, TF-IDF oder Bigram-Distance.

Wechsel

Ein anderes Mittel wäre, je nach Situation ein anderes Verfahren zu benutzen. Das könnte zum Beispiel abhängig davon sein, welche Daten für ein Objekt vorhanden sind. Sollte zum Beispiel der Fall eintreten, dass für einen PoI, für den ein Match gesucht wird, keine Koordinaten vorhanden sind, so wäre der Einsatz eines Geofilters überflüssig. Desweiteren könnte man auch je nach Beschaffenheit der Titel zwischen den unterschiedlichen Verfahren des Zeichenkettenvergleichens wechseln.

Kaskade

Als Kaskade versteht man die Aneinanderreihung der Verfahren. Dabei dient das Ergebnis des einen Algorithmus als Eingabe des nächsten und so weiter. Dieses System wurde auch im Zuge dieser Arbeit benutzt. Die meisten Ergebnisse kamen zustande, indem zuerst die Zeichenketten bearbeitet wurden, dann wurde der Geofilter angewandt, um die Liste der Kandidaten einzugrenzen. Zum Schluss wurde auf diese Kandidatenliste noch ein Maß zum Vergleich von Zeichenketten angewandt.

3.5.1 TF-IDF mit Edit-Distance

Ein großes Problem des TF-IDF-Algorithmus sind Tippfehler, da sie umgehend zu einem anderen Term führen, der dann eigenständig behandelt wird. Dieses Problem könnte man in den Griff bekommen, wenn man die Edit-Distance mit in den Ablauf integriert. Man könnte alle Terme als gleich deklarieren, die eine Edit-Distance von 1 haben. Auf diesem Wege würde man Tippfehler, die nur einen Buchstaben betreffen, ausschließen.

4 Implementierung

Die Implementierung wurde mithilfe von Java umgesetzt. Dazu gehören die Matching-Algorithmen, die Erweiterung des Webservice und die Erweiterung der Anwendung für Android-Mobiltelefone. Dabei bekamen die Matching-Algorithmen die Daten aus Trainings- und Testdatensätzen. Die Trainingsdaten waren bereits während der Entwicklung verfügbar, die Testdaten bekam ich erst zur Evaluierung. Die Datenbank war bereits vorhanden. Es kamen nur die Paare dazu, die als Ergebnis des Matching entstanden sind.

Um die unterschiedlichen Daten zu halten, haben die vier Netzwerke jeweils ihre eigene Klasse, die von der gemeinsamen Oberklasse *GeoReferencedBean* abgeleitet ist. Die Oberklasse besitzt die Attribute Länge, Breite und Name. Die Klassen folgen dem simplen Prinzip der *Java Beans*, das sind Klassen, die außer ihren Attributen nur Konstruktoren und Getter/Setter besitzen. Je nach Netzwerk kommen noch zusätzliche Attribute hinzu. Diese Art der Klassen eignet sich sehr gut, um die Daten der jeweiligen Objekte über Schnittstellen im Web zu transportieren. Zusätzlich ist noch eine Containerklasse vorhanden, die die Matches der Netzwerke speichert. Sie enthält drei Hashmaps für jedes der Netzwerke Qype/Facebook Places/Panoramio, jeweils mit den Einträgen des PoIs bei diesen Netzwerken als Schlüssel und dem Match von OSM als Wert, wenn denn ein passender gefunden wurde.

4.1 Matching

Das Matching umfasst im Wesentlichen drei Phasen:

1. PoIs aus der Datenbank lesen
2. den passenden Match in OSM finden
3. das Paar in die Datenbank schreiben

4.1.1 Aus der Datenbank lesen

Für jedes der drei Netzwerke Qype/Facebook Places/Panoramio gibt es zwei Dateien mit Trainings- und Testdatensätzen. In der einen Datei befinden sich je Netzwerk 34 Trainings-

datensätze mit korrekten Matchpaaren und in der anderen befinden sich 16 Testdatensätze, ebenfalls mit den korrekten Matches. Jeder dieser Datensätze bestand aus einem PoI eines der drei Netzwerke und einem passenden OSM-PoI, auf den er gematcht werden sollte. Dabei waren bei den Daten von Qype und Panoramio nur die IDs[1] zu den PoIs angegeben, da sie bereits in der Datenbank vorhanden waren. Da die PoIs von Facebook Places so noch nicht in der Datenbank vorhanden waren, standen relevante Metadaten in der Datei. Anhand der IDs werden die Objekte aus der Datenbank in Instanzen der jeweiligen Klassen geladen. Alle Daten eines Netzwerks werden als Liste repräsentiert. Mit diesen Listen und weiteren Parametern werden die Funktionen aufgerufen, die das Matching durchführen.

Das Schema, das der Datenbank zugrunde liegt, ist in Abbildung 4.1 zu sehen. Bis auf die Tabelle „matches“ waren alle anderen Tabelle bereits vorher in der Datenbank vorhanden. Dabei sind die Daten, die in der Tabelle „socialmetadata“ liegen, die PoIs aus den Netzwerken Qype, Facebook Places und Panoramio. Das Feld „sourceid“ verweist auf die Tabelle „source“, in der nur 3 Datensätze stehen, die den drei Quellen entsprechen. In der Tabelle „osm“ sind alle PoIs von OSM hinterlegt. Ein Eintrag in der Tabelle „matches“ besteht immer aus einem PoI aus der Tabelle „socialmetadata“, der durch die beiden Felder „socialid“ und „sourceid“ adressiert wird, und der optionalen ID eines PoI aus OSM, die nur vorhanden ist, wenn ein Match gefunden wurde.

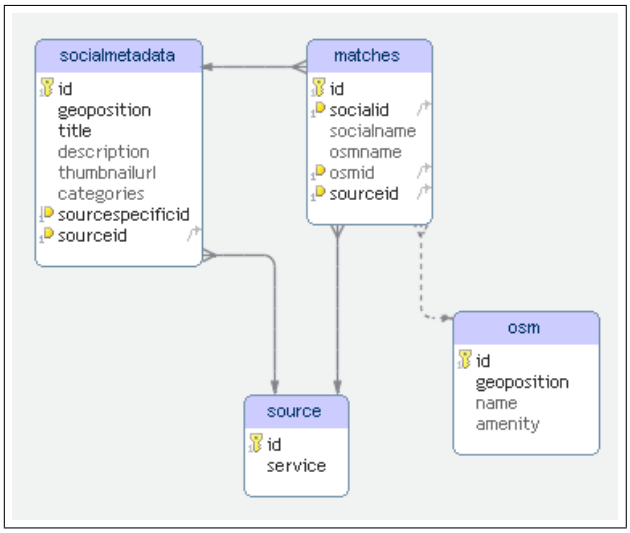


Abbildung 4.1: Datenbank-Schema

[1] künstlich zugewiesenes Merkmal zur eindeutigen Identifizierung eines Objektes

4.1.2 Match finden

Der grobe Ablauf ist bei allen Matchingverfahren der gleiche. Als Eingabe bekommt der Algorithmus drei Listen mit PoIs aus den drei Netzwerken, für die Matches gefunden werden sollen. Desweiteren gab es noch einen Parameter, der die jeweilige Funktion in ihrem Ergebnis beeinflussen soll: bei einigen Algorithmen den Mindestwert, den die Berechnung erreichen muss, bevor ein Match gültig ist. Der erste Schritt ist das Auslesen der Liste der OSM-PoIs aus der Datenbank. Bei allen Algorithmen außer den beiden Baselines wird diese Liste als nächstes mithilfe des Geofilters verkleinert. Dabei werden nur die PoIs in die neue Liste übernommen, die in der Bounding Box um den PoI herum liegen, für den der Match gesucht wird. Danach findet das eigentliche Matching statt. Der Algorithmus sucht nun den PoI von OSM, der am wahrscheinlichsten als Match infrage kommt. Sollte keiner der Kandidaten die Schwellen der jeweiligen Algorithmen überschreiten, so wird kein Match zugewiesen. Die Werte wurden nach einigen Durchläufen bestimmt und es wurden die Werte benutzt, die bei den Trainingsdaten die besten Ergebnisse erzielt haben.

Im Folgenden wird kurz darauf eingegangen, was bei den verschiedenen Algorithmen besonders war und wie sie implementiert wurden.

Longest common substring (LCS)

Die Schwelle bei diesem Verfahren, die überschritten werden musste, war die minimale Länge der Teilzeichenkette. Das beste Ergebnis wurde bei einer minimalen Länge von $0,4 \cdot n$ erzielt. Dabei ist n die Länge des Namens des PoI bei OSM ist, für den der Match gesucht wurde. Im Folgenden sieht man den Pseudo-Code für diesen Algorithmus:

```
Input: Zwei Zeichenketten s1 und s2
Output: longest common substring
longestsubstring :=""
for i := 0 to length(s1) do
  s3 :=""
  for j := 0 to length(s2) do
    while i < length(s1) and j < length(s2) and s1[i] = s2[i] do
      s3 := s3 + s1[i]
      i := i + 1
      j := j + 1
    end while
```

```
      if length(s3) > length(longestsubstring) then
        longestsubstring := s3
      end if
    end for
  end for
  return longestsubstring
```

Bestimmt wird die längste Teilzeichenkette indem jeder Buchstabe der Zeichenkette s_1 mit jedem Buchstaben der Zeichenkette s_2 verglichen wird. Wurde ein Buchstabe gefunden, der in beiden Zeichenketten vorkommt, so wird er in eine neue Zeichenkette s_3 geschrieben und es werden die darauf folgenden Buchstabe der beiden Zeichenketten verglichen. Sind auch die gleich, wird der Buchstabe zu dem ersten in s_3 geschrieben und der nächste Buchstabe wird verglichen usw. bis sich die beiden Buchstaben unterscheiden oder eine der beiden Zeichenketten zuende ist. Jetzt wird überprüft, ob es bereits eine längere Teilzeichenkette gab, als die momentane in s_3. Falls nicht, so wird s_3 in *longestsubstring* geschrieben. Falls ja, dann wird s_3 geleert und es wird weitergesucht, wenn möglich. Diese Berechnung wird mit jedem PoI in der Kandidatenliste durchgeführt. Der PoI, dessen Titel die längste Teilzeichenkette hat, die auch über die nötige Mindestlänge verfügt, ist wahrscheinlich der gesuchte Match.

Nähester PoI (NP)

Als Schwellkriterium für dieses Verfahren dient eine maximale Distanz, die nicht überschritten werden darf. Das beste Ergebnis wurde erzielt bei einem maximal erlaubten Abstand von $0,001°$, das entspricht etwa 100 Metern. Die Distanz wird, wie weiter oben bereits erläutert, mit der Euklidischen Distanz von zwei Punkten in der Ebene berechnet. Der PoI, der am wahrscheinlichsten der Match ist, ist der, der die geringste Distanz zu dem PoI hat, zu dem der Match gesucht wurde.

Geofilter, Stringpreprocessing und LCS (GSL)

Hierbei werden die Kandidaten, bevor der Match schlussendlich über den LCS gesucht wird, durch den Geofilter eingegrenzt. Außerdem werden die Sonderzeichen in den Zeichenketten ersetzt. Die Kandidatenliste setzt sich aus den PoIs zusammen, die den Abstand in Länge und Breite von $0,005°$ (etwa 500 Meter) nicht überschreiten. Diese Bounding Box wurde für alle weiteren Verfahren ebenfalls mit demselben Abstand erstellt. Die Mindestlänge der Teilzeichenkette ist dieselbe, wie bei der Baseline LCS.

Geofilter, Stringpreprocessing und Edit-Distance bzw. Bigram-Distance (GSE bzw. GSB)

Auch bei diesen Algorithmen wurden zuerst die Liste der Kandidaten mithilfe des Geofilters eingegrenzt und anschließend die Sonderzeichen bearbeitet. Für die Berechnung der Edit-Distance wurde eine Funktion der Bibliothek *Lingpipe*[2] verwendet. Als Parameter galt hierbei eine maximal erlaubte Edit-Distance von $0.4 \cdot min(n, m)$, wenn n die Länge der Zeichenkette s_1 und m die Länge der Zeichenkette s_2 ist. Wenn keine Zeichenkette ein Ergebnis unter diesem Wert erreichte, so wurde kein Match zugewiesen. Für die Berechnung der Bigram-Distance wurde eine Funktion der Bibliothek *Lucene*[3] verwendet. Das Ergebnis dieser Funktion ist die Ähnlichkeit der beiden Zeichenketten. Um als Match zu gelten, musste eine minimale Ähnlichkeit von $0,4$ erreicht werden.

Geofilter, Stringpreprocessing und TF-IDF-Wichtung (GST)

Der Algorithmus hat als Eingabe den Titel des PoIs, zu dem der Match gefunden werden soll, eine Liste mit Kandidaten aus OSM, die nach dem Geofilter als Match infrage kommen, und eine minimale Ähnlichkeit, die erreicht werden muss, damit es einen Match gibt. Diese minimale Ähnlichkeit wurde auf 0.3 festgelegt, weil dies das beste Ergebnis bei den Testdaten lieferte. Zuerst werden alle Namen der OSM-PoIs in eine Liste geschrieben. Mit dieser Liste und dem Titel des Anfragedokuments wird die Termliste erstellt. Als nächstes wird in einen zweidimensionalen Integer-Array die Häufigkeit (vgl. Tabelle 3.2) jedes Terms in jedem Dokument angegeben. Dabei entspricht die erste Zeile dem Anfragedokument und die letzte Zeile der Anzahl der Dokumente, die diesen Term enthalten. Mithilfe dieser Matrix können die Wichtungen bestimmt werden. Sie werden in einem zweidimensionalen Double-Array gespeichert (vgl. Tabelle 3.3). Dabei repräsentiert die erste Zeile wieder das Anfragedokument. Mit diesen Wichtungen kann nun für jedes Dokument die Ähnlichkeit mit dem Anfragedokument berechnet werden. Liegt das Ergebnis der Berechnung über der minimal benötigten Ähnlichkeit, die über den Parameter angegeben wurde, dann wurde ein möglicher Match gefunden. Das Dokument mit der höchsten Ähnlichkeit repräsentiert den PoI, der mit hoher Wahrscheinlichkeit der gesuchte ist.

[2] http://alias-i.com/lingpipe/
[3] http://lucene.apache.org/java/docs/index.html

4.1.3 Paar eintragen

Am Schluss des Matching-Prozesses müssen die Paare in die Datenbank eingetragen werden, um für die Anwendung verfügbar zu sein. Dabei werden folgende Metadaten gespeichert: die ID des PoIs in dem Netzwerk Qype/Panoramio/Facebook Places, der Titel in diesem Netzwerk, die ID, aus welcher Quelle der PoI stammt (1, 2 oder 3), der Titel in OSM und die ID in OSM.

4.2 Webservice mit JSON-Rückgabe

Der Webservice, der im Rahmen dieser Arbeit benutzt wurde, wurde bereits während des Praktikums entwickelt. Dieser musste lediglich um eine Funktion erweitert werden, die die Matches zurückgibt. Wenn diese Funktion über *HTTP GET* aufgerufen wird, liefert sie alle Matches, die in der Datenbank stehen. Dabei werden zuerst die Matches für jedes der drei Netzwerke separat aus der Tabelle gelesen. Da sie unterschiedliche Daten bereitstellen, müssen sie auch unterschiedlich behandelt werden. Man benötigt dafür die beiden IDs. Im Bezug auf das Schema in Abbildung 4.1 sind das die beiden Felder „socialid“ und „osmid“ der Tabelle „matches“. Das Feld „socialid“ referenziert dabei einen Eintrag in der Tabelle „socialmetadata“, in der die PoIs von Qype/Facebook Places/Panoramio stehen. Das Feld „osmid“ referenziert den passenden Match in der Tabelle „osm“. Der nächste Schritt ist die Anfrage an die Datenbank, die Daten dieser PoIs auszugeben. Die Daten werden dann in eine Instanz der Beanklasse für das jeweilige Netzwerk geschrieben, und anschließend als Schlüssel-Wert-Paar in die Hashmap eingetragen. Sind diese Schritte für alle drei Netzwerke erledigt, wird der Container im JSON-Format[4] an den Dienst zurückgegeben, der die Funktion aufgerufen hat.

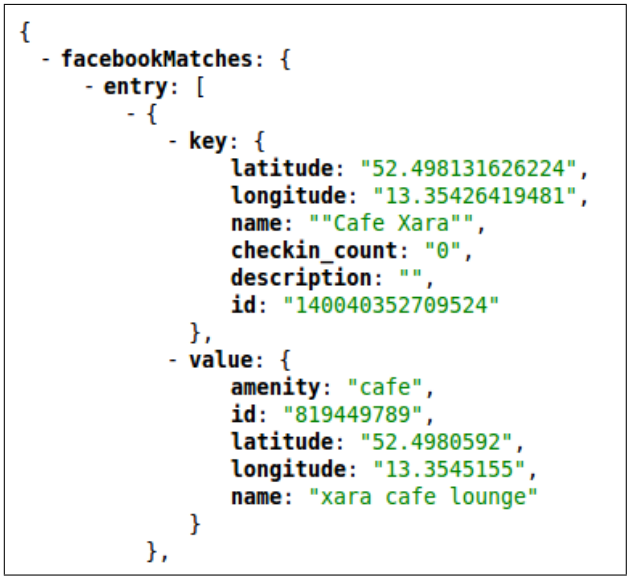
4.3 Android-Application

Die Anwendung, die benutzt wurde, wurde im Rahmen des Projekts Voice2Social am DFKI entwickelt. Für diese Arbeit wurde sie erweitert. Die Anbindung an einen beliebigen RESTful-Webservice[5] (siehe auch [Fie00]) über eine URL, die Umwandlung der Rückgabe des Webservices in eine Zeichenkette und die Darstellung der Google-Karte waren bereits

[4] JavaScript Object Notation - kompaktes Datenformat zum Zweck des Datenaustauschs
[5] REST (representational state transfer) bezeichnet einen Softwarearchitekturstil und dient der Bereitstellung von Schnittstellen über HTTP

implementiert und konnten weiterverwendet werden. Die folgenden Dinge mussten dementsprechend selbstständig hinzugefügt werden:

- JSON-Rückgabe interpretieren
- Overlay mit PoIs generieren
- Informationen beim „Antippen“ eines PoIs anzeigen



```
{
  - facebookMatches: {
      - entry: [
          - {
              - key: {
                    latitude: "52.498131626224",
                    longitude: "13.35426419481",
                    name: ""Cafe Xara"",
                    checkin_count: "0",
                    description: "",
                    id: "140040352709524"
                },
              - value: {
                    amenity: "cafe",
                    id: "819449789",
                    latitude: "52.4980592",
                    longitude: "13.3545155",
                    name: "xara cafe lounge"
                }
            },
```


Abbildung 4.2: Beispiel einer JSON-Rückgabe (Facebook PoI), erstellt mit dem Mozilla Firefox Addon JSONView

JSON-Rückgabe interpretieren

In Abbildung 4.2 sieht man solch eine Rückgabe von dem Webservice (das Zeichen “-“ dient nur der Visualisierung und gehört nicht zum JSON-Format). Diese Rückgabe ist ein Objekt mit den drei Hashmaps „facebookMatches“, „qypeMatches“ und „panoMatches“. Jede dieser Hashmaps enthält eine Liste (mit Namen „entry“) von Objekten, die jeweils aus einem Schlüssel und einem optionalen Wert bestehen. In dem Fall (Abb. 4.2) ist der Schlüssel ein Facebook Places PoI und der Wert ist der gematchte PoI von OSM. Beide PoIs liefern alle verfügbaren Daten mit Titel des Attributs und dessen Inhalt.

Overlay mit PoIs generieren

Overlays sind Grafiken, die man auf die Karte legt, um so Markierungen, Kreise, Linien oder etwas anderes auf der Karte zu veranschaulichen. Zuerst muss dafür eine Instanz der Klasse

Overlay oder einer abgeleiteten Klasse erstellt werden. Diesem Overlay werden dann Overlay-Items hinzugefügt, das sind die Markierungen, die dann auf der Karte angezeigt werden. Für jeden PoI muss dabei ein einzelnes Item erzeugt werden. Dieses Item bekommt sowohl Länge, Breite und Titel des PoIs, als auch alle restlichen verfügbaren Informationen, die zu diesem PoI vorhanden sind. Auf der Abbildung 4.3 sieht man den selben Kartenausschnitt einmal ohne (links) und einmal mit Overlay (rechts). Die Abbildung 4.4 zeigt zwei weitere Kartenausschnitte, einen etwas dichter (links) und der andere zeigt den Tiergarten in Berlin (rechts).

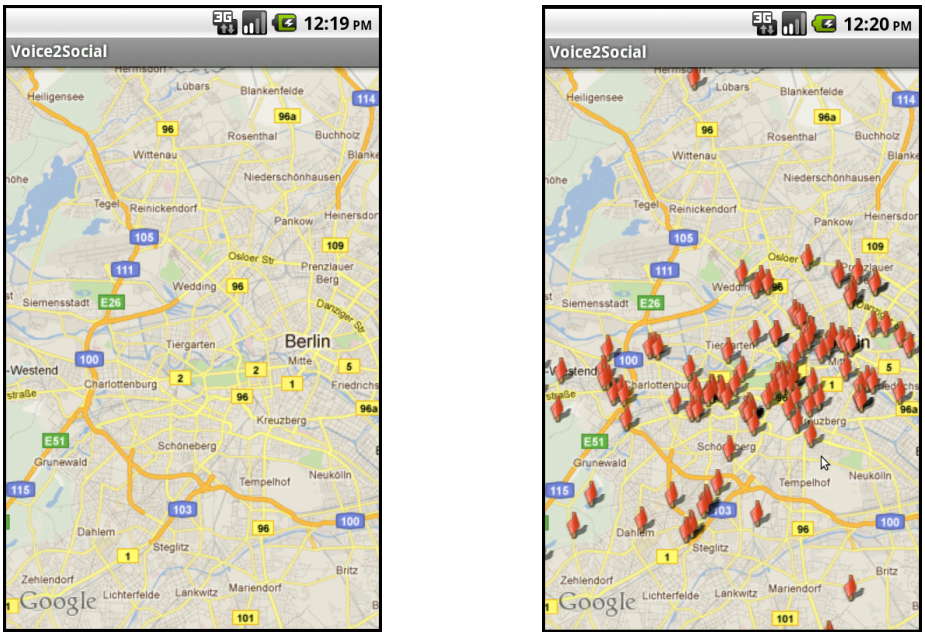


Abbildung 4.3: links: Karte mit einem Ausschnitt Berlins ohne Matches, rechts: der gleiche Ausschnitt mit Matches

Informationen des PoIs anzeigen

Beim Berühren einer Flagge auf der Karte, die einen PoI symbolisiert, sollen alle verfügbaren Daten zu dem PoI angezeigt werden. Dazu muss die Funktion „onTap()“ der abgeleiteten Overlay-Klasse überschrieben werden. Sobald ein Punkt berührt wurde, so wechselt die Anwendung auf eine neue Seite mit den Informationen. Die Abbildung 4.5 zeigt jeweils eine solche Seite von Qype (links) und Panoramio (rechts).

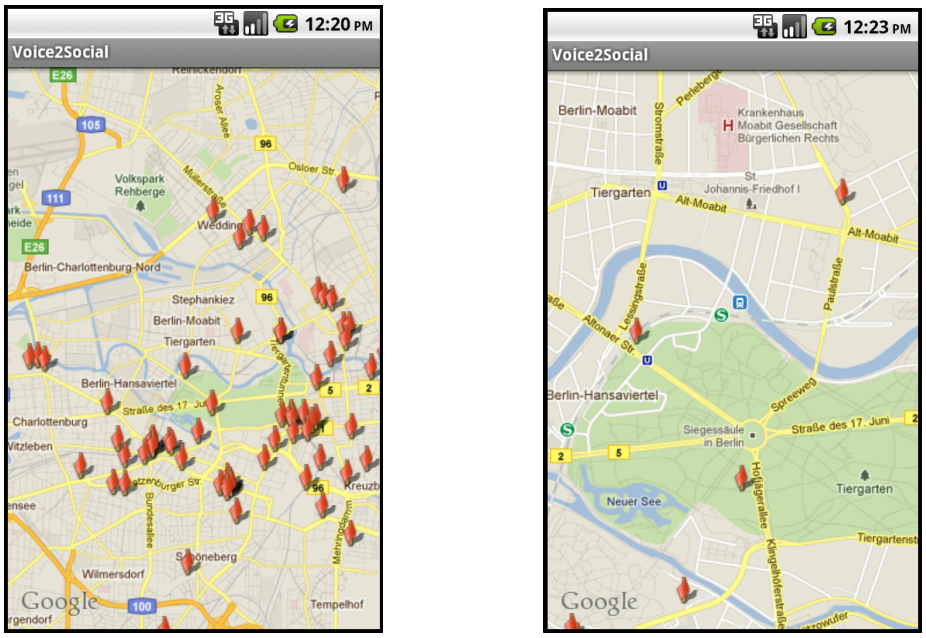


Abbildung 4.4: links: Ergebnisse vom TF-IDF-Matching bei einer niedrigeren Zoomstufe, rechts: Matches im Berliner Tiergarten

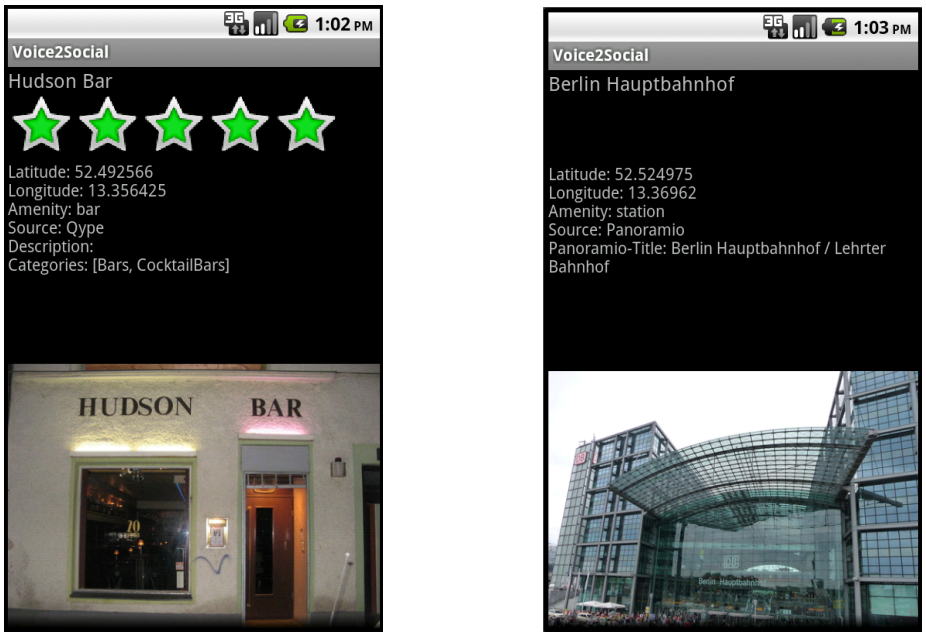


Abbildung 4.5: links: Beispiel eines Matches von Qype auf OSM, rechts: Beispiel eines Matches bei Panoramio

5 Evaluierung

Das Ziel war ein Verfahren bzw. eine Kombination von Verfahren zu finden, das bzw. die die höchstmögliche Genauigkeit bietet. Außerdem sollen dabei falsche Matches möglichst vermieden werden.

5.1 Mögliche Fehler

Beim Matching von PoIs gibt es prinzipiell drei Fehlertypen, auf die näher eingegangen werden muss. Das sind zum einen Points of Interest, die keinen passenden Match haben, aber trotzdem einen gefunden haben. Es sind PoIs, die einen passenden Match besitzen, aber es wurde ein falscher zugewiesen. Als letztes gibt es Points of Interest, die eigentlich einen Vertreter in einem anderen Netzwerk hätten, aber es wurde kein passender Match gefunden. Prinzipiell könnte man die ersten beiden Fehler zu einem zusammenfassen, bei denen es sich um falsche Matches handelt.

Die Raten für die beiden resultierenden Fehlertypen hängen unmittelbar zusammen. Um das Auftreten des einen Fehlers zu vermindern, muss man ein Ansteigen des Auftretens des anderen in Kauf nehmen. Für den vorhandenen Zweck, Points of Interest aus verschiedenen Netzwerken zusammenzulegen, ist der erste Fehlertyp, also die falschen Matches, sehr schlecht. Der zweite Fehlertyp ist im Gegenteil dazu vertretbar, insofern er nicht zu verhindern ist bzw. so dass die Rate an falschen Matches nicht steigt. Wenn man falsche Matches hat, dann vermischen sich die Daten von zwei völlig verschiedenen Punkten aus zwei Netzwerken, da man teils Daten aus dem einen Netzwerk und teils Daten aus dem anderen Netzwerk verwendet. Es entsteht ein neuer PoI, der eine Mischung der beiden ursprünglichen ist. Der Nachteil der Nicht-Matches ist, dass dann zwei Punkte auf der Karte direkt beieinander liegen, die den selben PoI repräsentieren. Das führt nur dazu, dass die Übersichtlichkeit der Anwendung eventuell abnimmt.

5.2 Ergebnisse

In diesem Kapitel werden die Ergebnisse der verschiedenen Verfahren miteinander verglichen. Es werden fünf Fälle unterschieden, in die ein gefundener Match fallen kann. Die Ergebnisse werden in der Form „A → B“ dargestellt. Der linke Teil des Ergebnistyps bezeichnet dabei immer das richtige Ergebnis und der Teil rechts neben dem „→“ bezeichnet das Ergebnis, das erzielt wurde. Es ergeben sich folgende Fälle:

- richtiger Match - „X → X“
- richtig, dass kein Match zugewiesen wurde - „null → null“
- kein Match wurde zugewiesen, obwohl es einen gibt - „X → null“
- falscher Match - „X → Y“
- Match wurde zugewiesen, obwohl es keinen gibt - „null → Y“

An dieser Stelle noch ein paar Beobachtungen zu den Schwellwerten und welchen Einfluss sie auf das Ergebnis hatten: Wie man vermuten könnte, gab es bei den Algorithmen immer weniger falsche Matches, wenn man die Schwellwerte für die Algorithmen herauf gesetzt hat. Das hatte aber auch zur Folge, dass die richtigen Ergebnisse abnahmen und mehr Ergebnisse entstanden, wo kein Match zugewiesen wurde, obwohl einer vorhanden wäre. Auffällig war auch, dass bei verringerten Schwellwerten die Anzahl an richtigen Matches nicht unbedingt zunahm, nur die falschen Matches wurden häufiger. Bei allen Verfahren, außer dem Verfahren des Nähesten PoIs, gibt es einen Bereich für den Schwellwert zwischen $0,3$ und $0,6$, bei dem die Ergebnisse fast identisch ausfallen und sich kaum etwas ändert. Das liegt wahrscheinlich daran, dass die meisten richtigen Matches einen weitaus höheren Wert bei der Ähnlichkeit erreichen und die meisten falschen nicht ähnlich genug sind, um diesen Wert zu erreichen.

Im Folgenden werden die Ergebnisse vorgestellt, je Netzwerk in den Tabellen 5.1 bis 5.3. Die Zahl in der linken oberen Ecke jeder Tabelle besagt, wieviele Datensätze in der Tabelle in der Datenbank vorhanden waren, die erste steht dabei für die Menge der Testdaten und die zweite für die Menge der Trainingsdaten. Die Werte in den Tabellen sind auf die selbe Weise repräsentiert. Die Ergebnisse des Matchings bei Panoramio umfassen im Gegensatz zu den anderen Netzwerken nur 33 Datensätze. Die restlichen 17 Datensätze wurden durch den in der Zeit des Praktikums geschriebenen Crawler nicht gefunden und fehlen dementsprechend in der Datenbank.

Die Verfahren, die betrachtet wurden, waren die Baseline mit dem naiven Herangehen über den *longest common substring* (LCS), die alternative Baseline mit dem nähesten PoI (NP),

die Erweiterung des *longest common substring* um Geofilter und Stringpreprocessing (GSL), die Edit-Distance (GSE), die Bigram-Distance (GSB) und die Wichtung mit TF-IDF (GST). Die letzte Spalte entspricht der Genauigkeit des jeweiligen Verfahrens, das ist der Quotient aus Anzahl an richtigen Matches und der Anzahl der Datensätze. Sie beschreibt, wieviel Prozent der Ergebnisse richtig waren.

Tabelle 5.1: Facebook Places Ergebnisse

16|34	X → X	null → null	X → null	X → Y	null → Y	Genauigkeit in %
LCS	6|17	0|0	0|3	7|10	3|4	38|50
NP	2|11	2|2	3|7	8|12	1|2	25|38
GSL	4|19	3|3	5|5	4|6	0|1	44|65
GSE	2|13	3|3	9|13	2|4	0|1	31|47
GSB	2|16	3|3	10|12	1|2	0|1	31|56
GST	8|21	3|4	4|9	1|0	0|0	69|74

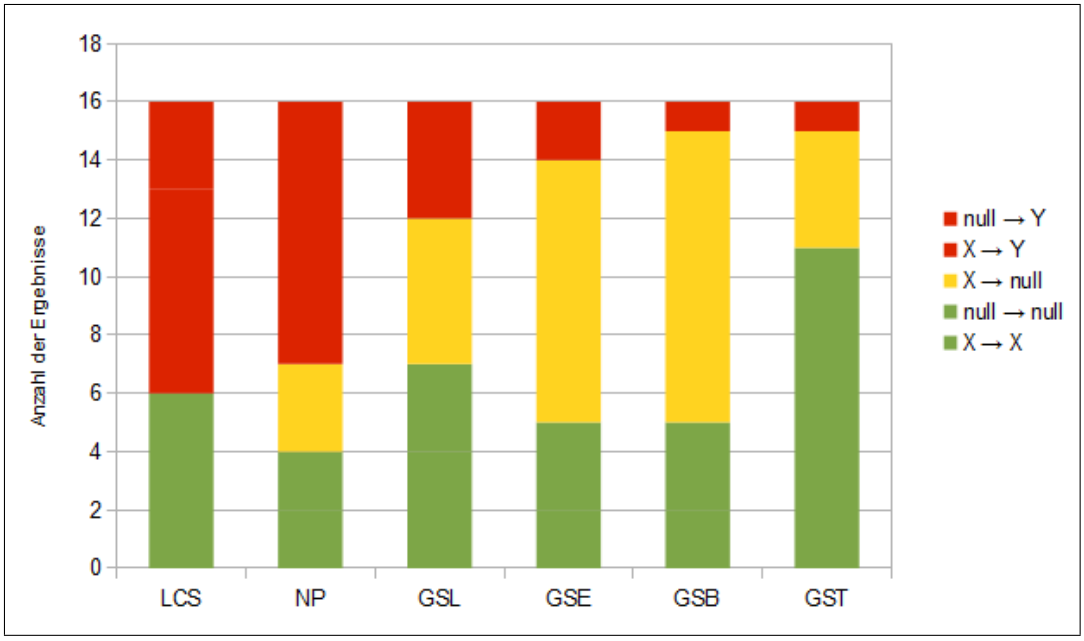


Abbildung 5.1: Ergebnisse der Testdaten von Facebook Places, Grün = richtige Matches, Gelb = nicht gefundene Matches, Rot = falsche Matches

Tabelle 5.1 zeigt die Resultate des Matchings für das Netzwerk Facebook Places. Die Zeilen repräsentieren dabei jeden der verwendeten Algorithmen (siehe Kapitel 4.1.2) und die Spalten entsprechen den fünf Fällen. Auf den ersten Blick fällt auf, dass das TF-IDF-Verfahren das beste Ergebnis erzielt hat, es hat die meisten richtigen Matches (die ersten beiden Spalten) und die wenigsten falschen (Spalten 3 bis 5). Dabei gelten die Fehler in Spalte 3 (X → null), wie in Kapitel 5.1 beschrieben, als am wenigsten gravierend, denn sie führen nur dazu, dass

auf der Karte eventuell zwei Punkte nebeneinander liegen, wohingegen die anderen beiden Fehler dazu führen, dass falsche Resultate entstehen.

Die beiden Verfahren Edit-Distance und Bigram-Distance schneiden, was die richtigen Matches angeht, schlechter ab als die Baseline. Das Problem der Baseline sind allerdings die vielen falschen Matches, die das Ergebnis, trotz vieler richtiger Matches, schlecht machen.

Außerdem ist zu sehen, dass die alternative Baseline schlechter abschneidet als die erste Baseline. Sie hat weniger richtige Matches und mehr falsche. Zusätzlich fällt auf, dass die falschen Matches erheblich abnehmen, sobald mehrere Systeme eingesetzt werden. Die beiden Baselines liefen nur alleine, bei den anderen Verfahren kamen Geofilter und Stringpreprocessing dazu, die das Ergebnis dann zumindest weniger falsch gemacht haben, wodurch das Ergebnis besser wurde, trotzdem es nicht unbedingt mehr richtige Matches gab.

Die Abbildung 5.1 zeigt die Ergebnisse der Testdaten in einem Balkendiagramm. Dabei zählen die richtigen Matches zu dem grünen Teil eines Balkens, der gelbe Teil repräsentiert den Fall „X → null“ und der rote Teil eines Balkens bezeichnet die Fehler der Algorithmen, wo wirklich etwas falsches zugewiesen wurde. Man sieht, dass durch die Zuhilfenahme des Geofilters und des Stringpreprocessings bei dem *longest common substring* kaum richtige Matches dazugekommen sind, aber es gab danach wesentlich weniger falsche, was das Ergebnis schon sehr verbessert.

Tabelle 5.2: Qype Ergebnisse

16|33	X → X	null → null	X → null	X → Y	null → Y	Genauigkeit in %
LCS	10|17	1|0	1|3	4|9	0|4	69|52
NP	9|10	0|0	3|5	3|14	1|4	56|30
GSL	10|18	1|3	5|10	0|1	0|1	69|64
GSE	9|15	1|4	6|13	0|1	0|0	63|58
GSB	9|17	1|4	6|10	0|2	0|0	63|64
GST	11|21	1|4	4|7	0|1	0|0	75|73

Tabelle 5.2 zeigt die Ergebnisse von Qype. Für Qype waren nur 49 Datensätze verfügbar, da der 50. PoI über die Qype-API nicht erreichbar ist und deshalb nicht in der Datenbank vorhanden war. Wie bei Facebook Places ist auch hier zu sehen, dass das Verfahren TF-IDF dasjenige war, was am besten abschnitt. Es gibt hier ebenfalls nur einen einzigen wirklichen Fehler (bei allen 49 Datensätzen). Bei diesen Ergebnissen ist auch zu sehen, dass die erste Baseline bessere Ergebnisse liefert, als die alternative. Die beiden Distanz-Verfahren haben auch bei Qype weniger richtige Matches als die Baseline, dafür aber auch deutlich weniger falsche.

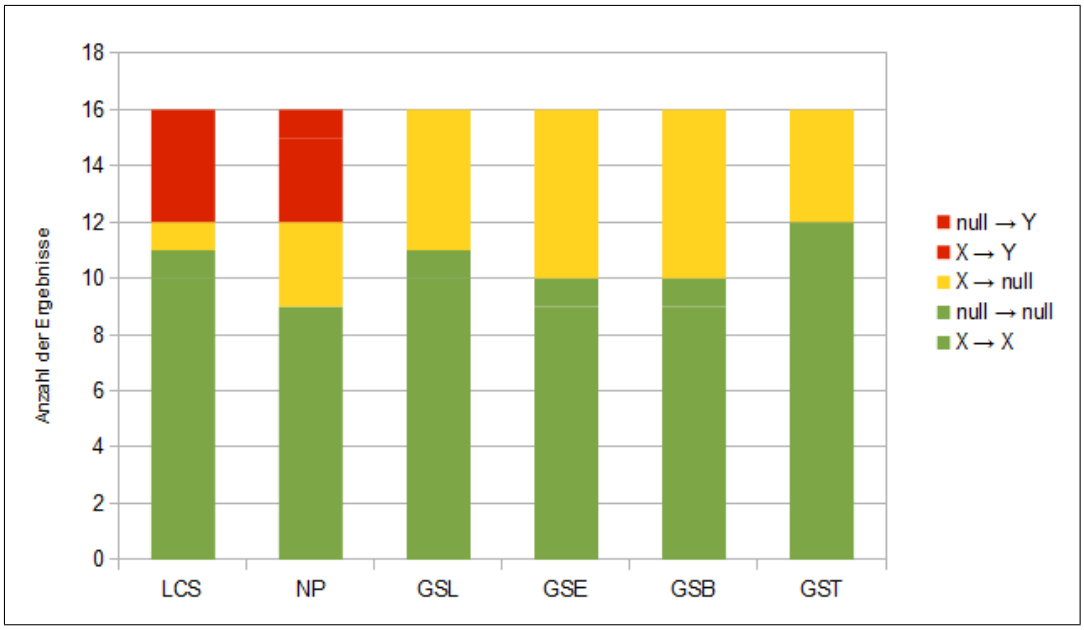


Abbildung 5.2: Ergebnisse der Testdaten von Qype

In Abbildung 5.2 sind die Ergebnisse der Testdaten visualisiert. Man sieht, dass sich die Anzahl an richtigen Matches bei allen Algorithmen nicht nennenswert unterscheidet (zwischen 9 und 11). Die Fehler, die in den beiden Baselines auftreten, wurden bei den Erweiterungen korrigiert. Statt eines falschen Matches, wurde diesen PoIs kein Match mehr zugewiesen.

Tabelle 5.3: Panoramio Ergebnisse

11|22	X → X	null → null	X → null	X → Y	null → Y	Genauigkeit in %
LCS	0|7	0|0	3|9	7|5	1|1	0|32
NP	1|2	1|1	4|6	5|13	0|0	18|14
GSL	1|7	1|1	7|11	2|3	0|0	18|36
GSE	0|3	1|1	9|16	1|2	0|0	9|18
GSB	0|5	1|1	9|16	1|0	0|0	9|27
GST	3|5	1|1	7|13	0|3	0|0	36|27

Die Ergebnisse von Panoramio in Tabelle 5.3 gründen nur auf 33 Datensätzen, denn 17 PoIs waren nicht in der Datenbank vorhanden. Es waren 22 Datensätze in den Trainingsdaten und 11 in den Testdaten verfügbar. Aber selbst die kleine Menge verdeutlicht bereits das Problem, das beim PoI-Matching in der Panoramio-Plattform auftritt. Alle Verfahren schneiden wesentlich schlechter ab, als bei den beiden anderen Netzwerken, was vor allem an den Titeln liegt, die sehr verschieden von den OSM-Titeln sind. Das führt dazu, dass sehr viele PoIs einfach keinen Match finden (26 von 33 bei Edit-Distance). Dieses Problem zeigt sich auch in der Visualisierung der Testdaten in Abbildung 5.3. Die gelben Balken machen

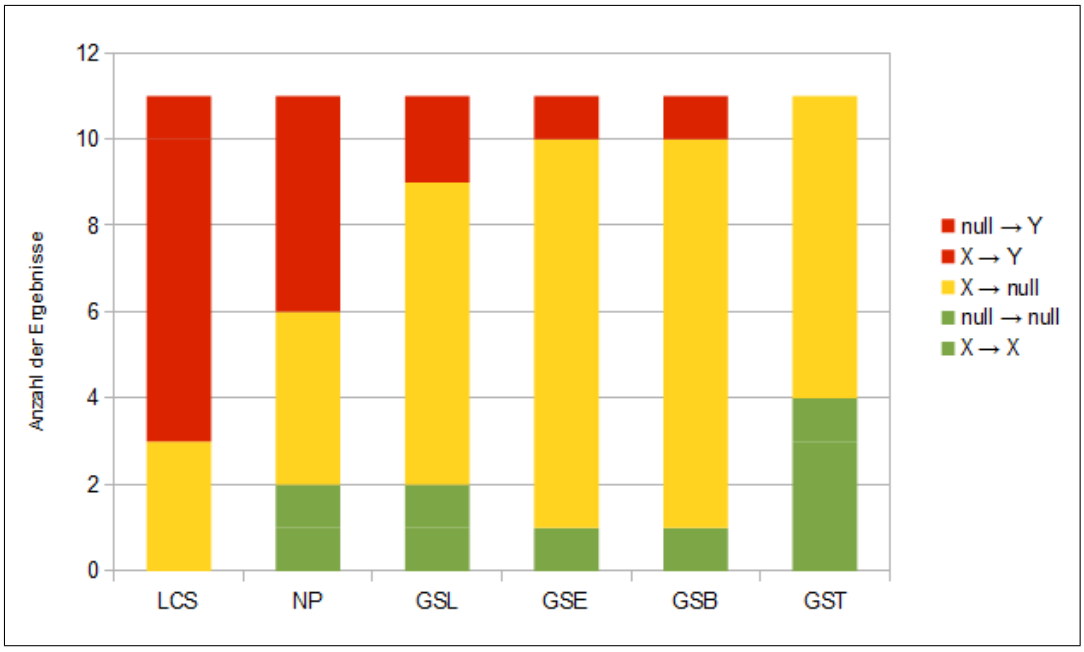


Abbildung 5.3: Ergebnisse der Testdaten von Panoramio

die PoIs kenntlich, zu denen kein Match gefunden wurde. Wie bei den Ergebnissen von Facebook Places ist hier allerdings auch zu erkennen, wie die Anzahl der Fehler mit der Weiterentwicklung der Verfahren abgenommen hat. Das Ergebnis von Panoramio hätte mit Sicherheit noch verbessert werden können, wenn man für dieses Netzwerk mit den Schwellwerten mehr experimentiert hätte. In Abbildung 5.4 wird die Genauigkeit aller Algorithmen und Netzwerke zusammengefasst.

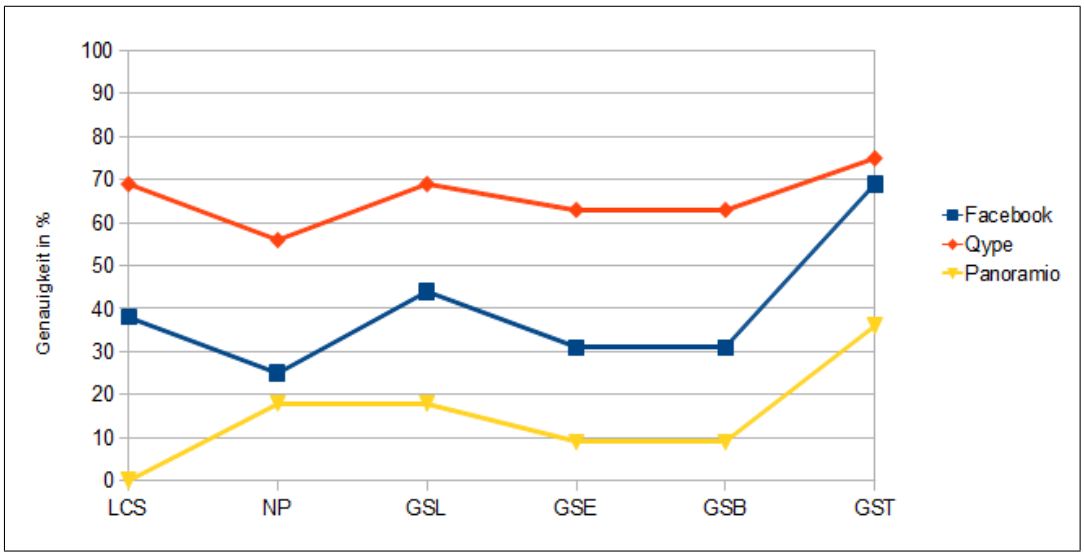


Abbildung 5.4: Genauigkeit der Ergebnisse für die drei Netzwerke mit den sechs Algorithmen

5.2.1 Fazit

Das Verfahren, dass durchweg die besten Ergebnisse erreicht hat, war TF-IDF in Kombination mi dem Geofilter und Stringpreprocessing. Es führte zu den wenigsten falschen Matches und den meisten richtigen. Allerdings waren die vorhandenen Titel der PoIs weitestgehend fehlerfrei, was diesem Verfahren zugute kommt. Bei größeren Datenmengen sollte über eine mögliche Verfeinerung dieses Verfahrens nachgedacht werden, beispielsweise wie die Erweiterung mit der Edit-Distance, um mögliche Tippfehler zu berücksichtigen. Überraschend war das Ergebnis des *longest common substring*. Dieses Verfahren ergab sehr viele richtige Matches, allerdings auch sehr viele falsche. Durch die Irrelevanz der Reihenfolge der Wörter, hat TF-IDF einen großen Vorteil, was die Arbeit mit Zeichenketten unterschiedlicher Quellen angeht. Dem Ziel, eine ordentliche Lösung für das Problem der PoIs verschiedener Netzwerke zu finden, kommt man mit dem Verfahren TF-IDF ziemlich nahe, allerdings hat es, wie alle anderen Verfahren auch, seine Nachteile, mit Tippfehler nicht umgehen zu können. Wenn ein Weg gefunden wird, mit diesen Nachteilen umzugehen, dann könnte das Ergebnis noch weiter verbessert werden.

6 Zusammenfassung

Im Rahmen der Bachelorarbeit wurden Verfahren vorgestellt, um Points of Interest in OSM, wie Restaurants, Bars, Cafés usw., mit ihren Entsprechungen in anderen Netzwerken zu matchen.

Dabei wurden zuerst die vier Netzwerke vorgestellt, die in dieser Arbeit involviert waren, OSM, Qype, Facebook Places und Panoramio. Es wurde auf die Problematik der Datenheterogenität eingegangen, die auftritt, wenn Daten zusammengeführt werden sollen, die auf unterschiedliche Weise repräsentiert sind. Anschließend wurden einige Verfahren vorgestellt, mit denen man die Ähnlichkeiten zwischen verschiedenen Feldern solcher Daten feststellen kann.

Im folgenden Kapitel wurde erläutert, welche Daten vorhanden waren, wie die vorhandenen Verfahren angepasst werden mussten, um Attribute von Points of Interest zu vergleichen, um so zu bestimmen, wie ähnlich sich PoIs sind. In diesem Kapitel wurde auch darauf eingegangen, wie die anfänglichen Algorithmen erweitert wurden, um bessere Ergebnisse zu erzielen.

Im Kapitel Implementierung wurde beschrieben, wie die Verfahren umgesetzt wurden und welche Rolle die Datenbank und der Webservice bei der Android-Anwendung spielten.

Im letzten Kapitel, der Evaluierung, wurde vorgestellt, welche Fehler beim Matching von PoIs auftreten können. Die unterschiedlichen Algorithmen wurden hinsichtlich dieser Fehler untersucht. Das beste Ergebnis wurde von TF-IDF erzielt, das die meisten richtigen Matches und die wenigsten falschen aufwies.

6.1 Ausblick

Im nächsten Schritt muss die Android-Anwendung bearbeitet werden. Sie muss die Matches, die dargestellt werden sollen, auf den momentan dargestellten Kartenausschnitt begrenzen. Ferner soll auch die Art der PoIs eingrenzbar sein. Für den Matching-Algorithmus TF-IDF sollte eine mögliche Erweiterung in Betracht gezogen werden, um die Tippfehler zu berücksichtigen, wie beispielsweise die Erweiterung mit der Edit-Distance, die im Rahmen dieser Arbeit vorgestellt wurde. Ebenfalls sollte das Matching auch auf die gesamte Datenbank erfolgen und die Matches in der dafür vorgesehenen Tabelle verfügbar gemacht werden.

Literaturverzeichnis

[Bur02] Burke, R.: Hybrid Recommender Systems: Survey and Experiments. In: *User Modeling and User-Adapted Interaction* (2002)

[EIV07] Elmagarmid, A. K. ; Ipeirotis, P. G. ; Verykios, V. S.: Duplicate Record Detection: A Survey. In: *Knowledge and Data Engineering, IEEE Transactions* 19 (2007)

[Fer03] Ferber, R.: *Information Retrieval - Suchmodelle und Data-Mining-Verfahren für Textsammlungen und das Web*. dpunkt.verlag, 2003

[Fie00] Fielding, R. T.: *Architectural styles and the design of network-based software architectures*, PhD Thesis, 2000

[Kon05] Kondrak, G.: N-gram similarity and distance. In: *Proceedings of the Twelfth International Conference on String Processing and Information Retrieval (SPIRE 2005)*, 2005

[kow07] kowoma.de: *Erreichbare Genauigkeit bei GPS-Positionierung*. `http://www.kowoma.de/gps/Genauigkeit.htm`. Version: 2007. – [Online; besucht 28-Juli-2011]

[MRS08] Manning, C. D. ; Raghavan, P. ; Schütze, H.: *Introduction to Information Retrieval*. Cambridge University Press, 2008

[OW11] OSM-Wiki: *Genauigkeit von Koordinaten*. `http://wiki.openstreetmap.org/w/index.php?title=DE:Genauigkeit_von_Koordinaten&oldid=586205`. Version: 2011. – [Online; besucht 13-Juli-2011]

[SJ72] Spark Jones, K.: A statistical interpretation of term specificity and its application in retrieval. In: *Journal of Documentation* 28 (1972), Nr. 1, S. 11–21. ISBN 0–947568–21–2

Abbildungsverzeichnis

Tabellenverzeichnis