UNIVERZITA PARDUBICE

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

DIPLOMOVÁ PRÁCE

2024 Bc. Pavel Klečanský

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Deduplikace dat a jejich využití

Bc. Pavel Klečanský

Diplomová práce
2024

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2023/2024

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Pavel Klečanský**
Osobní číslo:	**I22162**
Studijní program:	**N0613A140007 Informační technologie**
Téma práce:	**Deduplikace dat a jejich využití**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování

Cílem práce je podrobný popis algoritmů a problematiky deduplikace a spojování záznamů. Praktickým výstupem práce bude vytvoření knihovny, která umožní provádět spojování záznamů mezi dvěma zdroji dat a deduplikaci jednoho zdroje dat. V knihovně budou použity vybrané algoritmy popsané v teoretické části. V teoretické části bude popsán celý workflow deduplikace dat a spojování záznamů, a to od čištění dat až po klasifikaci. Práce také popíše a ukáže možnosti využití například Jaro-Winklerovy vzdálenosti, Levenshteinovy vzdálenosti a Damerau-Levenshteinovy vzdálenosti. Součástí práce bude i Jaccardův index a Q-gram podobnost.

Rozsah pracovní zprávy: **cca 60 stran**
Rozsah grafických prací: **–**
Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

CHRISTEN, Peter. *Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*. Heidelberg: Springer Berlin, 2012. ISBN 978-3-642-31164-2.
DEZA, Michel, Marie, DEZA, Elena. *Encyclopedia of Distances*, 4th ed.; Springer: Berlin, Germany, 2016. ISBN: 978-3-662-52844-0

Vedoucí diplomové práce: **Ing. Monika Borkovcová, Ph.D.**
Katedra informačních technologií

Datum zadání diplomové práce: **8. listopadu 2023**
Termín odevzdání diplomové práce: **17. května 2024**

L.S.

Ing. Zdeněk Němec, Ph.D. v.r.
děkan

prof. Ing. Antonín Kavička, Ph.D. v.r.
vedoucí katedry

V Pardubicích dne 30. listopadu 2023

Prohlašuji:

Práci s názvem Deduplikace dat a jejich využití jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 7/2019 Pravidla pro odevzdávání, zveřejňování a formální úpravu závěrečných prací, ve znění pozdějších dodatků, bude práce zveřejněna prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 17. 5. 2024

Bc. Pavel Klečanský

PODĚKOVÁNÍ

Rád bych poděkoval vedoucí práce, Ing. Monice Borkovcové Ph.D., za vedení a užitečné rady při zpracování diplomové práce. Dále bych rád poděkoval mé rodině, přítelkyni a přátelům za trpělivost a podporu po celou dobu studia.

ANOTACE

Diplomová práce se zabývá popisem problematiky deduplikace a spojování záznamu. Teoretická část zahrnuje celý proces deduplikace, od čištění dat až po klasifikaci. Práce také popisuje algoritmy, jako jsou Jaro-Winklerova vzdálenost, Levenshteinova vzdálenost, Damerau-Levenshteinova vzdálenost, Jaccardův index a podobnost Q-gramů. V praktické části práce je vytvořena knihovna**,** která umožňuje provádění spojování záznamů mezi dvěma zdroji dat a deduplikaci jednoho zdroje dat. Knihovna je implementována v jazyce Java a realizuje celý proces deduplikace a spojování záznamů pomocí vybraných algoritmů popsaných v teoretické části.

KLÍČOVÁ SLOVA

deduplikace, spojování záznamů, porovnávání dat, klasifikace, kvalita dat

TITLE

Data deduplication and usage options.

ANNOTATION

The thesis deals with the description of deduplication and record linkage. The theoretical part covers the whole workflow of deduplication, from data cleaning to classification. The thesis also describes algorithms such as Jaro-Winkler distance, Levenshtein distance, Damerau-Levenshtein distance, Jaccard index and Q-gram similarity. In the practical part of the thesis, a library is developed to perform record linkage between two data sources and deduplication of one data source. The library is implemented in Java and implements the entire workflow of deduplication and record linkage using the selected algorithms described in the theoretical part.

KEYWORDS

deduplication, record linkage, data matching, classification, data quality

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM ZDROJOVÝCH KÓDŮ

SEZNAM ZKRATEK

SQL	Structured query language
ISBN	International Standard Book Number
USA	United States of America
NYSIIS	New York State Identification and Intelligence System
CSV	Comma-separated values
XML	Extensible Markup Language
OWL	The Web Ontology Language
RDF	Resource Description Framework
FRIL	Fine-grained record linkage software
OYSTER	Open sYSTem Entity Resolution
FAMER	FAst Multi-source Entity Resolution system
SPARQL	SPARQL Protocol and RDF Query Language
LIMES	Link Discovery Framework for Metric Spaces
GUI	Graphical user interface
LTS	Long-term support
JDK	Java Development Kit
EM	Expectation–maximization
TIA	Total Information awareness
MATRIX	Multistate Anti-Terrorism Information Exchange System
EPC	Electronic Product Code

ÚVOD

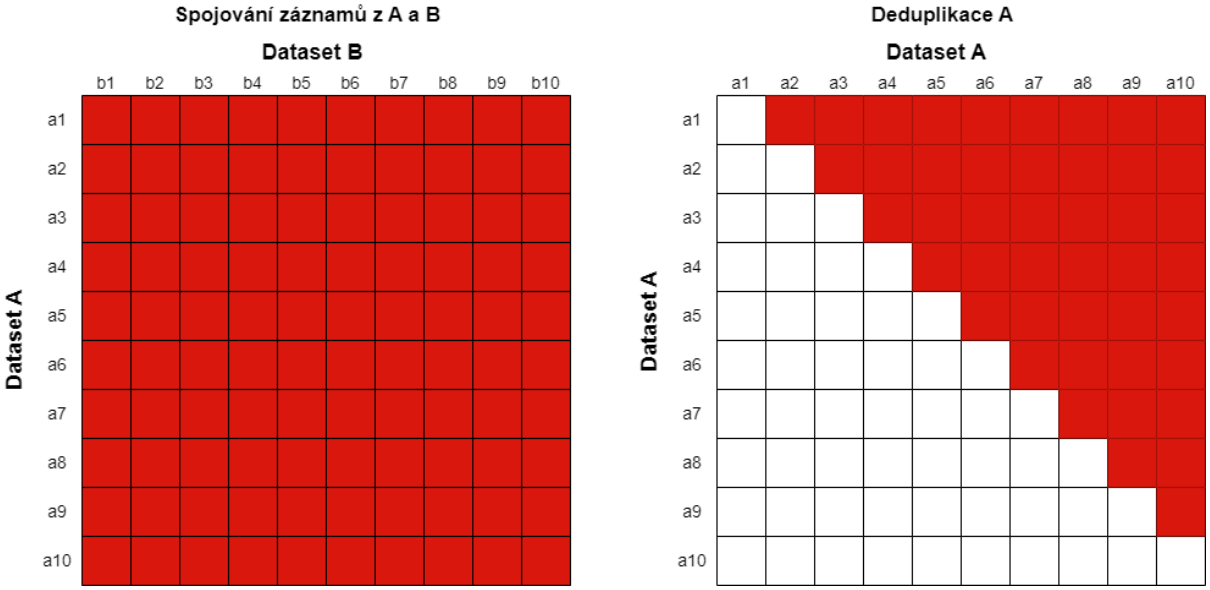
V posledních letech vzrostl zájem o inovativní techniky umožňující kvalitnější analýzu narůstajícího objemu dat, která shromažďují nejen podniky a vládní orgány, ale i jednotlivce. Hlavními cíli pro zavedení těchto technik je převážně zvýšení konkurenční výhody komerčních podniků, zlepšení produktivity vládních agentur a posílení národní bezpečnosti. Integrace a porovnávání dat různého původu je často zásadní v rozsáhlých projektech a přináší značné analytické výhody a spolehlivější závěry než jakých lze dosáhnout v rámci jedné databáze. V mnoha případech se však sloučení databází na základě jedinečných identifikátorů ukazuje jako neproveditelné, ať už z důvodu ochrany osobních údajů, nebo z důvodu neexistence takových identifikátorů. Proto je potřeba se obrátit k takovým metodám, které vycházejí ze statistiky, informatiky a strojového učení. Tyto metody jsou v obecné rovině označovány jako spojování záznamů nebo také deduplikace.

Diplomová práce se bude zabývat celým procesem spojeným s deduplikací a spojováním záznamů. Teoretická část práce se zaměřuje na koncepty deduplikace a spojování záznamů s důrazem na jednotlivé kroky celého procesu, a to včetně metod předzpracování dat, následného blokování a porovnávání záznamů a s tím spojených algoritmů pro klasifikaci záznamů. Kromě toho budou v práci vysvětleny algoritmy určené k řešení podobnosti řetězců, zahrnující mimo jiné Jaro-Winklerovu vzdálenost, Levenshteinovu vzdálenost, Damerau-Levenshteinovu vzdálenost, Jaccardův index a podobnost Q-gramů. Vedle podrobného popisu procesu se bude teorie zabývat využitím deduplikace a spojováním záznamů v různých oblastech. Následně bude provedena analýza stávajících řešení, a to i ve vztahu k praktickému výstupu.

Cílem praktické části této práce je vytvoření knihovny v jazyce Java s názvem Match4j, která umožní provádět kompletní proces deduplikace jednoho datového zdroje a spojování záznamů mezi dvěma datovými zdroji za využití vybraných algoritmů. Softwarové řešení bude navrženo a implementováno tak, aby umožňovalo práci s řetězci i čísly za využití algoritmů pro editační vzdálenost, založených na tokenech, a to i s využitím Q-gramové metody a specifických algoritmů pro porovnání číselných hodnot. S implementovanými algoritmy bude následně provedeno jejich porovnání, a to nad vybraným datovým setem. Tvorba knihovny Match4j bude v práci detailně zmíněna a zaměří se na popis implementace knihovny, včetně jednotlivých částí procesu deduplikace a spojování záznamů, které byly popsány v teoretické části práce.

1 DEDUPLIKACE A SPOJOVÁNÍ ZÁZNAMŮ

Spojování záznamů je technika propojení záznamů ze dvou nebo více datasetů do jediného záznamu, o kterém se předpokládá, že se vztahuje ke stejné entitě – například ke stejné osobě, rodině nebo firmě. S touto problematikou také úzce souvisí problematika deduplikace jednoho datasetu, deduplikaci lze chápat jako spojení jednoho datasetu se sebou samým. [1][2]



Obrázek 1: Porovnání spojování záznamů a deduplikace[1]

1.1 Problémy spojování záznamů a deduplikace

Úkol identifikovat a spojit záznamy, které se vztahují ke stejným entitám v rámci jedné nebo několika databází, je náročný z několika důvodů. Jedním z hlavních problémů spojování záznamů je absence jedinečných identifikátorů nebo klíčů v databázích, které mají být spojeny nebo deduplikovány. Příklady jedinečných identifikátorů záznamů jsou jedinečná čísla pacientů nebo daňových poplatníků nebo kódy spotřebitelských produktů. Pokud jsou takové identifikátory k dispozici ve všech databázích, které mají být spojovány, pak by se úloha spojování záznamů vyřešila jednoduchým databázovým spojením, který lze efektivně realizovat pomocí příkazů SQL. [1]

Dalším z problémů spojování záznamů je potřeba potenciálně porovnávat každý záznam v jedné databázi se všemi záznamy v druhé databázi, aby se zjistilo, zda dvojice záznamů odpovídá stejné entitě, či nikoli. Při deduplikaci jedné databáze je potenciálně třeba porovnat každý záznam se všemi ostatními. Výpočetní náročnost porovnávání dat proto roste kvadraticky s tím,

[1] vlastní zdroj

jak se zvětšuje počet záznamů v porovnávaných databázích. Naopak potenciální počet skutečných shod roste pouze lineárně s velikostí porovnávaných databází. [1]

V mnoha případech spojování záznamů není znám skutečný stav dvou záznamů, které jsou porovnávány napříč dvěma databázemi, tj. nejsou k dispozici základní pravdivé údaje nebo údaje "zlatého standardu", které by určovaly, zda dva záznamy odpovídají stejné entitě, či nikoli. Tím se liší od mnoha jiných aplikací pro dolování dat nebo strojové učení, kde jsou tréninková data snadno dostupná. Bez dalších informací (například kontaktování jednotlivců a dotazování se na správnost jejich osobních údajů) nelze mít jistotu, že výsledky projektu porovnávání dat jsou správné. To je problém zejména u rozsáhlých databází, které pokrývají velkou část populace, jako je tomu v případě zdravotnických nebo vládních sbírek dat. [1]

V posledních letech je čím dál tím více kladen důraz na ochranu osobních údajů, což přináší řadu problémů pro celou problematiku spojování záznamů. Především proto, že spojování záznamů často závisí na osobních údajích, jako jsou jména, adresy a data narození jednotlivců. Proto je potřeba pečlivě zohlednit ochranu soukromí zejména v případech, kdy jsou databáze spojovány mezi organizacemi nebo kdy mají být výsledky použity externí organizací nebo jednotlivci, například akademickými výzkumníky. Proto došlo k prudkému nárůstu výzkumu zaměřeného na vývoj technik, které umožňují spojování záznamů a zároveň chrání soukromí. Cílem takového výzkumu je umožnit porovnávání údajů mezi organizacemi, aniž by bylo ohroženo soukromí a důvěrnost porovnávaných údajů. [1]

1.2 Techniky deduplikace a spojování záznamů

1.2.1 Deterministické spojování záznamů a deduplikace

Deterministické techniky spojování záznamů a deduplikace lze použít, pokud jsou ve všech spojovaných datasetech k dispozici jedinečné identifikátory záznamu (nebo klíče), nebo pokud lze kombinací atributů vytvořit klíč spojení, který se použije k spojení záznamů se stejnou hodnotou klíče. Takové systémy spojování lze vyvinout na základě standardních dotazů SQL. Avšak dobrých výsledků spojení dosahují pouze tehdy, jsou-li identifikátory záznamů nebo klíče spojení ve vysoké kvalitě. To znamená, že musí být přesné, stabilní v čase a odolné vůči chybám. Optimálních výsledků spojování lze však dosáhnout pouze tehdy, pokud jsou identifikátory záznamů nebo klíče spojení vysoce kvalitní. To zahrnuje přesnost, stabilitu v čase a odolnost proti chybám. Alternativně lze ke klasifikaci dvojic záznamů použít sadu (často velmi složitých) pravidel. Tyto systémy založené na pravidlech sice nabízejí větší flexibilitu ve srovnání s použitím jednoduchého klíče pro spojování, ale jejich vývoj je náročný na lidskou

práci a je velmi závislý na spojovaných datasetech. Jednotlivci nebo týmy vytvářející taková pravidla potřebují komplexní znalosti nejen o datech, která mají být deduplikovány, ale také o samotném systému pravidel. V důsledku toho jsou deterministické systémy založené na pravidlech v praxi obvykle omezeny na ad hoc spojování menších datasetů. [3][4]

1.2.2 Pravděpodobnostní spojování záznamů a deduplikace

Protože jedinečné identifikátory záznamů jsou zřídkakdy k dispozici ve všech datasetech, které mají být spojeny, musí být proces spojování založen na existujících společných atributech. Ty obvykle zahrnují identifikátory osob, demografické informace a další specifické informace. Tyto atributy mohou obsahovat typografické chyby, mohou být kódovány odlišně, jejich části mohou být zastaralé nebo zaměněné nebo mohou chybět. Proto při provedení spojování záznamu dostane každá dvojice záznamu hodnocení, označující jejich podobnost, které se běžně označuje jako váha shody (matching weight). Pro zjištění, zda se jedná o skutečnou shodu, jsou stanoveny dva limity, dolní a horní, které rozdělují všechny potencionálně shodné záznamy do tří kategorii: shoda, potenciální shoda a neshoda. [3][4]

Kategorie potenciálních shod jsou ty dvojice záznamů, u nichž je k rozhodnutí o konečném stavu spojení nutný lidský dohled, známý také jako úřední kontrola (clerical review). Předpokládá se, že osoba provádějící tuto úřední kontrolu má k dispozici další údaje, které jí umožňují učinit informované rozhodnutí o stavu spojení. V praxi však často žádné další údaje k dispozici nejsou a úřední kontrola se stává procesem, v němž se při rozhodování uplatňují zkušenosti, zdravý rozum nebo lidská intuice. Což ve většině případů vede k tomu, že automatizované přístupy vedou ke spolehlivějším, konzistentnějším a nákladově efektivnějším výsledkům než plně manuálním spojování záznamů prováděné člověkem. [3][4]

V minulosti se zpravidla spojovaly pouze malé soubory dat a úřední kontrolu bylo možné zvládnout v rozumném čase. Při dnešních rozsáhlých datasetech s miliony záznamy se však tento proces stává nemožným. V těchto případech se i z malého procenta určeného pro úřední kontrolu stanou statisíce párů záznamů. Je zřejmé, že jsou zapotřebí přesnější a automatizované rozhodovací modely, které sníží nebo dokonce odstraní množství potřebných úřednických kontrol při zachování vysoké kvality propojení. [3]

1.2.3 Moderní přístupy ke spojování záznamů a deduplikaci

Tradiční pravděpodobnostní přístup k spojování záznamů a deduplikaci se ukázal být pro moderní aplikace v různých ohledech nedostatečný, což si vyžádalo řadu vylepšení. Mezi některá z vylepšení patří použití algoritmu maximalizace očekávání (EM) pro přesnější odhad

parametrů. Mezi další vylepšení patří použití přibližného porovnávání řetězců pro výpočet podobnosti hodnot nebo použití bayesovských sítí. V posledních letech se začaly využívat stále více rozšířené techniky strojového učení, dolování dat, a techniky získávání informací ke zlepšení procesu spojování záznamů. Mnohé z těchto přístupů jsou založeny na technikách učení s učitelem (supervised learning). Jedním z těchto přístupů jsou modely založené na technikách aktivního učení, které mohou být částečně nebo plně učeny s učitelem. Tyto modely fungují tím způsobem, že namísto použití předem určené sady trénovacích dat aktivně vyžádají označená data od odborníka. Cílem těchto technik je zajistit rovnováhu mezi automatizací a lidskou interakcí, i když je obtížné tuto rovnováhu stanovit. Vzhledem k širokému využití spojování záznamu v mnoha různých oborech může být obtížné předem vybrat dostatečně kvalitní tréninková data, díky tomu tyto modely dosahují dobrých výsledků v mnoha aplikacích spojování záznamů. [3][4]

Kromě technik aktivního učení jsou často používané i hybridní systémy, které v procesu spojování záznamů využívají technik učení s učitelem a učení bez učitele (unsupervised learning). Ukazuje se, že techniky strojového učení překonávají pravděpodobnostní techniky a poskytují nižší podíl potenciálních shod. Aby se překonali problémy nedostupnosti trénovacích dat v reálných datasetech, byly navrženy hybridní techniky. Tyto techniky zahrnují identifikaci dvojice záznamů jako shodné nebo neshodné pomocí neřízeného shlukování (unsupervised clustering) a následné použití výsledných dat jako trénovací množiny pro klasifikátor využívající technik učení s učitelem. [3]

1.3 Historie spojování záznamů a deduplikace

Spojování záznamů a deduplikace mají dlouhou historii. Ještě před vynálezem moderních počítačů se statistici a výzkumníci v oblasti veřejného zdraví zajímali o identifikaci záznamů, které patřily stejnému subjektu z jedné nebo několika různých databází. V roce 1946 poprvé použil Halbert L. Dunn termín spojování záznamů, aby popsal myšlenku sestavení knihy života pro každého jednotlivce na světě. Každá z těchto knih by začínala záznamem o narození a končila záznamem o úmrtí a mezi tím by se skládala ze záznamů o kontaktech jedince se zdravotním systémem a systémem sociálního zabezpečení a zahrnovala by také záznamy o sňatcích a rozvodech. Dunn si uvědomoval, že sestavení takovýchto knih života pro všech jedince v populaci by poskytlo cennou sbírku informací. Tyto údaje by mohly vládám umožnit zlepšit národní statistiky, optimalizovat plánování služeb a zdokonalit metody identifikace osob. [1]

V 50. a na počátku 60. let 20. století pak Howard Newcombe navrhl využití počítačů k automatizaci procesu spojování záznamů. Rozvinul také základní myšlenky úspěšného přístupu pravděpodobnostního spojování záznamů. Na základě Newcombeových myšlenek publikovali v roce 1969 dva statistici Ivan Fellegi a Alan Sunter svůj zásadní článek o pravděpodobnostním spojování záznamů. Jejich teorie prokázala, že optimální pravděpodobnostní rozhodovací pravidlo lze nalézt za předpokladu, že atributy použité při porovnávání záznamů jsou na sobě nezávislé. Tato práce se stala základem mnoha systémů a softwarových produktů pro spojování záznamů a dodnes se hojně využívá. [1]

Během několika posledních desetiletí byl základní pravděpodobnostní přístup různými způsoby rozšířen. Významnou práci provedl William Winkler a jeho kolegové z amerického úřadu pro sčítání lidu. Vyvinuli techniky, které umožňují zohlednit variabilitu řetězců pomocí funkcí porovnávání podobnosti řetězců. Vedle úsilí statistiků a výzkumných pracovníků v oblasti veřejného zdraví vyvinula databázová komunita techniky pro odhalování duplicitních záznamů v rámci jedné databáze a metody pro zvyšování kvality databáze v rámci procesu čištění dat. Duplicitní záznamy se běžně vyskytují v databázích, a proto je jejich identifikace a eliminace důležitým úkolem. Na rozdíl od pravděpodobnostního přístupu, který navrhli Fellegi a Sunter, se výzkumníci v oblasti databází rozhodli pro metody, jako je třídění databází na základě atributů pro spojení podobných záznamů a využití funkcí pro porovnávání řetězců k identifikaci přibližných podobností. [1]

V posledních letech došlo k výraznému nárůstu výzkumu v oblasti spojování záznamů, zejména v oblastech, jako je dolování dat, strojové učení a také v komunitách zabývajících se databázemi a datovými sklady. Vzhledem k tomu, že mnoho organizací shromažďuje rozsáhlejší databáze, stává se úkol identifikovat záznamy, které se vztahují ke stejným entitám v různorodých databázích, častějším než kdy dříve. [1]

1.4 Reálné využití deduplikace a spojování záznamů

1.4.1 Celostátní sčítání lidu

Národní agentury pro sčítání lidu po celém světě shromažďují údaje o různých aspektech populace, kultury, ekonomiky a životního prostředí ve své zemi. Tyto informace jsou pak prezentovány v nejrůznějších statistických zprávách, které vlády a podniky využívají při plánování přidělování finančních prostředků a zdrojů. Spojování záznamů bylo uznáno za důležitý nástroj statistiky sčítání lidu. Umožňuje opětovné využití stávajících zdrojů dat k sestavení nových datasetů statistických údajů, a tím snižuje náklady a úsilí potřebné

k provedení rozsáhlých sčítání lidu. Pomáhá také zlepšit kvalitu a integritu údajů, protože spojování záznamů z různých sčítání může pomoci odhalit a opravit rozporující nebo chybějící informace. [1]

Srovnávání dat lze také použít k vytváření datasetů dlouhodobých dat, a to tak, že se spojí údaje ze sčítání lidu, které byly shromážděny v různých časových okamžicích (například každých 5 nebo 10 let). Obecně se uznává, že dlouhodobá data jsou důležitým zdrojem informací o tom, jak se charakteristiky populace v průběhu času mění. Různé země mají různé zákony a předpisy, které upravují, jaký druh spojování záznamů lze provádět. Například v Austrálii musí být údaje o jménech a adresách shromážděné při národních sčítáních lidu zničeny do jednoho roku po jejich shromáždění (a to jak fyzické papírové sčítací formuláře, tak všechny elektronické verze těchto údajů). Taková omezení velmi komplikují vytváření dlouhodobých datasetů, protože spojování záznamu se opírat o informace, jako je věk, pohlaví, místo narození, náboženství, nejvyšší dosažené vzdělání a podobně. [1]

1.4.2 Zdravotnictví

Vedle celostátního sčítání lidu je zdravotnictví druhou oblastí, kde se spojování záznamů využívá již několik desetiletí. Podrobné informace shromážděné lékaři, nemocnicemi, zdravotními pojišťovnami a lékárnami poskytují v průběhu života člověka podrobný obraz o jeho zdravotním stavu. Tyto informace mají v populačním měřítku obrovskou hodnotu. Spojené zdravotní údaje umožňují opětovné využití stávajících zdrojů údajů pro nové studie, čímž se snižují náklady a úsilí při získávání údajů. Například spojení adres pacientů s prostorovými daty může vést k odhalení korelací mezi faktory životního prostředí a místními ohnisky případů onemocnění. [1]

První velká studie spojování záznamů ve zdravotnictví vznikla ve Spojeném království v 60. letech 20. století. Cílem této studie pod názvem Oxford Record Linkage Study, bylo vyvinout nové počítačové techniky spojování záznamů a aplikovat tyto techniky na údajích o narození, úmrtí a nemocničních údajích přibližně 350 000 osob. Tato studie umožnila studovat souvislosti mezi určitými nemocemi a pomocí dlouhodobých porovnatelných dat analyzovat úmrtnost z povolání, migraci a související socioekonomické faktory. Dalším velmi důležitým a pravděpodobně jedním z nejúspěšnějších praktických programů spojování zdravotních údajů na světě probíhal od poloviny 90. let v Západní Austrálii. Na základě postupu oddělování osobních identifikátorů nutných pro spojování záznamů od zdravotních informací potřebných pro výzkumné studie byly spojeny záznamy z různých zdravotních (i nezdravotních) zdrojů

a pro každou identifikovanou osobu byl vytvořen řetězec záznamů. V rámci tohoto programu bylo vytvořeno více než 700 výstupů. Některé z významných výsledků zahrnují zlepšení zdravotní politiky (např. pravidelné lékařské prohlídky pacientů s duševním onemocněním) a změny v klinické praxi (např. instalace defibrilátorů ve všech ambulancích a nemocničních odděleních nebo komunitní služby pro psychiatrické pacienty ohrožené sebevraždou). [1]

Podobné iniciativy v oblasti spojování zdravotních údajů zavedla řada dalších zemí. Nicméně citlivost osobních zdravotních záznamů a obavy o soukromí a důvěrnost však dosud omezovaly široké používání porovnávání údajů ve zdravotnictví v mnoha zemích. V důsledku toho se současné výzkumné úsilí zaměřuje na vytvoření technik, které umožňují porovnávání údajů a zároveň zachovávají soukromí dotčených údajů. [1]

1.4.3 Národní bezpečnost

Po teroristických útocích na USA v roce 2001 americká vláda (stejně jako vlády dalších západních zemí) výrazně zvýšila své úsilí a financování pokročilých analytických programů s cílem lépe odhalovat, a dokonce předcházet budoucím teroristickým činům. Ve srovnání s tradičními armádami je mnohem obtížnější teroristy identifikovat a sledovat, protože jsou volně organizováni v tajných sítích a jednotlivých buňkách, skrývají se, útočí zřídka a získávají finanční prostředky z různých zdrojů. Nicméně i teroristé zanechávají transakce a záznamy o své komunikaci a činnosti v online prostoru. Cílem protiteroristických iniciativ, jako jsou programy TIA (Total Information awareness) a MATRIX (Multistate Anti-Terrorism Information Exchange System), bylo uplatnit pokročilé techniky z oblastí, jako je spojování záznamů, dolování dat, biometrie, zpracování přirozeného jazyka a rozpoznávání obrazu, k odhalení neobvyklých vzorců v rámci různých databází a napříč nimi. Databáze, které měly být v těchto programech zpřístupněny, pocházejí jak z vládních agentur, tak ze soukromých organizací, jako jsou banky, letecké společnosti, letecké školy a autopůjčovny. [1]

Problémy spojené s použitím spojování záznamů v takových aplikacích jsou různorodé. Databáze, které je třeba porovnat, jsou velmi rozsáhlé a obsahují mnoho milionů záznamů a jsou také velmi rozdílné. V těchto databázích jsou uloženy různé údaje o jednotlivcích, které jsou pravděpodobně shromažďovány v různých časových okamžicích, a proto se údaje o adrese a jménu stejné osoby mohou lišit. Zločinci a teroristé také pravděpodobně používají falešné nebo upravené identity, aby skryli své aktivity, proto najít velmi malý počet potenciálních teroristů z populace čítající stovky milionů osob je podobné jako najít jehlu ve velmi velké kupce sena. Přesnost modelů pro spojování záznamů musí být extrémně vysoká, protože jinak

se počet (potenciálně falešných) pozitivních shod může stát nástrojem, který brání efektivnímu vyšetřování. [1]

1.4.4 Odhalování a prevence trestné činnosti a podvodů

Boj proti trestné činnosti a podvodům se dnes opírá o sofistikované informační systémy, které umožňují přesnou identifikaci osob podezřelých z trestného činu nebo podvodu. Nedílnou součástí těchto moderních informačních systémů pro boj s kriminalitou jsou techniky spojování záznamů. Zločinci při výslechu policisty běžně uvádějí pozměněné nebo dokonce smyšlené identifikační osobní údaje. Těmito klamavými identifikačními údaji mohou být například adresy známých, data narození zemřelých osob nebo falešná čísla sociálního pojištění či řidičských průkazů. Proto je hlavní výzvou při použití spojování záznamů v oblasti odhalování trestné činnosti a podvodů skutečnost, že na rozdíl od většiny jiných oblastí nedochází k odchylkám a chybám pouze v důsledku nepozornosti při zadávání údajů a měnící se povahy osobních údajů osob, ale proto, že jednotlivci záměrně upravují své údaje, jelikož nechtějí být identifikováni. Tyto záměrné změny jsou prováděny takovým způsobem, že upravené údaje vypadají skutečně (a případně odpovídají jiné osobě) a podobají se "nevinným" odchylkám nebo chybám, které mohly vzniknout náhodou. [1]

Techniky spojování záznamů mohou pomoci odhalit, zda tyto klamavé údaje o totožnosti odpovídají skutečné osobě, či nikoli. Aby byly techniky spojování záznamů v této oblasti použití efektivní, musí umožnit odhalení potenciálně shodných záznamů v (téměř) reálném čase, tj. během několika sekund. Podobnou aplikací, kde se stále častěji používá spojování záznamů, je ověřování totožnosti zaměřené na omezení trestných činů souvisejících s totožností. V mnoha zemích dochází k nárůstu trestných činů krádeže totožnosti, což má za následek miliardové ztráty finančních organizací a vážné sociální důsledky pro dotčené osoby. K trestným činům spojeným s identitou dochází, když podvodník získá přístup ke službám a výhodám pomocí falešné identity. Se stále rozšířenějším využíváním elektronických finančních transakcí a online služeb veřejné správy se stalo nezbytným ověřit identitu nepřítomných účastníků s vysokou mírou jistoty. Na základě statistické analýzy 300 milionů účtů bylo oznámeno, že přibližně 90 % úspěšně otevřených podvodných bankovních účtů v USA používá syntetickou identitu, přičemž téměř 75 % všech ztracených dolarů je způsobeno podvody se syntetickou identitou. [1]

Spojování záznamů je klíčovou součástí ověřování totožnosti, protože umožňuje porovnat identifikační údaje poskytnuté jednotlivcem s různými databázemi, které obsahují ověřené

a přesné záznamy o subjektech. Takové databáze mohou zahrnovat registrace voličů (známé také jako seznamy voličů), databáze řidičských průkazů a čísel sociálního pojištění či telefonní seznamy. Srovnání záznamů z těchto databází umožní vytvořit celkový obraz o posuzované osobě a vypočítat rizikové skóre, které poskytuje ukazatel pravděpodobnosti, zda se předložené identifikační údaje vztahují ke skutečné osobě, či nikoli. Podobně jako při spojování záznamů používaných v boji proti trestné činnosti je zapotřebí porovnávat v reálném čase proud jednotlivých záznamů dotazů v několika rozsáhlých databázích. [1]

1.4.5 E-commerce

Vzhledem k tomu, že spotřebitelé po celém světě stále častěji využívají k nakupování spotřebního zboží a služeb internet, staly se oblíbenými webové stránky, které poskytují srovnání cen. Tyto stránky umožňují spotřebitelům buď zadat dotaz na určitý výrobek, nebo procházet kategorie, typy a značky výrobků. Výzvou pro tyto srovnávací nákupní stránky je určit, které popisy výrobků v různých internetových obchodech odpovídají stejnému zboží. Přestože některé typy zboží mají jedinečné identifikátory, jako jsou čísla ISBN u knih, nebo systémy, jako je elektronický kód výrobku (EPC), i přesto se často popisy téhož výrobku v několika internetových obchodech značně liší. [1]

Aby bylo možné přesně a komplexně porovnat ceny jednotlivých spotřebitelských produktů, musí být srovnávací nákupní stránka schopna přesně určit, které popisy produktů se vztahují ke stejnému skutečnému produktu. V porovnání s jinými typy údajů, které se běžně používají pro porovnávání dat (jako jsou údaje o jménech a adresách osob), vyžadují rozdíly v popisech spotřebních výrobků odlišné míry podobnosti. [1]

2 PŘEDZPRACOVÁNÍ DAT

Proces předzpracování dat, někdy také nazývaný čištění dat, je technika, jejímž cílem je opravit, odstranit nebo změnit data na základě jejich hodnot. Předpokládá se, že nové hodnoty zlepší kvalitu dat a budou tak užitečnější v procesu spojování záznamů. [5]

2.1 Předzpracování dat u spojování záznamů a deduplikace

Předzpracování dat je často opomíjeným, ale důležitým krokem v procesu spojování záznamu a deduplikace. Důvodem, proč je předzpracování dat tak důležitým krokem, je skutečnost, že většina databází a datasetů obsahuje šum, nekonzistentní a chybějící data způsobená řadou faktorů. Výzkum naznačuje, že kvalitnější vstupní data vedou ke zlepšení kvality spojování záznamů. Například první studie pravděpodobnostního spojování v oblasti zdravotnictví ukázaly, že větší množství osobních identifikačních údajů výrazně zlepšilo přesnost výsledků spojování. Při absenci silně identifikačních osobních údajů bylo čištění dat shledáno jako jeden z klíčových způsobů, jak zlepšit kvalitu spojování. Odborníci často označují čištění dat za klíčový krok, který si může vyžádat až 75 % samotného úsilí při spojování záznamů. [1][5]

2.2 Problémy datové kvality spojené se spojováním záznamů a deduplikací

Pravděpodobně nejdůležitějšími měřítky kvality dat pro spojování záznamů a deduplikaci dat jsou přesnost a konzistence, jelikož velká část úsilí v krocích indexování, srovnávání a klasifikace se zabývá nepřesnými a nekonzistentními daty. Přesnost dat představuje úroveň, do jaké data odpovídají skutečnosti. Konzistence představuje, jak se shoduje formát a kódování jednotlivých hodnot atributů mezi jedním nebo více datasety. Pokud by data byla dokonalé kvality, pak by se spojování záznamů mohlo provádět pomocí jednoduchých operací spojování databází a nebylo by zapotřebí žádných sofistikovaných indexovacích technik nebo přibližných porovnávacích funkcí. Dokud jsou však data nedokonalé kvality, jsou zapotřebí techniky, které se dokážou vypořádat s nepřesnými a nekonzistentními daty a zároveň dosáhnout vysoké kvality porovnávání. [1][6]

2.3 Techniky čištění dat

Při spojování záznamů se můžou používat různé techniky čištění dat. Některé metody čištění se zaměřují na jednoduchou transformaci řetězců do specifické reprezentace, aniž by ve skutečnosti měnily význam. Další techniky se zaměřují na změnu informací v atributech záznamu, a to buď odstraněním neplatných hodnot, změnou hodnot, nebo přiřazením prázdných hodnot. Některé z těchto technik budou v dalším textu popsány. [5]

2.3.1 Přeformátování hodnot

Hodnoty dat lze jednoduše změnit na nový formát, aniž by se ztratila nějaká informace. Tím je zajištěno, že všechna data jsou ve společném standartu pro porovnávání při spojování. Například dva datasety, které uchovávají datumy v různém formátu (například 18.02.99 a 18. února 1999), je třeba pro porovnání změnit na společný formát. Touto transformací se nemění žádná data, pouze jejich reprezentace. Tato technika je nezbytná pro zajištění možnosti porovnávání shodných atributů záznamů. [5]

2.3.2 Odstranění interpunkce

Neobvyklé znaky a interpunkční znaménka se z textových řetězců obvykle odstraňují. Názvy s mezerami, pomlčkami nebo apostrofy mohou být s větší pravděpodobností zkresleny a jejich odstraněním lze odstranit případné rozdíly mezi těmito hodnotami. [5]

2.3.3 Odstranění alternativních chybějících hodnot a neinformativních hodnot

Datasety mohou často obsahovat speciálně kódované vstupní hodnoty, které jsou použity, pokud nejsou k dispozici žádné informace – například "9999" pro chybějící poštovní směrovací číslo. Různé datasety mohou obsahovat informace, které nejsou pro proces spojování užitečné jako například případy, kdy záznamy o přijetí do nemocnice obsahují "Dítě Viktorie" v atributu pro jméno nebo "NEZNÁMÁ ADRESA" v atributu pro adresu. Při tradičním pravděpodobnostním spojování, pokud jsou dvě proměnné se shodnými hodnotami obě označeny například jako "NEZNÁMÁ ADRESA", by tyto proměnné představovali shodu a přispívaly by pozitivně k celkovému skóre podobnosti záznamů, což by mohlo vést k nepřesným vyhodnocením. Spojování zahrnující chybějící nebo prázdnou hodnotu obvykle nezíská žádné pozitivní ani negativní skóre. Proto jsou tyto speciální hodnoty běžně odstraňovány. [5]

2.3.4 Fonetické kódování

Vytvořením kódování fonetické informace obsažené v textové proměnné (např. příjmení) je možné spojovat jména, která jsou zapsána odlišně, ale znějí stejně. Fonetické kódování je běžnou technikou při spojování záznamů. Mezi běžné kódovací algoritmy používané při spojování záznamů patří Soundex, NYSIIS a Metaphone. [5]

2.3.5 Standardizace názvu a adresy

Standardizace jména nebo analýza jména je proces rozkladu celého jména osoby na jednotlivé složky. Například pole jména se záznamem "Ing. Honza Petr Pavel" lze rozložit na titul, jméno, druhé jméno a příjmení a tyto složky lze jednotlivě porovnat. [5]

Podobně lze adresu rozdělit na části, jako je číslo ulice, název ulice a typ ulice. Vytvořením více proměnných tímto způsobem zapříčiní že malé rozdíly mezi záznamy, jako je například odlišné pořadí, mají menší vliv na spojování těchto záznamů. [5]

2.3.6 Vyhledávání přezdívek

Soubor přezdívek, který obsahuje běžné přezdívky a zdrobněliny jmen, lze použít k převodu předešlých jmen na běžnou hodnotu. Pomocí vyhledávání přezdívek lze osobě zaznamenané jako Pepa v jednom datasetu a Josef v jiném datasetu přiřadit stejné křestní jméno, což může tyto záznamy spojit. [5]

2.3.7 Přiřazení pohlaví

U záznamu s chybějící hodnotou pohlaví lze tuto hodnotu odhadnout na základě křestního jména. To vyžaduje vyhledávací tabulku, která přiřadí běžná křestní jména k pohlaví. [5]

2.3.8 Konzistence v záznamu

Záznamy obsahující nekonzistentní proměnné lze upravit a tuto nekonzistenci odstranit. Například záznam o adrese s hodnotu Bílé Předměstí Pardubice a poštovním směrovacím číslem 580 01 je nekonzistentní, protože se jedná o nesprávné poštovní směrovací číslo pro tuto část Pardubic. Problémem této techniky čištění ale je, že často není jasné, kterou proměnnou změnit, aby se tato nekonzistence odstranila. [5]

3 TECHNIKY SNÍŽENÍ POČTU SROVNÁNÍ

Naivní spojování datasetu s n_a záznamy s datasetem s n_b záznamy vyžaduje jako počáteční krok provedení $n_a n_b$ porovnání, kde deduplikace jednoho datasetu s n záznamy vyžaduje provedení $n(n-1)/2$ porovnání. I když by byla samotná porovnání relativně nenáročná na výpočet a dataset by měl střední velikost, může být tento krok výpočetně neúnosný. Z tohoto důvodu ve většině reálných aplikací spojování záznamů je nutné vyřadit velký počet dvojic záznamů, aniž by bylo provedeno úplné porovnání obou záznamů, což je proces známý jako indexování nebo někde také nazývaný jako blokování. Úložiště a další aspekty často vedou k dalšímu kroku a to filtrování, kdy jsou vyřazeny dvojice záznamů, u nichž je velmi nepravděpodobné, že by se jednalo o skutečné shody (true matches). [2]

3.1 Metody blokování a indexování

Za účelem zmenšení vyhledávacího prostoru bylo v poslední době navrženo mnoho blokovacích metod, z nichž některé budou v následující části uvedeny. Hlavní myšlenkou blokování je seskupení podobných záznamů, tzv. bloků nebo shluků, s využitím dostupných informací z těchto záznamů. [7]

3.1.1 Standardní blokování

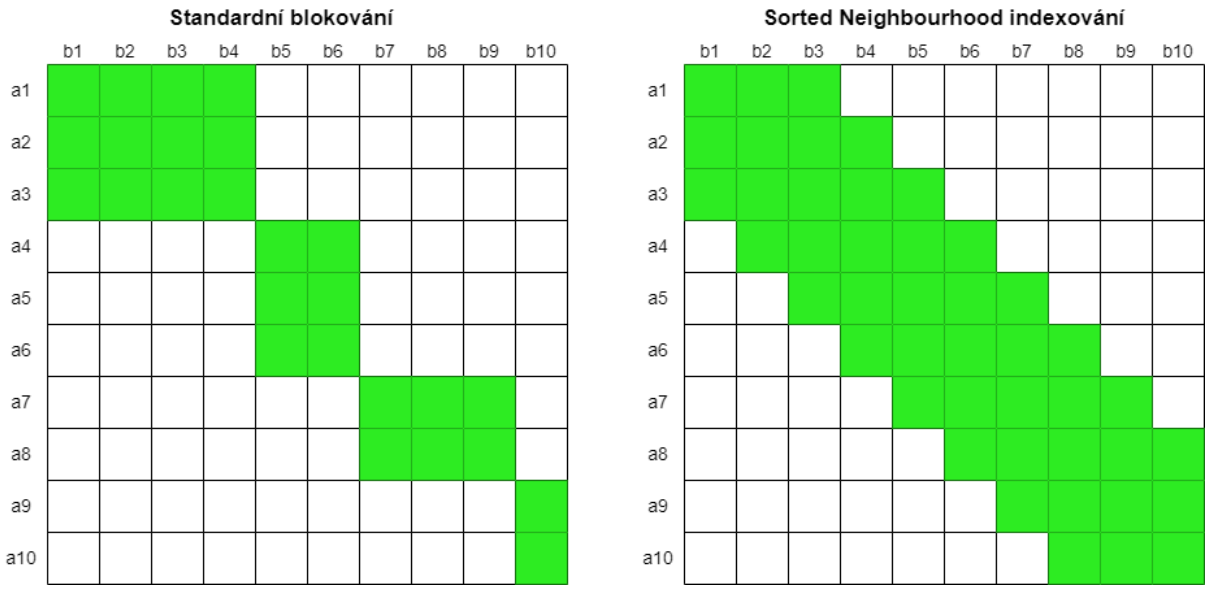
Metoda standardního blokování rozděluje záznamy do vzájemně se vylučujících bloků, přičemž záznamy v každém bloku sdílejí identickou hodnotu blokovacího klíče (blocking key). Blokovací klíč je hodnota složená z atributů záznamů v datasetu. Příkladem blokovacího klíče může být prvních pět znaků atributu příjmení. Blokovací klíč může být také složen z více než jednoho atributu, například atribut poštovního směrovacího čísla v kombinaci s atributem jména. Při výběru blokovacích klíčů je třeba zvážit kompromis mezi výhodami a výdaji. Dalším hlediskem při výběru blokovacích klíčů jsou chybové charakteristiky použitých atributů záznamů. Pro dosažení vysoké přesnosti spojení je vhodnější použít jako blokovací klíč atributy s nejmenší chybovostí. Jako způsob zmírnění vlivu chyb v blokačních klíčích se rovněž používá více blokačních klíčů. Další strategií, jak snížit vliv pravopisných a transkripčních chyb v atributech záznamů používaných jako blokační klíče (typicky atributy jména a adresy), je použití fonetických kódování těchto atributů, jako je Soundex nebo NYSIIS. [2][7]

Ve standardním blokování závisí počet vygenerovaných párů záznamů na počtu bloků a jejich velikosti. Za předpokladu, že se mají propojit dva datasety po n záznamech, výsledkem blokovacího klíče je b bloků a každý blok obsahuje n/b záznamů, je výsledný počet dvojic

záznamů $\frac{n^2}{b}$. Jedná se samozřejmě o ideální případ, který je u reálných dat těžko dosažitelný. Obecně proto platí, že počet dvojic záznamů je $\sum_{i=1}^{b} n_i^2$, kde n_i je počet záznamů v bloku i. Tudíž počet porovnávaných dvojic záznamů může být dominantní u velkých bloků. [7]

3.1.2 Sorted neighbourhood indexování

Sorted neighbourhood indexování, které představili Hernandez a Stolfo v polovině 90. let, je první alternativní technikou ke standardnímu blokování. Tato metoda funguje na principu, kdy namísto generování bloků podle blokového klíče jsou záznamy setříděny na základě třídicího klíče (sort key), a postupného posouvání okna pevné velikosti w (kde $w > 1$) nad setříděnými záznamy. Záznamy v tomto okně jsou poté navzájem spárovány a zařazeny do seznamu kandidátních dvojic záznamů. [1][7]



Obrázek 2: Porovnání standardního blokování a sorted neighbourhood indexování[2]

Kritéria používaná pro definici dobrého klíče třídění se liší od kritérií používaných pro definici dobrého klíče blokování. Zatímco blokovací klíč musí vyvažovat kvalitu generovaných dvojic kandidátních záznamů (kolik pravdivých shod je zahrnuto) s velikostí generovaných bloků (a tedy počtem výsledných dvojic kandidátních záznamů), definice třídicího klíče musí být konkrétnější a jemnější, aby třídění datasetu přiblížilo podobné záznamy k sobě. Důležitým faktorem při definování klíče pro třídění je skutečnost, že třídění datasetu je velmi citlivé na začátek klíče, zejména na první znak. Například dvě podobné hodnoty jmen "christina" a "kristina" si nebudou blízké, pokud je třídění založeno na jménech. Podobně jako u definic

[2] vlastní zdroj

více blokovacích klíčů, které se běžně používají pro standardní blokování, je výhodné provést několik iterací přístupu s použitím různých definic třídicích klíčů. [1][7]

Použití okna omezuje počet možných porovnání dvojic záznamů pro každý záznam na $2w - 1$. Výsledný počet porovnáních dvojic záznamů za předpokladu dvou datasetů po n záznamech je vypočítán metodou wn. Jeden z problémů metody nastává, pokud je počet záznamů v bloku větší než velikost okna. Například při třídicím klíči příjmení mohou mít stovky záznamů hodnotu "Novák", a pokud je velikost okna malá, nebudou porovnány všechny záznamy s hodnotou příjmení "Novák". Tento nedostatek je možné vyřešit dynamickou změnou velikosti okna w podle hodnot třídícího klíče. Velikost okna se zvětšuje, dokud jsou si klíče v setříděném atributu podobné podle funkce porovnávání řetězců. Okno pokryje sekvenci třídících klíčů (a jejich záznamů), které mají vzájemnou podobnost vyšší než určitý práh podobnosti. [1][7]

3.1.3 Problémy standardního blokování a sorted neighbourhood indexování

Jedním z hlavních problémů u těchto technik je zpracování chybějících hodnot, kdy standardní blokování, ani sorted neighbourhood indexování neposkytují žádné významné způsoby ošetření chybějících hodnot. Proto se při použití těchto metod v praxi chybějící hodnoty obvykle odstraňují vynecháním řádků nebo sloupců ze spojovaných datasetů. Ve skutečnosti jsou však chybějící hodnoty pouze chybějící. Skutečnost, že hodnota v určitém záznamu chybí, nepotvrzuje ani nepopírá spojení tohoto záznamu s jiným záznamem. Proto je často žádoucí povolit určité omezené párování chybějících hodnot. [8]

Dalším problémem zmíněných metod je netolerance pro neshody atributů. Záznamy, které se vztahují ke stejné entitě, často obsahují rozdíly v hodnotách atributu , které by je mohly z hlediska třídění výrazně odlišit. Například u záznamů uvedených v Tabulka 1 níže se záznamy týkají dvou osob Veroniky a Josefa, z nichž každá osoba má dvojici záznamů. V každé dvojici záznamů jsou tři atributy, která se buď přesně shodují, nebo mají vysokou shodu. V obou případech se však nejedná o stejná tři atributy. V situacích, jako je tato, by bylo žádoucí pravidlo pro výběr dvojice záznamů typu "libovolné tři hodnoty z pěti atributu se přibližně shodují". [8]

Tabulka 1: Ukázka tabulky uživatelů s duplicitními hodnotami[3]

Jméno	Příjmení	Adresa	Datum narození	Rodné číslo
Veronika	Nováková	Strukov 35	27.01.1992	925127/1245

[3] vlastní zdroj

Verča	Veselá	Strukov 35	27.01.1992	925127/2478
Josef	Kučera	Hostějov 54	18.06.1978	780618/1234
Josef	Kučero	Kutrovice 123	18.06.1978	780618/1234

Indexování na tento způsob lze konstruovat pomocí průniků a sjednocení standardních blokových indexů nebo sorted neighbourhood indexů. Ale počet jednotlivých indexů, které by bylo třeba vypočítat a spojit, by se rychle stal nepraktickým. V situacích, jako je tato, může být užitečný algoritmus, který tento typ porovnávání poskytuje snadněji. [8]

3.1.4 Q-gram indexování

U dat, která jsou znečištěná a obsahují velké množství chyb a variací, nemusí být standardní blokování ani sorted neighbourhood schopny vložit záznamy do stejných bloků, například pokud se začátek hodnoty třídicího klíče liší. Cílem indexování založeného na Q-gramech je překonat tuto nevýhodu generováním variant každého blokovacího klíče a použít tyto varianty jako konkrétní indexovací klíče pro standardní přístup založený na blokování. Každý záznam je vložen do několika bloků podle variant vygenerovaných z jeho klíče. Při indexování založeném na Q-gramech se každá hodnota blokujícího (nebo třídicího) klíče převede na seznam q-gramů určité minimální délky, která je určena uživatelem zvoleným limitem t ($t \leq 1$). Q-gram (někdy také n-gram) je podřetězec o délce q znaků. Obvyklé možnosti pro q jsou $q = 2$ (bigramy nebo digramy) nebo $q = 3$ (trigramy). Řetězec, který má c znaků, obsahuje $k = (c - q + 1)$ q-gramů s $l = \max(1, [k \times t])$ délkou nejkratšího podseznamu a celkovým počtem podseznamů s, vypočítaných pomocí vzorce $s = \sum_{i=l}^{k} \binom{k}{i}$. Seznam q-gramů řetězce se generuje pomocí metody klouzavého okna (sliding window), která z řetězce s vybírá q znaků na libovolné pozici od 1 do k řetězce. Například seznam bigramů, který je generován z řetězce "kateřina", je ["ka", "at", "te", "eř", "ři", "in", "na"]. [1][9][10]

Jednou z výhod indexování založeného na q-gramech je schopnost překonat chyby a nesrovnalosti v blokačních klíčích. Výsledkem je, že záznamy odpovídající skutečným shodám budou s větší pravděpodobností seskupeny ve stejném indexovém seznamu, i když jejich klíče vykazují odchylky. To vede k porovnávání více pravých shod než standardní blokování i techniky sorted neighbourhood indexování, a tím ke zlepšení kvality porovnávání. Nevýhodou však je, že se vygeneruje mnohem větší počet kandidátních dvojic záznamů, což vede k časově náročnějšímu kroku porovnávání. Z toho důvodu není q-gramové indexování doporučováno pro spojování nebo deduplikaci velkých datasetů. [1][9]

3.1.5 Canopy Clustering

Canopy shlukování je dvoukroková metoda shlukování pro více rozměrné datasety. Při použití pro indexování ji lze chápat jako shlukování záznamů v datasetech, které mají být porovnány nebo deduplikovány tak, že záznamy, jež jsou si navzájem podobné, jsou vloženy do stejného shluku. Nevýhodou většiny shlukovacích algoritmů je to, že jsou často velice výpočetně složité, což jde přímo proti zásadám indexování, které by mělo být výpočetně levné, aby bylo možné generovat dvojice kandidátních záznamů rychlým a škálovatelným způsobem. Z hlediska výpočetní rychlosti se Canopy shlukování liší od ostatních algoritmů primárně tím, že klíčová myšlenka tohoto přístupu zahrnuje použití levné, přibližné míry vzdálenosti mezi blokovými klíči k efektivnímu rozdělení dat do překrývajících se shluků, které se nazývají canopy. Každý shluk se pak stává blokem, z něhož jsou generovány dvojice kandidátních záznamů. [1][7]

Shlukování se pak provádí měřením přesných vzdáleností pouze mezi body, které se vyskytují ve společném canopy. V případě blokování se pro výpočet podobnosti mezi blokovými klíči a následným vytvořením skupin canopy používá buď Jaccard nebo TF/IDF (Term-Frequency / Inverse Document Frequency), kde se obě tyto funkce používají také pro přibližné porovnávání řetězců. Výsledkem vytvoření skupiny canopy je, že pouze záznamy ve stejné canopy se podrobně porovnají. Počet dvojic záznamů, které vrátí z canopy shlukování, je $\frac{fn^2}{c}$ kde c je počet skupin canopy a f je průměrný počet canopy, do kterých záznam patří. Parametry by měly být nastaveny tak, aby f bylo malé a c velké, aby se snížilo množství výpočtů. [1][7]

3.2 Meta-blokování

Blokování je typická metoda, která snižuje počet párových porovnání tím, že podobné entity rozdělí do bloků a provádí pouze porovnání v rámci každého bloku. Většina blokovacích metod, které zpracovávají data, používá redundanci, což znamená, že každý profil entity (entity profile) je umístěn ve více blocích. Profily entity jsou záznamy v datasetu, které potenciálně patří k nějaké entity. Většina metod blokování je ve skutečnosti pozitivně redundantní, čím více bloků dvě entity sdílejí, tím je pravděpodobnější, že se budou shodovat. Na druhou stranu redundance zahrnuje dva druhy zbytečných porovnání, a to nadbytečná porovnávání opakovaně stejného profilu entity ve více blocích, zatímco přebytečné (superfluous) porovnávají profily, které se neshodují. Z toho důvodu vznikly metody meta-blokování, jejichž cílem je restrukturalizovat kolekci bloků na novou, která má nižší počet porovnání, což zvyšuje efektivitu tím, že se překrývající bloky zbavují zbytečných porovnání. [11]

Meta-blokování je nezávislé na metodách blokování které byly použity, protože je dostatečně obecné na to, aby zvládlo jakoukoli kolekci bloků s pozitivní redundancí. Je důležité mít na paměti že meta-blokování nenahrazuje, ale doplňuje stávající blokovací metody. Stávající techniky meta-blokování fungují ve dvou fázích. Nejprve transformují kolekci vstupních bloků do grafu označovaného jako blokovací graf (blocking graph), kde profily entit slouží jako uzly a hrany spojují dva uzly, pokud se profily vyskytují společně alespoň v jednom bloku. Tento graf je navržen tak, aby eliminoval všechna nadbytečná porovnání tím, že každou dvojici společně se vyskytujících profilů spojuje jediná hrana, což znamená, že se porovnávají pouze jednou. Ve druhé fázi se používají techniky meta-blokování, které z grafu vyřazují nadbytečné porovnání. Za tímto účelem se každé hraně přiřadí váha na základě vlastností redundantních blokových kolekcí, které říkají, že podobnost mezi dvěma profily entit je úměrná jejich společnému výskytu v blocích. Vysoké váhy jsou přiřazeny shodným hranám (tj. hranám, které pravděpodobně spojují duplicity) a nižší váhy neshodným hranám. K odstranění hran s nízkými váhami, a tedy k vyřazení části nadbytečných porovnání, lze použít různé ořezávací algoritmy. Například jedna z takových strategií, nazývaná Weight Edge Pruning, vyřazuje všechny hrany, které mají nižší váhu, než je průměrná váha hrany v celém grafu. [12][13]

4 POROVNÁNÍ ATRIBUTŮ A ZÁZNAMŮ

Jak již bylo uvedeno v přechozích kapitolách, data používaná při spojování záznamů mohou být a často i jsou nekvalitní. Mohou obsahovat chyby a (typografické) odchylky, hodnoty jmen a adres se mohou v průběhu času měnit a pro mnoho osobních a jiných jmen může existovat několik platných tvarů. Ani sofistikované techniky čištění a standardizace dat nejsou vždy schopny vytvořit vysoce kvalitní data, která převedou hodnoty do přesně stejné podoby pro všechny atributy v páru záznamů, které se vztahují k stejné entitě. Namísto porovnávání hodnot atributů mezi dvěma záznamy pouze pomocí přesné shody je zásadní používat takové porovnávací funkce, které vracejí údaj o tom, jak jsou si dvě hodnoty atributů podobné. [1]

4.1 Přesné porovnání

Nejjednodušší porovnávací funkcí je funkce přesné shody dvou hodnot atributů s_1a s_2. Tato funkce počítá pouze přesnou podobnost. [1]

$$\mathrm{sim}_{\mathrm{exact}}(\mathrm{s}_1, \mathrm{s}_2) = \begin{cases} 1.0 & \text{if } s_1 = s_2, \\ 0.0 & \text{if } s_1 \neq s_2. \end{cases} \tag{4.1}$$

Pokud se jedná o řetězcové hodnoty, existují dvě rozšířené varianty funkce přesného shody. V první variantě se pro přesné porovnání bere v úvahu pouze začátek (nebo konec) dvou hodnot atributu. Pokud je prvních x znaků řetězcové hodnoty označeno pomocí $s[1 : x]$ a posledních y znaků pomocí $s[y : n]$, kde n je počet znaků v řetězci, pak lze definovat dvě porovnávací funkce takto: [1]

$$\mathrm{sim}_{\mathrm{truncate\_begin(x)}}(\mathrm{s}_1, \mathrm{s}_2) = \begin{cases} 1.0 & \text{if } s_1[1:x] = s_2[1:x], \\ 0.0 & \text{if } s_1[1:x] \neq s_2[1:x], \end{cases} \tag{4.2}$$

$$\mathrm{sim}_{\mathrm{truncate\_end(y)}}(\mathrm{s}_1, \mathrm{s}_2) = \begin{cases} 1.0 & \text{if } s_1[y:n] = s_2[y:n], \\ 0.0 & \text{if } s_1[y:n] \neq s_2[y:n]. \end{cases} \tag{4.3}$$

Druhou variantou přesného porovnávání řetězců je technika kdy, nejprve zakódujeme hodnoty řetězců s_1 a s_2 pomocí fonetické kódovací funkce. Tato kódování nahrazují původní hodnoty řetězců kódy tak, že podobně znějící řetězce jsou nahrazeny stejným kódem. Kódy vrácené z kódovací funkce jsou poté porovnány. [1]

$$\mathrm{sim}_{\mathrm{encode}}(\mathrm{s}_1, \mathrm{s}_2) = \begin{cases} 1.0 & \text{if } \mathrm{encode}(s_1) = \mathrm{encode}(s_2), \\ 0.0 & \text{if } \mathrm{encode}(s_1) \neq \mathrm{encode}(s_2). \end{cases} \tag{4.4}$$

4.2 Algoritmy editační vzdálenosti

Funkce pro porovnávání řetězců, které jsou založeny na konceptu editační vzdálenosti, fungují na principu počítání nejmenšího počtu editačních operací, které jsou nutné k převodu jednoho řetězce na jiný. Různé implementace tohoto konceptu umožňují provádět různé editační operace, včetně vkládání znaků, nahrazování znaků, mazání znaků nebo nahrazování znaků. Obecně je editační vzdálenost mezi dvěma řetězci s_1 a s_2 minimální cena transformace s_1 na s_2 pomocí zadané sady editačních operací s přiřazenými funkcemi. Náklady na transformaci pak odpovídají součtu nákladů na jednotlivé editační operace. Jedním z nejznámějších a nejzákladnějších algoritmů z této kategorie porovnání řetězců je Levenštejnova editační vzdálenost známá také jako základní editační vzdálenost. [1][14]

4.2.1 Levenshteinova editační vzdálenost a Damerau-Levenshteinova vzdálenost

Levenshteinova editační vzdálenost je definována jako nejmenší počet vložení, vymazání a záměn jednotlivých znaků, které jsou nutné k převodu jednoho řetězce na jiný. Jedná se o speciální případ editační vzdálenosti, protože používá jednotkovou váhu a tři základní editační operace. Pomocí algoritmu dynamického programování lze vzdálenost (počet úprav) mezi dvěma řetězci s_1 a s_2 vypočítat v čase $O(|s_1| \times |s_2|)$. Z dvojice řetězců se vytvoří matice d, která se používá k výpočtu editační vzdálenosti mezi řetězci. Buňka $d[i,j]$ v řádku i $(0 \leq i \leq |s_1|)$ a sloupci j $(0 \leq j \leq |s_2|)$ těchto matic odpovídá počtu úprav potřebných k převodu prvních i znaků řetězce s_1 (zobrazeného v prvním sloupci matice) na řetězec tvořený prvními j znaky řetězce s_2 (zobrazeného v horním řádku matice). [1][15]

		b	a	r	b	o	r	a
	0	1	2	3	4	5	6	7
b	1	0	1	2	3	4	5	6
r	2	1	2	1	2	3	4	5
a	3	2	1	2	3	4	5	4
m	4	3	2	3	4	5	6	5
b	5	4	3	4	3	4	5	6
o	6	5	4	5	4	3	4	5
r	7	6	5	4	5	4	3	4
a	8	7	6	5	6	5	4	3

Obrázek 3: Ukázka Levenshteinovy matice na slově barbora a brambora[4]

Algoritmus dynamického programování začíná vyplněním prvního řádku a prvního sloupce matice odpovídajícími hodnotami sloupce nebo řádku. Buňka $d[0, j]$ v řádku 0 a sloupci j ($0 \leq j \leq |s_1|$) je vyplněna hodnotou j a buňka $d[i, 0]$ v řádku i ($0 \leq \mathrm{i} \leq |s_2|$) a sloupci 0 je vyplněna hodnotou i. Zbývající buňky matice se vyplní pomocí následujícího rekurzivního postupu a to jestliže $s_1[i] = s_2[j]$, pak

$$\mathrm{d[i,j]} = \mathrm{d[i-1,j-1]}. \tag{4.5}$$

Pokud se ale $s_1[i] \neq s_2[j]$, pak

$$d[i,j] = \text{minimum} \begin{cases} \mathrm{d}[i-1,j]+1 & \text{vymazání,} \\ \mathrm{d}[i,j-1]+1 & \text{vložení,} \\ \mathrm{d}[i-1,j-1]+1 & \text{nahrazení.} \end{cases} \tag{4.6}$$

Levenshteinova editační vzdálenost mezi s_1 a s_2 je hodnota v pravé dolní buňce, $\mathrm{dist}_{\mathrm{levenshtein}}(s_1, s_2) = d[|s_1|, |s_2|]$. Vypočítanou vzdálenost lze také převést na podobnost pomocí vzorce:

$$\mathrm{sim}_{\mathrm{levenshtein}}(s_1, s_2) = 1.0 - \frac{\mathrm{dist}_{\mathrm{levenshtein}}(s_1, s_2)}{\max(|s_1|, |s_2|)} \tag{4.7}$$

Ačkoli se Levenshteinova vzdálenost v praxi hojně používá, nejedná se o vhodný algoritmus, pokud se liší celé segmenty řetězce, např. když je jeden řetězec prefixem druhého řetězce (Ing. Josef Hlava a Josef Hlava) nebo když řetězce používají zkratky (Honza FNovák vs. Honza Filip Novák). Tyto problémy jsou způsobeny především tím, že všechny editační operace mají stejnou váhu a každý znak je posuzován samostatně. Pro řešení tohoto problému byly navrženy další míry editační vzdálenosti, přičemž všechna tato nová řešení používají koncepty a algoritmy podobné těm, které se používají k výpočtu Levenshteinovy vzdálenosti. [1][14][15][16]

Jednou z těchto nových editačních vzdáleností je Damerau-Levenshteinova vzdálenost, někdy nazývaná chybová editační vzdálenost. Jedná se o rozšíření Levenshteinovovy editační vzdálenosti, kde jsou základní tři editační operace rozšířeny o čtvrtou transpoziční operaci, která umožňuje prohodit sousední znaky. Damerau-Levenshteinova vzdálenost má oproti základní editační vzdálenosti několik výhod. Tím, že zohledňuje transpozice, zvládá širší

[4] vlastní zdroj

rozsah chyb a variací v řetězcích, takže je vhodnější pro reálné případy. Kromě výhod má Damerauova-Levenshteinova vzdálenost také nevýhody, především to, že může být výpočetně náročná, zejména u dlouhých řetězců nebo velkých datasetů, z důvodu zvětšení prostoru pro vyhledávání v důsledku přidané operace transpozice. [15][16][17][18]

4.2.2 Jarova vzdálenost a Jaro-Winklerova vzdálenost

Tuto rodinu funkcí pro přibližné porovnávání řetězců vyvinuli Matthew Jaro a William Winkler z amerického úřadu pro sčítání lidu. Tyto funkce jsou navrženy speciálně pro porovnávání jmen. Základní srovnávací funkce Jaro kombinuje editační vzdálenost a techniky založené na q-gramech. Spočívá v počítání počtu znaků, které se vyskytují ve dvou řetězcích společně v určitém okně znaků, podobně jako u porovnávání na základě q-gramů. Konkrétně funkce Jaro zahrnuje výpočet počtu shodných znaků označených jako c, které se nacházejí v polovině délky delšího řetězce, a počtu transpozic označených jako t mezi těmito společnými znaky. Na základě těchto dvou čísel se pak vypočte hodnota podobnosti Jaro jako: [1][15]

$$\mathrm{sim}_{\mathrm{jaro}}(s_1, s_2) = \frac{1}{3}\left(\frac{c}{|s_1|} + \frac{c}{|s_2|} + \frac{c-t}{c}\right). \qquad (4.8)$$

Nad algoritmem Jaro byly postupně prováděny úpravy na základě zkušeností z projektů spojování záznamů provedených americkým úřadem pro sčítání lidu a empirických studií, které ukázaly, že na začátku jmen je méně chyb než uprostřed nebo na konci. Jedna z prvních úprav spočívá v zvýšení podobnosti mezi dvěma řetězci, pokud mají stejný začátek a rozdíly se vyskytují pouze uprostřed a na konci obou řetězců. Základní Winklerův algoritmus zvyšuje hodnotu podobnosti Jaro až pro čtyři shodné počáteční znaky. Základní Winklerova podobnost se pak vypočítá jako:

$$\mathrm{sim}_{\mathrm{winkler}}(s_1, s_2) = \mathrm{sim}_{\mathrm{jaro}}(s_1, s_2) + (1.0 - \mathrm{sim}_{\mathrm{jaro}}(s_1, s_2))\frac{p}{10}, \qquad (4.9)$$

přičemž p $(0 \leq p \leq 4)$ je počet shodných znaků na začátku dvou řetězců (společný prefix). [1][15]

4.3 Algoritmy založené na tokenech

Míry podobnosti založené na tokenech porovnávají dva řetězce tak, že je nejprve rozdělí na množinu tokenů pomocí tokenizační funkce, která se označuje jako tokenizer. Jednotlivé tokeny jsou podřetězce původního řetězce. Jednoduchý příklad může být, že tokenizační funkce rozdělí řetězec na tokeny na základě bílých znaků. Výsledkem řetězce "Harry Potter" je pak množina

tokenů [Harry, Potter]. Hlavní výhodou míry podobnosti založené na tokenech je, že podobnost je méně citlivá na záměny slov ve srovnání s mírami podobnosti založenými na editačních vzdálenostech. To znamená, že porovnání Harry Potter a Potter Harry představuje maximální podobnost, protože oba řetězce obsahují naprosto stejné tokeny. Na druhou stranu jsou penalizovány typografické chyby uvnitř tokenů, například podobnost Harry Poter a Hary Potter bude nulová. [14]

4.3.1 Q-gram

Metoda přibližného porovnávání řetězců na základě q-gramů spočívá v rozdělení dvou vstupních řetězců na krátké podřetězce o délce q znaků, nazývaných q-gramy, pomocí klouzavého okna. Poté se spočítá, kolik těchto q-gramů se vyskytuje v obou vstupních řetězcích. Počet q-gramů v řetězci s o délce $|s|$ je roven $c = |s| - q + 1$. Q-gramy jsou někdy také známé jako n-gramy. Nejčastěji volené hodnoty q pro atributy používané při porovnávání dat jsou $q = 2$ (bigramy nebo digramy) nebo $q = 3$ (trigramy). Míra podobnosti q-gramů mezi dvěma řetězci se stanoví tak, že se spočítá počet společných q-gramů, tedy q-gramů obsažených v obou řetězcích, a tento počet se vydělí buď počtem q-gramů v kratším řetězci (tzv. koeficient překrytí (overlap coefficient)), počtem v delším řetězci (tzv. Jaccardův koeficient (Jaccard coefficient)) nebo průměrným počtem q-gramů v obou řetězcích (tzv. Diceův koeficient (Dice coefficient)). Pokud c_{comon} označuje počet q-gramů společných pro s_1 a s_2, c_1 představuje počet q-gramů v řetězci s_1 a c_2 počet q-gramů v řetězci s_2, lze normalizovanou podobnost v rozsahu $0.0 \leq s \leq 1.0$ vypočítat pomocí následujícího postupu pro koeficient překrytí:

$$\mathrm{sim}_{\mathrm{overlap}}(s_1, s_2) = \frac{\mathrm{c}_{\mathrm{common}}}{\min(c_1, c_2)}, \tag{4.10}$$

jaccardův koeficient:

$$\mathrm{sim}_{\mathrm{jaccard}}(s_1, s_2) = \frac{\mathrm{c}_{\mathrm{common}}}{c_1 + c_2 - \mathrm{c}_{\mathrm{common}}}, \tag{4.11}$$

diceův koeficient:

$$\mathrm{sim}_{dice}(s_1, s_2) = \frac{2 \times \mathrm{c}_{\mathrm{common}}}{c_1 + c_2}. \tag{4.12}$$

Porovnávací funkce založená na q-gramech má z hlediska času i paměti složitost výpočtu $O(|s_1| + |s_2|)$. To je mnohem menší složitost ve srovnání s porovnávacími funkcemi

založenými na editační vzdálenosti, což činí porovnávání řetězců založené na q-gramech efektivnější, zejména pro delší řetězce. [1][15]

4.3.2 Jaccardův koeficient a jeho rozšíření

Základní Jaccardův koeficient někdy také označovaný jako Jaccardův index počítá podobnost mezi dvěma množinami položek, A a B, jako podíl počtu položek obsažených v průniku těchto dvou množin, který je dělený počtem položek obsažených ve sjednocení těchto dvou množin:

$$\text{sim}_{\text{jaccard}}(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \qquad (4.13)$$

Tento přístup se používá také ve funkci porovnávání řetězců na základě q-gramů popsané v přechozí části, kde tokeny v A a B jsou q-gramy extrahované z porovnávaných řetězců. [1]

Při porovnávání řetězců s_1 a s_2, které se skládají z více slov, umožňuje přesunutí důrazu z q-gramů na jednotlivá slova (tokeny) rozšíření základní Jaccardovy podobnosti. Toto rozšíření zahrnuje použití sekundární funkce podobnosti, sim', k posouzení podobnosti mezi každou dvojicí slov v porovnávaných řetězcích. Označíme-li A jako množinu tokenů extrahovaných z řetězce s_1 a B jako množinu tokenů extrahovaných z řetězce s_2, je sdílená množina tokenů S mezi oběma řetězci definována jako:

$$S = \{(a_i, b_j) \mid a_i \in A \wedge b_j \in B : \text{sim}'(a_i, b_j) \geq \theta\}, \qquad (4.14)$$

V tomto případě θ představuje práh podobnosti ($0 < \theta < 1$), který se používá k určení, zda jsou dva tokeny považovány za podobné. Tokeny, jejichž skóre podobnosti je vyšší než θ, jsou považovány za shodné, a jsou tedy zahrnuty do sdílené množiny tokenů S. Tokeny jedinečné pro řetězec s_1 tedy takové které nejsou v S jsou reprezentovány $U_A = \{a_i | a_i \in A \wedge b_j \in B \wedge (a_i, b_j) \notin S\}$ a ty, které jsou jedinečné pro řetězec s_2, jsou označeny $U_B = \{b_j | a_i \in A \wedge b_j \in B \wedge (a_i, b_j) \notin S\}$. S těmito definicemi lze rozšířenou Jaccardovu funkci podobnosti formulovat takto:

$$\text{sim}_{\text{jaccard\_ext}} = \frac{|S|}{|S| + |U_A| + |U_B|} \qquad (4.15)$$

Další rozšíření je možné tím, že jsou váhy přiřazeny jak sdíleným tokenům, tak jedinečným tokenům. [1]

4.4 Porovnání numerických hodnot

V určitých aplikacích se data používaná k porovnávání neomezují pouze na řetězcové hodnoty, ale zahrnují také atributy obsahující číselné informace. Typickými příklady jsou finanční údaje, jako jsou mzdy, úspory, výdaje nebo výše daní. Při porovnávání dvou číselných hodnot, označených jako n_1 a n_2, se v každém z dvojice kandidátních záznamů předpokládá, že jejich podobnost $\mathrm{sim}(n_1, n_2)$ bude rovna 1,0, pokud jsou obě hodnoty shodné ($n_1 = n_2$). Stejně jako u přístupů pro porovnávání řetězců je nutné umožnit výpočet přibližné podobnosti mezi číselnými hodnotami, aby bylo možné zahrnout odchylky a chyby v číselných údajích. Existují dva různé přístupy k dosažení tohoto cíle. V prvním přístupu je tolerován maximální absolutní rozdíl d_{max} mezi dvěma hodnotami n_1 a n_2, nezávisle na jejich skutečných hodnotách. Podobnost mezi n_1 a n_2 je pak vypočítána jako:

$$\mathrm{sim}_{\mathrm{num\_abs}} = \begin{cases} 1.0 - \left(\dfrac{|n_1 - n_2|}{d_{\max}}\right) & \text{když } |n_1 - n_2| < d_{\max}, \\ 0.0 & \text{ostatní.} \end{cases} \qquad (4.16)$$

Vypočtená přibližná hodnota podobnosti je nezávislá na porovnávaných číselných hodnotách. V určitých situacích by mohl být vhodnější relativní rozdíl než absolutní tolerovaný rozdíl. Takovou relativní číselnou hodnotu podobnosti lze vypočítat na základě procentuálního rozdílu pc mezi dvěma hodnotami, n_1 a n_2, jako

$$pc = \frac{|n_1 - n_2|}{\max(|n_1|, |n_2|)} \cdot 100. \qquad (4.17)$$

Podobně jako u absolutního číselného rozdílu d_{max} se pro výpočet přibližné hodnoty podobnosti používá maximální procentuální rozdíl pc_{max}:

$$\mathrm{sim}_{\mathrm{num\_perc}} = \begin{cases} 1.0 - \left(\dfrac{pc}{pc_{\max}}\right) & \text{když } pc < pc_{\max} \\ 0.0 & \text{ostatní.} \end{cases} \qquad (4.18)$$

Volba mezi těmito dvěma přístupy a přijatelnými hodnotami pro maximální absolutní nebo procentuální rozdíly závisí na konkrétním datasetu a na míře odchylky považované za přijatelnou pro konkrétní úlohu porovnávání dat. [1]

4.5 Porovnání časových hodnot

Vzhledem k tomu, že spojování záznamů často spočívá v porovnávání osobních údajů, má možnost porovnávat datumy, časy a hodnoty věku značný význam. Na datumy lze pohlížet jako

na specializovanou formu číselných údajů. Obvykle se při práci s datumy používá běžný přístup spočívající ve výpočtu rozdílu mezi dvěma datumy převedenými na dny a následným použitím metody porovnání absolutního rozdílu dnů založených na výše uvedeném numerickém přístupu. Ve srovnání s obecnými číselnými údaji však data představují zvláštní případy chyb, které se běžně vyskytují, a proto je nutné je zvážit. Jedním z takových scénářů je situace, kdy jsou v hodnotách datumů dvou záznamů zaměněny hodnoty dne a měsíce, zatímco hodnota roku zůstává nezměněna. Například 08.03.2024 a 03.08.2024 představují dva datumy, u nichž došlo k záměně hodnot dne a měsíce. Tento problém vzniká především v případě, že obě hodnoty dne a měsíce spadají do intervalu 1 až 12, což ztěžuje automatickou identifikaci chyb. [1]

Dalším přístupem k porovnávání hodnot datumů, jako jsou data narození, je jejich převod na hodnoty věku. Tyto hodnoty věku lze pak porovnávat pomocí předem stanoveného procentuálního rozdílu, jak bylo uvedeno výše. Výpočet hodnot věku vyžaduje referenci na pevné datum, kterým může být datum porovnání databáze nebo jakékoli jiné datum relevantní pro úlohu porovnávání. V ideálním případě jsou data převedena na standardizovanou jednotku, například počet dní nebo let od narození. Pokud d_1 a d_2 představují dvě porovnávané hodnoty datumů, lze procentuální rozdíl věku apc vypočítat následujícím způsobem:

$$\text{apc} = \frac{|d_1 - d_2|}{max(d_1, d_2)} \cdot 100 \tag{4.19}$$

Na základě tohoto procentuálního rozdílu věku se pak vypočítá procentuální podobnost věku:

$$\text{sim}_{\text{age\_perc}} = \begin{cases} 1.0 - \left(\frac{\text{apc}}{\text{apc}_{max}}\right) & \text{když apc} < \text{apc}_{max} \\ 0.0 & \text{ostatní,} \end{cases} \tag{4.20}$$

Existují také aplikace pro spojování záznamů, kde je třeba porovnávat časové hodnoty. Toho lze dosáhnout podobně jako u datumů buď přístupem založeným na tolerování absolutních časových rozdílů, nebo na procentuálních rozdílech vzhledem k určitému pevnému času. Časové rozdíly používané při výpočtech porovnávání lze počítat v hodinách, minutách nebo sekundách v závislosti na požadavku aplikace pro spojování záznamů. [1]

4.6 Porovnání zeměpisných hodnot

Geografické informace jsou stále častěji dostupné v mnoha aplikacích, například v podobě zeměpisných poloh (zeměpisná délka a šířka) pro adresy. Namísto použití pouze podobnosti

mezi komponentami adresy (tvořenými řetězci) je alternativním způsobem výpočtu podobnosti mezi dvěma adresami výpočet jejich zeměpisné vzdálenosti a použití této číselné hodnoty pro číselné porovnání. V rámci porovnávání adres podle jejich zeměpisné polohy je důležité poznamenat, že kvůli problémům s kvalitou dat často není možné získat přesnou polohu adresy. Pokud údaje o adrese chybí (například není uvedeno číslo ulice nebo jsou v databázi k dispozici pouze poštovní směrovací čísla) nebo obsahují chybné hodnoty, může být poloha přesná pouze na úrovni regionu, například ulice nebo předměstí. [1]

4.7 Implementace porovnávacích algoritmů v relačních databázích

Většina databázových aplikací používá dotazy k získání dat a prezentaci výsledků vyhledávání v tabulkovém formátu. K tomu používá dotazovacího jazyka SQL. Vyhledávání se obvykle opírá o sadu parametrů a příkazů, které umožňují provádět akce, jako je porovnávání dvou slov na rovnost nebo vyhledávání podřetězců v textu bez ohledu na jejich pozici. Vyhledávání v databázích však vykazuje omezení v případě zadání nesprávných hledaných výrazů, jako jsou chybně napsaná slova nebo vynechaná písmena. Kvůli tomu většina dnes používaných databází obsahuje různé algoritmy porovnávání hodnot, které se v databázových systémech souhrnně označují jako fuzzy vyhledávání nebo fuzzy porovnávání. [19]

Databáze Oracle poskytuje vestavěné funkce pro využití porovnávání řetězců pro různé úlohy. Ve verzi Oracle 10g Release 2 byl představen balík UTL_MATCH který byl ale poprvé zdokumentován (a tedy podporován) v Oracle 11g Release 2. Balík UTL_MATCH nabízí funkce, jako jsou EDIT_DISTANCE a JARO_WINKLER_DISTANCE. Funkce EDIT_DISTANCE, umožňuje efektivní porovnání dvou řetězců a výpočet jejich Levenshteinovy vzdálenosti. Podobně lze využít funkci JARO_WINKLER_DISTANCE k měření podobnosti mezi řetězci pomocí Jaro Winkler vzdálenosti. Kromě těchto funkcí představila Oracle databáze ve verzi 23c takzvané operátory kvality dat. Jedná se o operátory FUZZY_MATCH a PHONIC_ENCODE, které umožňují provádět fuzzy porovnávání řetězců. Operátor FUZZY_MATCH má podobné vlastnosti jako funkce v balíčku UTL_MATCH. Významný rozdíl však spočívá v metodách vyhodnocování, UTL_MATCH zkoumá bajt po bajtu, zatímco FUZZY_MATCH hodnotí znak po znaku. Kromě přechozího rozdílu FUZZY_MATCH nabízí také podporu širšího spektra algoritmů, včetně LEVENSHTEIN, DAMERAU_LEVENSHTEIN, JARO_WINKLER, BIGRAM, TRIGRAM, WHOLE_WORD_MATCH a LONGEST_COMMON_SUBSTRING. Operátor PHONIC_ENCODE převádí text na kódy specifické pro daný jazyk na základě výslovnosti textu. [20][21][22][23]

Kromě databáze Oracle poskytuje PostgreSQL také podporu pro různé algoritmy na porovnání řetězců. Pro využití těchto algoritmů je nutné přidat rozšíření fuzzystrmatch. Toto rozšíření bylo do PostgreSQL přidáno ve verzi 7.2 a od té doby prošlo řadou vylepšení. Mezi funkcemi v rámci tohoto rozšíření je funkce levenshtein, která počítá Levenshteinovu vzdálenost mezi dvěma řetězci. Dalšími důležitými funkcemi je dvojice funkcí, které pracují se Soundex systémem a jedná se o funkce soundex a difference. Funkce soundex převádí řetězec na odpovídající Soundex kód, zatímco funkce difference porovnává dva řetězce tak, že je převede na příslušné Soundex kódy a poté určí počet shodných pozic v kódu. Kromě systému Soundex podporuje rozšíření i Metaphone, který je stejně jako Soundex založen na myšlence vytvoření zástupného řetězce pro vstupní řetězec. Tyto řetězce jsou poté považovány za podobné, pokud vytvářejí stejné kódy. [24][25][26]

Ostatní databázové systémy jako například MySQL, Microsoft SQL Server nebo SQLite nemají tak rozsáhlou podporu jako Oracle nebo PostgreSQL. Všechny zmíněné systémy sice podporují práci se Soundex systémem, ale ani jeden neimplementuje Levenshteinovu vzdálenost nebo jiné podobné algoritmy pro porovnání řetězců.

4.8 Porovnání záznamů

U každé porovnávané dvojice potenciálně shodných záznamů se obvykle podrobně porovnává několik atributů. Pro každou dvojici záznamů se vytvoří takzvaný vektor číselných hodnot podobnosti, běžně nazývaný „srovnávací vektor" (comparison vector). Tyto srovnávací vektory jsou základem většiny klasifikačních technik pro spojování záznamů, kterým se bude věnovat další kapitola. [1]

5 KLASIFIKACE

Klasifikace kandidátních dvojic záznamů, které byly vytvořeny v kroku indexace a podrobně porovnány v kroku porovnávání, je založena především na hodnotách podobnosti ve srovnávacích vektorech těchto dvojic záznamů. Obecně platí, že čím jsou si dva záznamy podobnější, tím je pravděpodobnější, že se vztahují ke stejné entitě reálného světa. Klasifikační přístup může být buď neřízený (unsupervised), nebo řízený (supervised). Neřízený přístupy klasifikují dvojice nebo skupiny záznamů na základě podobností mezi nimi, aniž by měly dostupné jakékoli informace o skutečných vlastnostech shodných a neshodných dvojic záznamů. Oproti tomu řízené přístupy vyžadují trénovací data, kterým jsou známé informace o vlastnostech shodných a neshodných dvojic záznamů. Po kroku klasifikace je třeba v některých případech spojování záznamů nebo deduplikace dat sloučit záznamy, které byly spojeny, do nových záznamů. [1]

5.1 Klasifikace na základě limitů

Nejjednodušší způsob, jak zařadit dvojice kandidátských záznamů do dvou tříd shoda (matches) a neshoda (non-matches) a případně třetí třídy potenciální shoda (potential matches), je sečíst hodnoty podobnosti v jejich srovnávacích vektorech do jedné celkové hodnoty podobnosti (někdy nazvané SimSum) a poté použít limit podobnosti (nebo dva v případě, že se používají potenciální shody), aby se rozhodlo, do které třídy dvojice kandidátských záznamů patří. Při dvou třídách (shody a neshody) je pro klasifikaci dvojice záznamů (a, b) zapotřebí jediná klasifikační limita t:

$$\begin{aligned} SimSum[a,b] \geq t \Rightarrow [a,b] \rightarrow \text{Shoda}, \\ SimSum[a,b] < t \Rightarrow [a,b] \rightarrow \text{Neshoda}. \end{aligned} \tag{5.1}$$

Při třech třídách (shody, neshody a potenciální shody) jsou zapotřebí dva klasifikační limity, t_l (dolní) a t_u (horní), kdy dvojice záznamů (a, b) je klasifikována podle:

$$\begin{aligned} SimSum[a,b] \geq t_u \Rightarrow [a,b] \rightarrow \text{Shoda}, \\ t_l < SimSum[a,b] < t_u \Rightarrow [a,b] \rightarrow \text{Potenciální shoda}, \\ SimSum[a,b] \leq t_l \Rightarrow [a,b] \rightarrow \text{Neshoda}. \end{aligned} \tag{5.2}$$

Jednoduchá limitní klasifikace má dvě hlavní nevýhody. První spočívá v tom, že se všechny podobnosti atributů podílejí stejným způsobem na konečné součtové hodnotě podobnosti.

Důležitost různých atributů, stejně jako jejich vypovídací hodnota, pokud jde o rozlišení shod od neshod, se při takovém jednoduchém součtovém přístupu nebere v úvahu. Tuto nevýhodu lze překonat použitím vážených hodnot, kdy jsou různým atributům přiřazeny různé váhy podle jejich důležitosti nebo vypovídací hodnoty. Například hodnoty příjmení jsou ve srovnání s hodnotou pohlaví mnohem lepším ukazatelem toho, zda se dva záznamy vztahují ke stejným nebo různým osobám. Vážený součet se poté vypočítá tak, že se nejprve vynásobí hodnota podobnosti vypočtená pro určitý atribut váhovou hodnotou tohoto atributu před sečtením. Druhou nevýhodou sčítání podobností je, že se v kroku sčítání ztrácejí podrobné informace obsažené v jednotlivých hodnotách podobnosti (a to jak u neváženého, tak u váženého přístupu). [1]

5.2 Řízená klasifikace

Řízená klasifikace někdy také označovaná jako klasifikace s učitelem, je metoda klasifikace, která je postavená na technikách strojového učení. Metoda využívá trénovacích dat v podobě dvojic záznamů s jejich skutečným stavem shody (shoda nebo neshoda) k trénování klasifikačního modelu. Natrénovaný model se pak používá ke klasifikaci dvojic záznamů s neznámým stavem shody na shody a neshody. Pokud jsou porovnávané dvojice kandidátních záznamů klasifikovány pouze na shody a neshody (nikoli však na potenciální shody), pak je tato klasifikace známá jako binární klasifikační problém. V posledních desetiletích bylo v oblasti umělé inteligence, strojového učení a dolování dat vyvinuto mnoho technik binární klasifikace, z nichž některé byly aplikovány i na úlohy párování a deduplikace dat. [1]

5.2.1 Support vector machine

Klasifikační metoda známá pod zkratkou SVM nebo méně používaným českým názvem metoda podpůrných vektorů, je široce využívaná při spojování záznamů. Metoda vychází z myšlenky transformace trénovacích dat, které se skládají ze srovnávacích vektorů a štítků tříd označující shodu nebo neshodu, do vícerozměrného vektorového prostoru tak, aby byly trénovací záznamy ze dvou tříd odděleny a aby mezera mezi oběma třídami byla co největší. Rozhodovací hranice odpovídá nadrovině ve vysoce dimenzionálním prostoru a optimální rozhodovací hranice je ta, která má nejširší okraje vůči tréninkovým záznamům v obou třídách. Mapování z původního vstupního prostoru do vysoce dimenzionálního prostoru se provádí pomocí jádrové funkce, která umožňuje efektivní výpočet bodového součinu potřebného v procesu učení SVM. Postup učení zahrnuje řešení kvadratického optimalizačního problému, pro který existují účinné techniky. [1][27]

Pro spojování záznamů se používá řada technik založených na SVM. Jednou z nich je automatická klasifikační metoda, kterou vyvinul Peter Christen. Metoda funguje takovým způsobem, že v prvním kroku jsou z množiny všech srovnávacích vektorů vybrány trénovací data, které jednoznačně odpovídají shodám a neshodám. Jednoznačně shodné příklady jsou srovnávací vektory, kde jsou všechny hodnoty podobnosti rovny nebo velmi blízké přesné podobnosti 1, zatímco jasně neshodné příklady jsou srovnávací vektory, kde jsou všechny hodnoty podobnosti rovny nebo blízké 0. Pomocí této počáteční trénovací množiny je vycvičen první SVM. Následně jsou všechny srovnávací vektory, které nejsou v jedné ze dvou trénovacích množin, klasifikovány pomocí této počáteční SVM. V dalším kroku jsou srovnávací vektory, které byly klasifikovány jako nejvzdálenější od rozhodovací hranice SVM, začleněny do jedné ze dvou trénovacích množin podle toho, zda se nacházejí na straně shod nebo na straně neshod a na těchto rozšířených trénovacích množinách je trénován druhý SVM. Tento proces přidávání dalších srovnávacích vektorů do trénovacích množin a následného trénování nového SVM se opakuje, dokud není splněno kritérium zastavení.[1][28]

5.3 Neřízená klasifikace

Nevýhodou řízené klasifikace spojování záznamů je, že je závislá na existenci trénovací množiny. Taková trénovací množina není pro většinu reálných aplikací snadno dostupná. V metodách neřízené klasifikace pojem trénovací množiny neexistuje. Celá množina vzorů je zadána jako vstup pro algoritmus učení bez učitele, který předpovídá třídu každého neklasifikovaného vzoru nebo v případě spojování záznamů stav shody každé dvojice záznamů. Shlukování je běžně používaná metoda neřízené klasifikace, fungující na základním principu seskupování vzorů, které jsou si navzájem podobné. Jinými slovy, pokud je každý vzor reprezentován jako bod v prostoru, shlukovací algoritmy se snaží tyto body shlukovat do jednotlivých skupin v prostoru. [29]

5.3.1 K-means

Technika známá jako k-means shlukování má za cíl shlukovat body do k shluků. Algoritmus je častou volbou, pokud je znám počet tříd datových položek, a také kvůli jeho jednoduché implementaci a výpočetní efektivitě, zejména pokud je počet shluků k malý. Algoritmus se ukazuje dobrou volbou pro spojování záznamů a deduplikaci díky malému množství shluků, které se využívají. Použití tohoto algoritmu má dvě možnosti ohledně počtu shluků, a to buďto dva nebo tři shluky. Při použití dvou shluků jeden z nich představuje stav shody a druhý stav neshody, přidání třetího shluku nám umožní reprezentovat potenciální shodu. Tento algoritmus

pro neřízenou klasifikaci spojování záznamů a deduplikace funguje na způsobu, že považuje každý srovnávací vektor za bod v n-rozměrném prostoru, kde n je počet atributů v každém záznamu. Tyto body jsou poté shlukovány buďto do dvou shluků pro binární klasifikaci nebo do tří. Dobrých výsledků lze dosáhnout pomocí algoritmu shlukování k-means, pokud jsou všechny body rozloženy kolem k jasně oddělených shluků. [30][31]

6 FÚZE DAT

Dosud se předpokládalo, že proces spojování záznamů končí, jakmile jsou dvojice záznamů rozděleny na shodné a neshodné. V některých scénářích spojování a deduplikace dat však musí být párované záznamy také sloučeny, než je lze využít pro analýzu dat, dolování dat nebo další zpracování. Při slučování záznamů vzniká nutnost určit, které hodnoty atributů mají reprezentovat danou entitu. Pokud byly k určení shody použity techniky přibližného porovnávání nebo pokud je v páru přítomen další atribut, který však nebyl použit v procesu porovnávání, je třeba rozhodnout, která hodnota je nejvíce reprezentativní. Tyto výsledné hodnoty atributů se pak použijí k charakterizaci entity pro následné výpočty. Techniky usnadňující tento proces slučování jsou známy pod různými názvy, jako je kanonizace (canonicalization), fúze dat (data fusion), slučování dat (data merging) nebo normalizace záznamů (record normalization). [1][32][33]

6.1 Fúze dat a její využití

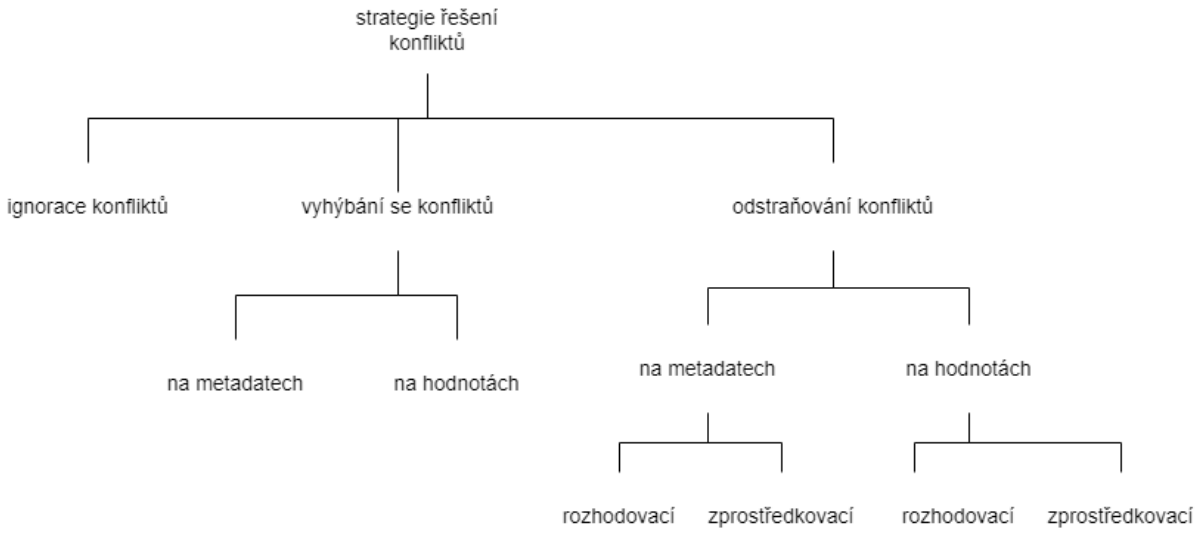
Různé techniky datové fúze jsou používaný i jiných oblastech než jen v souvislosti se spojováním záznamů. Jednou z těchto technik je fúze senzorů, známá také někdy jako fúze informací, která nachází uplatnění v různých oblastech výzkumu zahrnujících geografii, matematiku, medicínu a různá odvětví vojenského výzkumu. Základním principem, na němž je tato technika založena, je slučování dat z různých senzorů za účelem získání nových informací. V geografii se například používá pro překrývání satelitních snímků ze stejné oblasti, aby bylo možné rozeznat, zda je určitý kus země pokrytý lesem nebo není. Podobně v lékařské oblasti se fúze používá pro kombinaci několika snímků, aby se zlepšily možnosti detekce nemocí. [34][35]

Fúze dat se také uplatňuje v oblasti získávání informací (information retrieval), která se zabývá získáváním relevantních dokumentů z rozsáhlých sbírek. Běžným příkladem systémů pro vyhledávání informací jsou vyhledávače. Existuje velké množství vyhledávačů a tyto systémy se mohou značně lišit, například využíváním různých modelů, odlišným zacházením s různými aspekty dokumentu a podobně. Proto se výsledky jednotlivých vyhledávačů mohou dramaticky odlišovat v závislosti na tom, co hledáme. Jedním ze způsobů, jak zlepšit výsledky jednoho vyhledávače, je zadat stejný dotaz více vyhledávačům a výsledek integrovat. Tento problém lze řešit prostřednictvím meta vyhledávačů, jako je například www.metacrawler.com nebo www.metager.de. Výpočet jediného pořadí pro meta vyhledávač se vzhledem k různým

pořadím získaný z ostatních vyhledávačů používá fúze dat, někdy taky označovaná jako sloučení pořadí (rank merging). [34][35]

6.2 Řešení konfliktů a strategie pro řešení

Problém řešení konfliktů vzniká, když atributy vztahující se ke stejné entitě reálného světa mají rozdílné hodnoty v jednom nebo více zdrojích. Například zatímco zdroj 1 uvádí věk studenta "21", zdroj 2 ho uvádí jako "22". Tyto konflikty lze rozdělit do dvou tříd. První se označuje jako třída neurčitosti (uncertainties). Ta představuje situaci, ve které existují chybějící informace, například hodnoty atributů, které se vyskytují v jednom zdroji, ale chybí v jiném. Druhá třída rozporu (contradictions) zahrnuje konflikty, kdy je hodnota atributu v jednom zdroji v rozporu s hodnotou stejného atributu v jednom nebo více jiných zdrojích. [34]



Obrázek 4: Ukázka kategorii strategie řešení konfliktů fúze dat[5]

Řešení konfliktu lze chápat jako posloupnost rozhodnutí, která vedou ke konečnému závěru o správné hodnotě atributu. Při práci s nekonzistentními daty je třeba v prvním kroku určit, zda je třeba řešit existující konflikty dat. Pokud plánujeme odstranit datové konflikty, druhým rozhodnutím je obvykle rozhodnutí o konkrétních způsobech změny nebo interakce s daty, dokud nejsou konflikty vyřešeny. To často zahrnuje přímý výběr hodnoty atributu ze souboru možných hodnot. Strategie řešení konfliktů zahrnují přístupy používané k řešení konfliktních dat a poskytují rámec pro řešení takových situací. Tyto strategie uvádějí všeobecné postupy pro řešení konfliktních údajů, přičemž některé z nich určují přesné metodiky pro výběr konkrétní

[5] vlastní zdroj

hodnoty, sloučení hodnot nebo zavedení nové hodnoty. Popisují tak jednotnou, konzistentní reprezentaci, která vzniká při slučování dat. Tyto strategie lze rozdělit do tří základních kategorií, z nichž dvě mají několik podkategorií. Základní klasifikace strategií se rozděluje do tří kategorií podle přístupu k řešení konfliktů dat: ignorace konfliktů (conflict ignorance), vyhýbání se konfliktů (conflict ignorance) a odstraňování konfliktů (conflict resolution). [34][36]

6.2.1 Strategie ignorace konfliktů

Strategie ignorování konfliktů představuje kategorii strategií, které se zcela vyhýbají přímému řešení konfliktů dat. Při použití těchto strategií není nutné aktivně vyhledávat, identifikovat nebo ověřovat konflikty dat, protože tyto informace nejsou využity což může vést k nesouvislým výsledkům. Tyto strategie jsou použitelné vždy a ve všech situacích a lze je snadno implementovat. Mezi dva zástupce patří strategie „předej dál“ (pass it on) a „zvaž všechny možnosti“ (consider all possibilities). Strategie „předej dál“ jednoduše převezme všechny (případně konfliktní) reprezentace a hodnoty jejich atributů a předá je uživateli nebo jiné aplikaci, aniž by je změnila přičemž nechá uživatele nebo aplikaci rozhodnout, jak případné konflikty vyřešit. Při použití této strategie jsou v datech stále přítomny konflikty. Druhá zmíněná strategie funguje na principu, že dává uživateli na výběr ze všech možných kombinací hodnot atributů. Použití této strategie může zahrnovat vytváření kombinací hodnot atributů, které se ve zdrojových datech ještě nevyskytují. Při použití této strategie jsou v datech stále přítomny konflikty. [34][36]

6.2.2 Strategie vyhýbání se konfliktů

Strategie vyhýbání se konfliktů neřeší přímo jednotlivé konkrétní datové konflikty, ale přesto řeší nekonzistenci dat. Při rozhodování o způsobu řešení nekonzistence se nezohledňují konkrétní konfliktní hodnoty. Místo toho se tyto strategie rozhodují, zda nekonzistence vůbec řešit, a pokud ano, které atributové hodnoty v takových případech upřednostnit. Vzhledem k tomu, že při rozhodování nejsou zohledněny skutečné hodnoty, nejsou si tyto strategie vždy vědomy konfliktu. Z výpočetního hlediska jsou efektivnější než strategie řešení konfliktů, protože k rozhodnutí dospějí rychleji, aniž by se museli zabývat všemi konfliktními hodnotami. Obětují však přesnost, protože nezohledňují všechny dostupné informace, které by mohly být pro řešení konfliktu cenné. Tuto kategorii strategií lze dále rozdělit na dvě podkategorie a to jednu, která při rozhodování zohledňuje metadata nazývaná metadata based, a druhou, která je nezohledňuje, tzv. instance based. [34][36]

Dvě instance based strategie jsou „přijměte informace“ (take the information) a „žádné drby“ (no gossiping). První z nich zahrnuje shromažďování dostupných informací, přičemž ponechává stranou nulové hodnoty, což představuje přirozený přístup k řešení nejistoty. Druhá strategie pod názvem „žádné drby“ užívá principu, že pokud si nejsme jisti, jak s nekonzistencemi naložit, je lepší jí vynechat a informovat pouze o jistých faktech. Tento přístup zahrnuje do výsledku pouze konzistentní odpovědi a všechny nekonzistentní jsou ponechány stranou. Vzhledem k tomu, že rozhodnutí je založeno na datech a nekonzistentní odpovědi jsou ignorovány, je tato strategie strategií vyhýbání se konfliktům. Nejedná se, jak by se mohlo zdát, o strategii ignorující konflikty, protože konflikty správně identifikuje a je si jich vědoma. Příkladem strategie vyhýbání se konfliktům na základě metadat je „důvěřuj svým přátelům“ (trust your friends). Tato strategie je založena na důvěře v někoho jiného, a na tom, že poskytne správnou hodnotu. Rozhodnutí o tom, komu důvěřovat, je učiněno jednou a platí jednotně pro všechny hodnoty dat bez ohledu na přítomnost konfliktů. Příkladem této strategie je upřednostňování údajů z jednoho zdroje před údaji z jiných zdrojů. Určení preference zdroje je obvykle ponecháno na uvážení uživatele, ale může být také automaticky přiřazeno na základě kritérií, jako je nákladová efektivita, spolehlivost, velikost nebo jiné ukazatele kvality. [34]

6.2.3 Strategie odstraňování konfliktů

Oproti předchozím kategoriím berou strategie řešení konfliktů v úvahu data a metadata předtím, než se rozhodnou, jak konflikt vyřešit což je ve srovnání s jinými typy strategií výpočetně náročnější. Nabízejí však cestu k nejoptimálnějšímu řešení konfliktu. Na rozdíl od strategií ignorování a vyhýbání se konfliktům lze strategie řešení konfliktů dále rozdělit na rozhodovací (deciding) a zprostředkovací (mediating) strategie. Rozhodovací strategie se vyznačují výběrem hodnot z všech již přítomných hodnot. V závislosti na kategorii závisí výběr hodnoty pouze na hodnotách dat (instance based) nebo zohledňuje další informace, jako metadata (metadata based). Naopak zprostředkující strategie mohou zavést novou hodnotu, která nemusí nutně existovat mezi konfliktními hodnotami. V důsledku toho rozhodovací strategie obvykle umožňují snadný výpočet datové linie (data lineage). Ve všech případech je zřejmé, odkud hodnota pochází, a lze tedy dohledat její původ. U zprostředkující strategie informace o původu dat obvykle chybí, protože hodnoty mohou být vytvořeny libovolně a nemusí nutně odpovídat některé z existujících hodnot. [34][36]

Příklady instanced based rozhodovacích strategií jsou strategie „plakat s vlky“ (cry with the wolves) a „hod kostkou“ (roll the dice). První z nich pracuje s předpokladem, že správné hodnoty převažují nad nesprávnými. Tato strategie se shoduje s principem volby nejčastější

hodnoty mezi protichůdnými hodnotami. Ale při použití je nutné použít vhodné prostředky pro přerušení nerozhodného výsledku. Naproti tomu strategie „hod kostkou“ bere v úvahu všechny hodnoty a náhodně vybere jednu z nich. Ačkoli se může zdát, že se nejedná o příliš chytré řešení, jedná se o stále platnou strategii pro řešení konfliktů. Při nedostatku jakýchkoli dalších informací a vstupních údajů pro rozhodnutí o hodnotě je náhodná hodnota dostatečně dobrou volbou. Hlavní výhodou této strategie je její výpočetní efektivita. Příkladem instanced based zprostředkovací strategie je strategie „raději ohnout než zlomit“ (better bend than break) která se řídí principem kompromisu a neupřednostňuje jednu hodnotu před druhou. Místo toho se snaží vymyslet hodnotu, která je co nejblíže všem současným hodnotám. Mezi alternativní principy, které je možné použít patří například průměrování hodnot, zaokrouhlování nebo hledání nejmenšího společného jmenovatele mezi konfliktními hodnotami. [34]

Příklady metadata based rozhodovacích strategií jsou strategie „mějte aktuální informace“ (keep up to date), „blažená nevědomost“ (ignorance is bliss) a „ups, udělala jsem to zase“ (Oops, I did it again). První zmíněná strategie používá nejnovější hodnotu a vyžaduje některé další údaje o časovém razítku. Tyto informace mohou být obsaženy v tabulkách jako samostatný atribut nebo mohou být poskytnuty jiným způsobem. Druhá strategie pod názvem „blažená nevědomost“ je založená na neznalosti a vybírá mezi konfliktními hodnotami na základě toho, že některá hodnota je známá – na rozdíl od jiných neznámých hodnot. Je založena na metadatech, protože znalost hodnoty vychází z nějaké externí znalosti, např. experta, který má znalost hodnoty, nebo z nějakého seznamu či tabulky obsahující tuto hodnotu. Poslední ze zmíněných strategií využívá historických dat. Hodnota je vybrána, pokud se rozhodnutí o této hodnotě historicky osvědčilo neboli pokud byla vybrána již dříve. Realizace této strategie vyžaduje určitý druh paměti, seznam minulých rozhodnutí. Jedním z příkladů zprostředkovatelské strategie založené na metadatech je „raději ohnout než zlomit 2“ (better bend than break 2). Použijeme-li metadata k realizaci strategie „raději ohnout než zlomit“, dosáhneme zprostředkující strategie založené na metadatech. Příkladem může být výběr nejnižšího společného příznaku podle dané taxonomie. [34]

6.3 Výběr strategie

Výběr vhodné strategie řešení konfliktu pro konkrétní problém představuje náročný úkol, zejména pokud je k dispozici více alternativ. Proto toto rozhodnutí ve většině scénářů provádí odborníci. Výběr strategie závisí do značné míry na dané oblasti a úkolu a řídí se různými kritérii. Jedním z kritérií je dostupnost systému, kde posuzujeme, zda systém podporuje realizaci konkrétní strategie a jaké další strategie jsou dostupné. Dále se úvahy rozšiřují na

náklady na provedení, a kolik zdrojů jsme ochotni pro danou činnost vyčlenit. S náklady je spojená i kvalita dané fúze, neboť kvalita a cena na sobě závisí a kvalitní odpověď je nákladná. Kromě toho je také potřeba věnovat pozornost tomu, jestli máme všechny dostupné informace potřebné pro vybranou strategii. Výběr a implementace strategie představují složité procesy, které zůstávají převážně manuální, nejsou plně automatizované a spoléhají na odborné znalosti uživatelů. [34]

7 ANALÝZA STÁVAJÍCÍCH ŘEŠENÍ

7.1 Splink

Splink je open-source balíček v jazyce Python, který byl v roce 2020 navržen britským ministerstvem spravedlnosti (MoJ) za účelem usnadnění rozsáhlých projektů spojování záznamů. Důvodem k vytvoření projektu bylo primárně to, že Ministerstvo spravedlnosti a jeho agentury mají velkou řadu administrativních datových systémů. Tyto systémy byly vyvinuty v různých dobách pro různé účely a neexistuje jednotný identifikátor osoby, který by se používal ve všech systémech. To vede k problémům, když analytici a výzkumní pracovníci potřebují provést analýzu, která zahrnuje více systémů. [37][38]

Data z těchto systémů čítají celkově desítky milionů různých záznamů, z nichž každý se vztahuje k jednotlivci, ale chybí mu jednotný identifikátor. Proto při řešení tohoto problému dospěli k závěru, že vybraný nástroj musí být schopen rychle propojit až 100 milionů záznamů, zachovat maximální přesnost s ohledem na kvalitu dat, flexibilně pracovat se širokým spektrem vstupních dat a propojovat různé datasety, včetně těch, které vyžadují deduplikaci a spojování současně. Po provedení průzkumu existujících implementací spojování záznamu a deduplikace se rozhodli pro Fellegiho-Sunterův model, široce používaný statistický přístup s aplikacemi ve významných úlohách spojování záznamů, jako jsou úlohy prováděné americkým úřadem pro sčítání lidu a britským úřadem pro národní statistiku. [38][39]

Po výběru přístupu Fellegiho-Suntera se rozhodli k vytvoření softwarového balíčku. Tento balíček navazoval a rozšiřoval R implementaci algoritmu Expectation-Maximisatione pro odhad Fellegiho-Sunterova modelu vazeb FastLink a zahrnoval různá technická vylepšení, rozšířené funkce a možnosti přizpůsobení. Pro využití možností distribuovaných výpočtů bylo rozhodnuto, že bude možné provádět spojování záznamu a deduplikaci na Sparku což umožní rychlejší provádění srovnatelných úloh na podstatně větších datasetech. Druhá verze balíčku spoléhá především na Spark pro jeho vynikající výkon, což však nemuselo být optimální pro menší datasety. Avšak od třetí verze byla zavedena možnost počítat výpočty nad různými SQL backendy, jako je například DuckDB. Toto vylepšení rozšiřuje dostupnost Splinku pro širší uživatelskou základnu. Balíček Splink byl stažen více než 2 milionkrát a v svém GitHub repositáři, kde je umístěn zdrojový kód, získal přes tisíc hvězdiček. [38][39]

7.2 JedAI

Java gEneric DAta Integration (JedAI) je systém, který poskytuje všestranné funkce jako je sada nástrojů pro řešení spojování záznamů a nabízí tři základní způsoby využití. Za prvé slouží

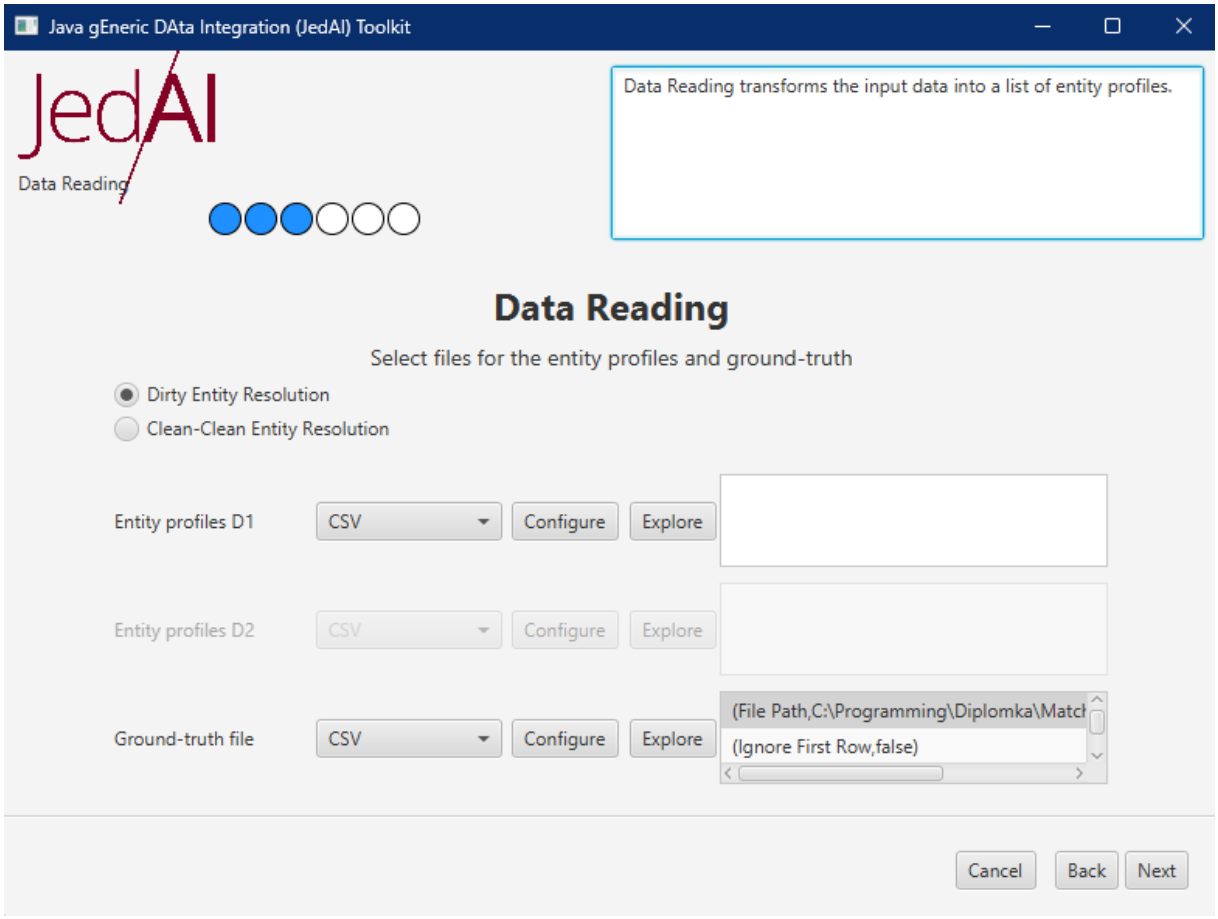
jako open source knihovna, která integruje moderní techniky do různých end-to-end procesů. Za druhé poskytuje desktopovou aplikaci s rozhraním, která umožňuje přístup k předem nakonfigurovaným řešením i neodborníkům. Zatřetí funguje jako komplexní pracovní prostředí, které usnadňuje porovnávání výkonnosti procesu ve strukturovaných a polostrukturovaných datasetech. [40][41]

Jádrem JedAI je moderní end-to-end postup pro spojování záznamů, který se jednotně aplikuje na strukturovaná a polostrukturovaná data. Tento proces je implementován pomocí JedAI backendu s názvem JedAI-core. Jedná se o open-source knihovnu, v níž je obsaženo několik moderních metod pro spojování záznamu, které se provádí v několika klíčových krocích: [40]

1. Prvním krokem je načítání data z disku do paměti, která podporuje různé formáty včetně CSV, XML, OWL, RDF a relačních databází.
2. Další částí je blokové sestavení, kde se podobné entity seskupují do bloků, aby se výrazně snížil prostor pro kandidáty na shodu a doba běhu spojování záznamů.
3. Následný krok čištění bloků dále zvyšuje časovou efektivitu odstraněním nadbytečných nebo zbytečných porovnání z překrývajících se bloků.
4. Čištění porovnání je dalším z klíčových kroků, pro snížení časové náročnosti. Čištění slouží stejnému účelu jako blokové čištění, ale pracuje na úrovni jednotlivých srovnání.
5. Jeden z nejdůležitějších kroků je krok porovnávání záznamů, který využívá několika metod k provádění porovnání v rámci finální sady bloků a vytváří graf podobnosti s jedním uzlem pro každý záznam a váženou hranou pro každou dvojici porovnávaných záznamů.
6. Předposledním krokem v celém spojování záznamů je shlukování záznamů, které rozděluje uzly v grafu podobnosti do shluků ekvivalence a zajišťuje, aby každý shluk obsahoval všechny záznamy odpovídající stejnému objektu reálného světa.
7. Nakonec se v rámci hodnocení posuzuje výkonnost identifikovaných shluků ekvivalence.

Jak již bylo zmíněno v předchozí části, JedAI je distribuováno s vlastním frontendem pod názvem JedAI-gui, jedná se o open-source desktopovou aplikaci s grafickým uživatelským rozhraním, která je vhodná jak pro odborné tak i laické uživatele. Je založena na průvodci, který umožňuje vytvářet proces pro spojování záznamů přímočarým způsobem, jednoduše výběrem z dostupných metod pro každý krok proces. Od uživatele není vyžadováno žádné ruční

dolaďování, protože každá metoda v JedAI-core zahrnuje neřízené funkce nezávislé na znalostech domény a je spojena s výchozí konfigurací parametrů, která trvale dosahuje vysokého výkonu – jedinou výjimkou je krok čtení dat, kde musí uživatel určit formát vstupních dat. [40]



Obrázek 5: Snímek aplikace JedAI Toolkit[6]

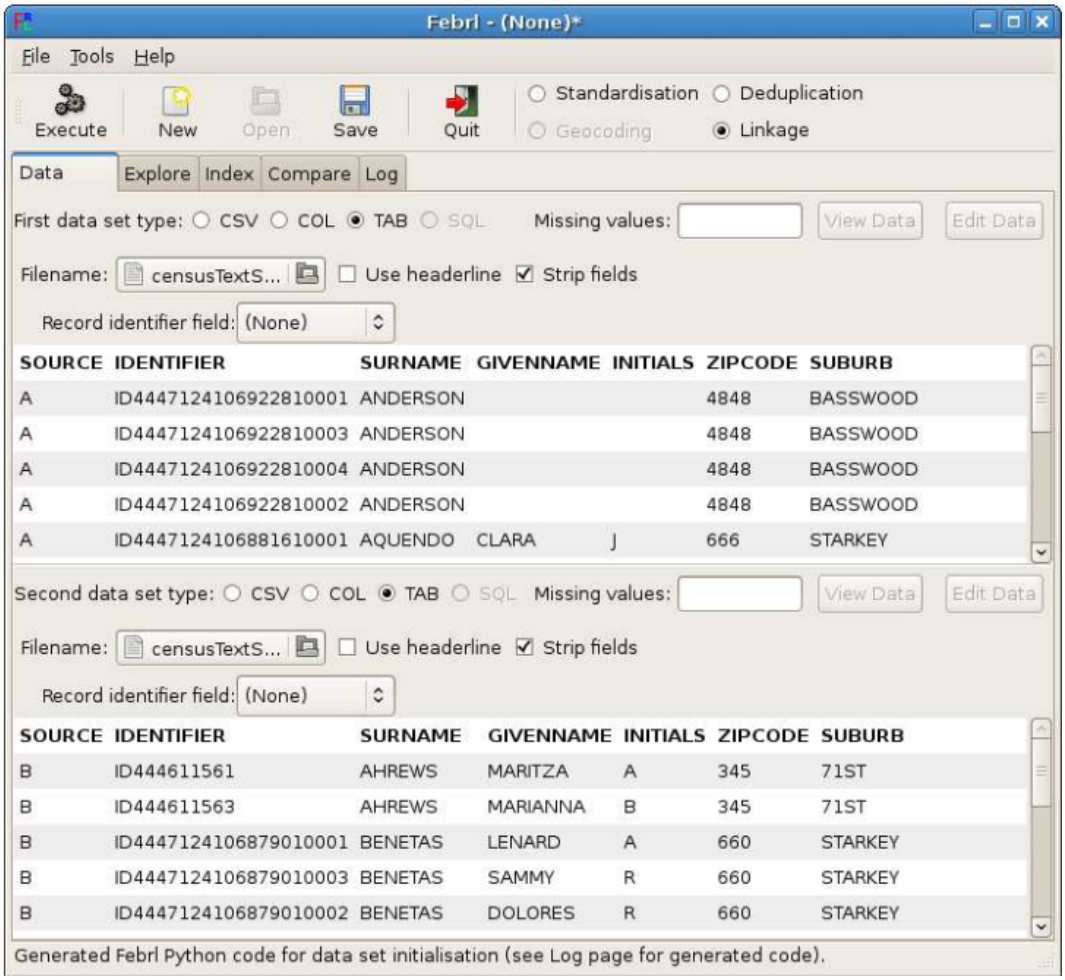
JedAI-gui může být využit i jako prostředek pro prozkoumání jednotlivých dostupných metod. To vede k vytvoření více než 4 000 různých kombinací, tj. pracovních postupů spojování záznamů, jejichž výkonnost lze snadno porovnat pomocí grafického uživatelského rozhraní. JedAI má na svém GitHub repositáři v současné době okolo dvě stě hvězdiček. [40]

7.3 Febrl

Software Febrl vznikl počátkem roku 2002 ve spolupráci Australské národní univerzity v Canberře a ministerstva zdravotnictví Nového Jižního Walesu v australském Sydney. Cílem tohoto společného výzkumného projektu bylo průkopnicky zavést inovativní metody pro

[6] vlastní zdroj

zlepšení čištění a standardizace dat, jakož i pro deduplikaci a spojování záznamů. Ačkoli se tento projekt primárně zaměřil na zdokonalení procesů souvisejících se zdravotnickými údaji, metodiky navržené a implementované v rámci projektu Febrl jsou univerzální, takže je lze použít v různých oblastech. Od svého prvního vydání na začátku září 2002 je software Febrl šířen pod licencí s otevřeným zdrojovým kódem podle vzoru Mozilla Public License 1.12. Tento licenční rámec nabízí uživatelům maximální flexibilitu a umožňuje jim bezproblémovou integraci Febrl s jinými softwarovými řešeními. [42][43]



Obrázek 6: Snímek aplikace Febrl[7]

Febrl je vytvořen v programovacím jazyce Python, který slouží jako optimální platforma pro rychlý vývoj prototypů. Od verze 0.4 obsahuje Febrl grafické uživatelské rozhraní, které je určeno uživatelům bez předchozí znalosti programování v jazyce Python. Toto grafické

[7] CHRISTEN, Peter. Development and user experiences of an open source data cleaning, deduplication and record linkage system. Online. ACM SIGKDD Explorations Newsletter. 2009, roč. 11, č. 1, s. 39-48. ISSN 1931-0145. Dostupné z: https://doi.org/10.1145/1656274.1656282.

rozhraní bylo vytvořeno s využitím knihovny PyGTK4 ve spojení se sadou nástrojů Glade. [42][43]

Od svého vzniku je software Febrl umístěn v repositáři s otevřeným zdrojovým kódem Sourceforge.Net. Poslední iterace, verze 0.4.2, byla vydána 14. ledna 2011. Vzhledem k datu vydání poslední verze a jejímu pokračujícímu spoléhání na Python 2 je zřejmé, že Febrl není aktuální. Přesto zůstává klíčovým nástrojem v oblasti spojování záznamů, o čemž svědčí jeho trvalé využívání, například v článku z roku 2022 nebo ve 181 citacích, které získal úvodní článek o tomto softwaru.

7.4 Ostatní stávající řešení

V předchozích kapitolách jsou popsány tři základní systémy spojování záznamů. Tuto problematiku však řeší řada dalších programů. Proto tato kapitola představuje stručný přehled dalších dostupných řešení. Je třeba poznamenat, že popisované systémy jsou výhradně open source programy. Stručný přehled klíčových open-source systémů spojování záznamů je uveden v Tabulka 2. U každé položky v tabulce je uvedeno, zda zahrnuje jednu nebo více metod ve procesu spojování záznamů, nebo zda podporuje paralelizaci, inkrementální spojování, grafické uživatelské rozhraní a také programovací jazyk, který je použit. Pro větší srozumitelnost jsou systémy rozděleny do tří skupin podle vstupních dat, a to systémy se strukturovanými daty, systémy s polostrukturovanými daty a hybridní systémy. [44]

Tabulka 2: Přehled klíčových open-source systémů spojování záznamů[8]

Název	Blokování	Čištění bloků	Porovnání	Shlukování	Paralelizace	Inkrementální spojování	GUI	Jazyky
Dedupe	✓	-	✓	-	multi-core	-	-	Python
DuDe	✓	-	✓	-	-	-	-	Java
FRIL	✓	-	✓	-	-	-	✓	Java
OYSTER	✓	-	✓	-	-	-	-	Java
RecordLinkage	✓	-	✓	-	-	-	-	R
Magellan	✓	-	✓	-	Apache Spark	-	✓	Python
FAMER	-	-	-	✓	Apache Flink	-	-	Java
Silk	✓	-	✓	-	Apache Spark	-	✓	Scala
LIMES	✓	-	✓	-	multi-core	-	✓	Java
Duke	✓	-	✓	-	-	✓	-	Java
KnoFuss	✓	-	✓	-	-	-	-	Java
SERIMI	✓	-	✓	-	-	-	-	Ruby
MinoanER	✓	✓	✓	-	Apache Spark	-	-	Java
JedAI	✓	✓	✓	✓	Apache Spark	-	✓	Java

Mezi nástroje pro strukturovaná data patří Dedupe, FRIL, OYSTER, RecordLinkage, DuDe, Magellan a FAMER. Všechny nabízejí alespoň jednu metodu pro blokování a porovnávání, přičemž neberou v úvahu shlukování. Jedinou výjimkou je FAMER, který se zaměřuje výhradně na shlukování a implementuje několik technik v Apache Flink. V této kategorii je většina systémů implementována v jazyce Java nebo Python a pouze dva z nich nabízejí grafické uživatelské rozhraní. Systémy určené pro polostrukturovaná data obvykle přijímají jako vstup soubory RDF nebo SPARQL koncové body. Nejvýznamnější z nich jsou Silk a LIMES, které jsou přizpůsobeny pro problém Link Discovery, který zahrnuje identifikaci vztahů mezi entitami. Oba systémy využívají vlastní techniky blokování ve spojení s různými funkcemi podobnosti. Silk i LIMES nabízejí grafické uživatelské rozhraní (GUI), které je odlišuje od jiných systémů, jako jsou SERIMI, Duke a KnoFuss. Posledně jmenované systémy používají převážně jednoduché techniky blokování hodnot a zaměřují se především na porovnávání. Hybridní nástroje MinoanER a JedAI se uplatňují shodně na strukturovaná i polostrukturovaná data. Systém JedAI byl ve vetší podrobnosti popsán v přechozí kapitole. Tyto programy ve skutečnosti implementují hlavní schéma agnostické techniky pro blokování, porovnávání a shlukování. A jsou to také jediné systémy, které nabízejí techniky čištění bloků. Celkově lze konstatovat, že všechny systémy se zaměřují primárně na porovnávání a nabízejí velké množství různých implementací algoritmů podobnosti řetězců pro porovnávání hodnot atributů. Ale pouze malé množství systémů umožnuje provedení obecného end-to-end procesu pro spojování záznamů. [44]

[8] CHRISTOPHIDES, Vassilis; EFTHYMIOU, Vasilis; PALPANAS, Themis; PAPADAKIS, George a STEFANIDIS, Kostas. An Overview of End-to-End Entity Resolution for Big Data. Online. ACM Computing Surveys. 2021, roč. 53, č. 6, s. 1-42. ISSN 0360-0300. Dostupné z: https://doi.org/10.1145/3418896.

Kromě zmíněných řešení spojování záznamů a deduplikace existují další nástroje, které nejsou speciálně uzpůsobeny pro tyto úlohy, ale jsou pro tyto účely využívány. Jedním z těchto nástrojů je Elasticsearch, distribuovaný nástroj pro fulltextové vyhledávání a analýzu založený na Apache Lucene. Ačkoli Elasticsearch nezahrnuje celý proces spojování záznamů, nabízí nástroje umožňující spojování záznamů, zejména prostřednictvím procesu textové analýzy a podpory fuzzy porovnávání. Analýza textu se provádí pomocí tzv. analyzátorů, které slouží k přípravě příchozích textových dat pro efektivní ukládání a vyhledávání pomocí tokenizace a normalizace textu. Tokenizace zahrnuje rozdělení textového obsahu na jednotlivá slova pomocí různých kritérií, zatímco normalizace transformuje tokeny do standardizované podoby. Elasticsearch poskytuje řadu hotových analyzátorů, které je možné použít ve fázi analýzy textu. Fuzzy porovnávání umožňuje vyhledávat dokumenty obsahující výrazy podobné dotazovanému výrazu. Kromě nástrojů pro spojování záznamů, které jsou k dispozici přímo v Elasticsearch, existují navíc rozšíření, jejichž cílem je umožnit komplexní spojování záznamů v prostředí Elasticsearch. Jedním z takových nástrojů je rozšíření zentity. [45][46]

7.5 Hlavní nedostatky současných řešení

Současná řešení spojování záznamů se potýkají s několika problémy, které mohou být jedním z důvodů bránících širšímu praktickému využití. Jedním z nedostatků současných systémů je, že při spojování záznamů uživatelé často potřebují řadou komplexních kroky, jako je blokování, porovnávání, klasifikace, čištění a další, ale současné systémy obvykle nabízejí podporu pouze pro část těchto kroků a opomíjejí stejně důležité, avšak méně známé postupy. Další problém spočívá v tom, že spojování záznamů vyžaduje různorodé nástroje zahrnující dolování dat, vizualizaci, čištění dat, dotazování pomocí SQL, vyhledávání podle klíčových slov a další. Současné systémy spojování záznamů neposkytují dostatečnou podporu pro tyto techniky a neexistuje jednoduchý způsob, jak toho dosáhnout. Integrace všech těchto technik do jediného systému je velmi obtížná. Naopak alternativní řešení spočívající v pouhém přesouvání dat mezi současným systémem spojování záznamů a systémy, které provádějí čištění, vizualizaci atd. je rovněž těžkopádné a časově náročné. Hlavním důvodem pro tento problém je, že většina současných systémů jsou samostatné monolity, které nejsou od základu navrženy tak, aby dobře spolupracovaly s jinými systémy. Další a taky jeden z nejpalčivějších problémů, se který trápí současné systémy spojování záznamů, je jejich složitost používaní. V mnoha scénářích spojování záznamů se uživatelé často ocitají v nejistotě, protože si nejsou jisti, jak proces zahájit nebo určit následné kroky. Zdánlivě jednoduché úkoly, jako je základní provedení deduplikace dat, se mohou ukázat jako pozoruhodně složité. [47]

8 MATCH4J

Hlavním cílem praktické části je vyvinout knihovnu v jazyce Java, která podporuje celý proces spojování záznamů a deduplikace dat, od čištění dat až po klasifikaci. Kromě již zmíněných funkcí byla knihovna navržena tak, aby podporovala načítání CSV datasetů a poskytovala základní podporu fúze dat. Jak bylo uvedeno v předchozí kapitole, jedním z nejpalčivějších problémů, kterým čelí současné systémy spojování záznamů a deduplikace je jejich složitost používání. Proto bylo hlavním cílem, aby knihovna byla snadno použitelná jak pro uživatele, kteří mají rozsáhlou zkušenost se systémy spojování záznamů nebo deduplikace, tak i pro uživatele kteří se nikdy s podobnými systémy nesetkali. Z tohoto důvodu byl při tvorbě knihovny kladen důraz na jednoduchost používání a s tím spojenou snahu minimalizovat množství informací které uživatel potřebuje, aby mohl knihovnu používat. Jedním z přístupů, jak toho dosáhnout, bylo přijetí paradigmatu konvence nad konfigurací. Toto paradigma automatizuje určité konfigurace ve výchozím nastavení, čímž uživatele zbavuje nutnosti manuálního nastavení. Zároveň však umožňuje uživatelům, kteří chtějí některé nastavení nakonfigurovat sami, bezproblémové upravení podle potřeby. Ilustrativním příkladem konvence oproti konfiguraci může být textový atribut záznamu, kterému je automaticky přiřazen výchozí porovnávací algoritmus. Současně však mají uživatelé možnost specifikovat jiný algoritmus podle svých potřeb. Pokud tedy už nechce, nemusí složitě vybírat z řady dostupných algoritmů při každém použití, pokud však chce má tu možnost. Tento přístup snižuje bariéry pro méně zkušené uživatele, ale zároveň poskytuje flexibilitu těm, kteří mají specifické preference.

8.1 Použité technologie

V této části budou představeny technologie použité v praktické části diplomové práce. Zvoleným programovacím jazykem pro implementaci byla Java, konkrétně nejnovější verze LTS (Long-Term Support), Java 21. Při tvorbě knihovny bylo cílem minimalizovat závislost na externích knihovnách a místo toho upřednostnit využití standardní knihovny, která představuje kolekci tříd, rozhraní a metod, které jsou součástí JDK. V důsledku toho byly v implementační části knihovny použity pouze tři externí knihovny, a to jmenovitě opencsv, logback-classic a smile-core. Logback slouží jako logovací knihovna, využívající SLF4J jako fasádu pro logování, čímž umožňuje oddělení logovacího kódu od jeho zpracování a implementace ukládání. Opencsv nabízí podporu pro parsování souborů CSV, zatímco Smile (Statistical Machine Intelligence and Learning Engine) poskytuje širokou škálu klasifikačních

algoritmů a algoritmů strojového učení. Pro účely testování byly použity knihovny JUnit5 a AssertJ. Samotný vývoj knihovny probíhal v integrovaném vývojovém prostředí IntelliJ IDEA Ultimate od firmy JetBrains a jako nástroj pro sestavení byl využit Gradle. Mimo sestavování byl Gradle použit i pro formátování projektu za pomocí pluginu spotless který dokáže automaticky formátovat kód v závislosti na konfiguraci. V průběhu celého procesu vývoje byly také využívány GitHub Actions, které zajišťovaly, že nově zavedený kód prochází nad všemi testy a zachovává předepsané formátování. [48][49]

8.2 Reprezentace dat a načítání strukturovaných dat do aplikace

Aby bylo možné provádět spojování záznamu nebo deduplikaci, musí mít data strukturu které knihovna rozumí. Knihovna pracuje s třemi základními třídami Table, Record a Element. Třída Table představuje tabulku obsahující pole záznamů, z nichž každý je reprezentován třídou Record, která se následně skládá z jednotlivých atributů reprezentovaných třídou Element. Existují čtyři základní typy atributu záznamu, a to DateElement, TextElement, NumberElement a HiddenElement. DateElement se používá pro časové hodnoty, TextElement pro textové hodnoty a NumberElement pro číselné hodnoty. HiddenElement slouží jako speciální typ, který umožnuje vyloučení atributů z deduplikačního procesu. K vytváření záznamů (třída Record) a jejich prvků (třída Element) se používá návrhový vzor stavitel (builder). Tento vzor umožňuje vytvářet složité objekty krok za krokem prostřednictvím zřetězených volání metod.

```
Table table = Table.of(records.stream()
        .map(item -> Record.builder()
                .element(TextElement.builder()
                        .name("title").value(item.getFirst()).build())
                .element(NumberElement.builder()
                        .name("average_rating").value(item.get(1)).build())
                .element(TextElement.builder()
                        .name("isbn").value(item.get(2)).build())
                .build())
        .toList());
```

Zdrojový kód 1: Ukázka kódu transformace dat na instanci třídy Table

Pro provádění deduplikace a spojování záznamů je potřeba transformovat data do přestavených struktur. Jeden z přístupů zahrnuje použití již zmíněného návrhového vzoru stavitel znázorněného v ukázce Zdrojový kód 1. Tento proces zaručuje přesnou transformaci dat do formátů srozumitelných knihovně bez ohledu na to, zda data pocházejí z databáze nebo jsou získána interní cestou nebo jiným způsobem. Tato metoda transformace dat může být zdlouhavá, pokud chceme rychle načíst data ve standardizovaném formátu a chceme s nimi

ihned provádět nějaké operace. Pro usnadnění tohoto procesu obsahuje třída Table dvě pomocné metody, a to file a csv.

```
public interface DatasetReader {

    Table read(File file) throws IOException;

    default Table read(String filename) throws IOException {
        return read(new File(filename));
    }
}
```

Zdrojový kód 2: Rozhraní DatasetReader určené pro načtení speciálního souboru

Metoda file přijímá dva parametry, a to soubor který chceme číst a třídu implementující rozhraní DatasetReader. Specifikace tohoto rozhraní je uvedena v ukázce Zdrojový kód 2. Rozhraní DatasetReader je jednoduchý interface, obsahující pouze dvě metody, z nichž jedna je výchozí. Implementační třída je povinna poskytnout implementaci metody read, která přijímá soubor a vrací objekt Table.

```
Table read = Table.csv("datasets/dataset1.csv",
        CsvOptions.fieldsWithHeader(
                    DatasetField.ofHidden("rec_id"),
                    DatasetField.ofText("given_name"),
                    DatasetField.ofText("sur_name"),
                    DatasetField.ofNumber("street_number"),
                    DatasetField.ofText("address_1"),
                    DatasetField.ofText("address_1"),
                    DatasetField.ofText("suburb"),
                    DatasetField.ofText("postcode"),
                    DatasetField.ofText("state"),
                    DatasetField.ofText("date_of_birth"),
                    DatasetField.ofText("soc_sec_id")));
```

Zdrojový kód 3: Ukázka kódu načtení dat ve formátu CSV

Druhou již zmíněnou pomocnou metodou třídy Table je metoda csv, která přijímá dva parametry, a to soubor který má být načten a třídu CsvOptions představující různé konfigurace pro čtení souboru CSV. Interně tato metoda využívá třídu CsvReader, implementaci rozhraní DatasetReader, která se stará o proces čtení souboru CSV.

8.3 Proces deduplikace a spojování záznamů

8.3.1 Čištění dat

Počáteční proces představuje čištění dat. V knihovně se k tomuto účelu používají dvě základní struktury, a to třída Pipeline a rozhraní Preprocessor. Třída Pipeline představuje pipeline pro zpracování dat, která umožňuje sekvenční zpracování vstupních dat prostřednictvím posloupnosti preprocesoru. Tato třída využívá návrhový vzor řetězce odpovědnosti (chain of

responsibility), kde je každá fáze v pipeline odpovědná za zpracování vstupu a jeho předání další fázi.

```
public class TrimPreprocessor implements Preprocessor<String, String> {

    @Override
    public String process(String value) {
        return value.strip();
    }
}
```

Zdrojový kód 4: Ukázka implementace třídy TrimPreprocessor

V rámci třídy Pipeline jsou jednotlivé prvky preprocesoru reprezentovány třídami implementujícími rozhraní Preprocessor. Knihovna implementuje čtyři základní preprocesory – TrimPreprocessor, ToLowerCasePreprocessor, RemoveSpecialCharactersPreprocessor, RemoveNonNumericCharactersPreprocessor. TrimPreprocessor odstraňuje počáteční a koncové bílé znaky, ToLowerCasePreprocessor převádí všechny znaky v řetězci na malá písmena, RemoveSpecialCharactersPreprocessor odstraňuje nealfanumerické znaky a RemoveNonNumericCharactersPreprocessor odstraňuje znaky, které nejsou číslice.

Čištění dat pracuje s jednotlivými atributy v záznamu, tyto vyčištěné hodnoty jsou poté využívaný pro indexování a porovnávání hodnot. Každý typ atributu je vybaven předdefinovanými výchozími preprocesory například TextElement využívá TrimPreprocessor a ToLowerCasePreprocessor. Uživatelé knihovny však mají při vytváření atributu možnost přidat další preprocesory nebo zavést vlastní.

8.3.2 Blokování

Následující fází procesu spojování záznamů a deduplikace představuje blokování. Blokování je klíčovým prvkem umožňujícím provádění deduplikace i u rozsáhlých datasetů. Podrobné vysvětlení blokování bylo uvedeno v dřívější teoretické části. Jádro knihovní části zabývající se blokováním tvoří rozhraní Indexer. Toto rozhraní slouží jako kontrakt, který musí třídy dodržovat, pokud chtějí implementovat některý z blokovacích algoritmů. V rámci rozhraní Indexer jsou dvě metody. Jedna, která přijímá dva objekty tabulky pro slučování záznamů, a druhá, která přijímá pouze jednu tabulku pro deduplikaci. Obě metody vrací kolekci instancí CandidatePair třídy, která reprezentuje dvoji kandidátních záznamů.

```
public interface Indexer {
    Collection<CandidatePair> index(Table firstTable, Table secondTable);

    default Collection<CandidatePair> index(Table table) {
        Collection<CandidatePair> index = index(table, table);
        return index.stream()
                .filter(candidate -> candidate.x() > candidate.y())
                .toList();
    }
}
```

Zdrojový kód 5: Rozhraní Indexer, které musí implementovat blokovací algoritmy

Knihovna obsahuje tři základní blokovací algoritmy FullIndexer, StandardBlocker a SortedNeighbourhoodBlocker. FullIndexer je jednoduchý algoritmus, který generuje všechny možné kombinace dvojic záznamů. StandardBlocker reprezentuje implementaci standardní blokovací metody, která spojuje podobné záznamy v blocích na základě blokujícího klíče. SortedNeighbourhoodBlocker implementuje metodu indexování sorted neighbourhood, která páruje záznamy na základě třídicího klíče, ale také bere v úvahu i záznamy v jejich okolí. Jak metoda standardní blokování, tak metoda sorted neighborhood jsou podrobně vysvětleny v teoretické části.

```
Indexer indexer = StandardBlocker.ofName("lastname");
Collection<CandidatePair> index = indexer.index(Table.of(records),
        Table.of(recordsToMatches));
```

Zdrojový kód 6: Ukázka použití blokování nad daty

Třídy StandardBlocker a SortedNeighbourhood obsahují pomocné statické tovární metody s názvem ofName a ofNames. Metoda ofName přijímá jeden řetězcový parametr, zatímco metoda ofNames přijímá pole řetězců. Tyto metody usnadňují výběr pojmenovaného sloupce v tabulce jako blokovacího klíče pro StandardBlocker nebo třídicího klíče pro SortedNeighbourhood.

8.3.3 Porovnání atributů a záznamů

Další klíčovou fází v procesu spojování záznamů a deduplikace je porovnávání záznamů. Jedná se o porovnání kandidátních dvojic záznamů s cílem zjistit jejich podobnost. Klíčovými strukturami při provádění porovnávání záznamů jsou třída Matcher a rozhraní NormalizedSimilary. Rozhraní NormalizedSimilary je kontraktem, který musí implementované algoritmy dodržovat, aby mohly být použity pro účely porovnávání atributů. NormalizedSimilary obsahuje pouze jednu abstraktní metodu, konkrétně normalizedSimilarity, která přijímá dva generické parametry a vrací jejich podobnosti.

```
public interface NormalizedSimilarity<T> {

    double normalizedSimilarity(final T firstValue, final T secondValue);

}
```

Zdrojový kód 7: Rozhraní NormalizedSimilarity, které implementují porovnávací algoritmy

Knihovna obsahuje rozsáhlou sadu algoritmu pro porovnání atributu, převážně pro textové řetězce, ale také pro číselné hodnoty nebo datumy. Algoritmy určené pro porovnání řetězce jsou poskytovány prostřednictvím tříd jako jsou Exact, Jaccard, Jaro, JaroWinkler, Levenshtein, DamerauLevenshtein a QGram. Třída Exact představuje nejjednodušší algoritmus podobnosti, který dosahuje nejvyšší podobnosti pouze pokud jsou si atributy přesně rovné. Oproti tomu ostatní zmíněné algoritmy používají složitější přístupy pro přesnější zjištění podrobnosti mezi dvěma řetězci. Konkrétně třída Jaccard implementuje Jaccardův koeficient, zatímco třídy Jaro a JaroWinkler zpracovávají Jarovu vzdálenost a její rozšíření, Jaro-Winklerovu vzdálenost. Podobně třídy Levenshtein a DamerauLevenshtein implementují Levenshteinovu editační vzdálenost a její rozšíření Damerau-Levenshteinova vzdálenost. Posledním implementovaným algoritmem určeným pro řetězce je Q-Gram v třídě QGram. Pro porovnání číselných hodnot slouží třídy NumericalAbsolute a NumericalPercentage. NumericalAbsolute implementuje algoritmus který toleruje absolutní číselnou odchylku, zatímco NumericalPercentage toleruje relativní procentuální odchylku. Pro hodnoty datumů nabízí knihovna algoritmus tolerující absolutní odchylku ve dnech prostřednictvím třídy DaysAbsolute. Podrobný popis všech těchto algoritmů, ať už pro textové řetězce, číselné hodnoty nebo hodnoty datumů, se nachází v teoretické části.

Algoritmy, porovnání atributu jsou specifikované přímo u jednotlivých atributů. Každý ze tří základních typů atributů má vlastní výchozí algoritmus, ten však lze při vytváření prvku změnit. Konkrétně u třídy TextElement se používá jako výchozí algoritmus JaroWinkler, u třídy NumberElement NumericalPercentage a u třídy DateElement DaysAbsolute.

```
Indexer indexer = StandardBlocker.ofName("given_name");
Matcher matcher = Matcher.builder().indexer(indexer).build();
Collection<ComparisonVector> match = matcher.match(FEBRLDataset1);
```

Zdrojový kód 8: Ukázka provádění porovnání záznamů

Pro provedení porovnání záznamů v jednom nebo dvou datasetech se používá již zmíněná třída Matcher. Třída Matcher využívá blokovacích technik ke snížení počtu porovnávaných hodnot, čímž a snižuje zátěže samotného porovnáváním záznamů. Proto vyžaduje při vytváření objektu instanci rozhraní Indexer. Matcher obsahuje pouze dvě veřejné metody, metodu match, která

přijímá jeden objekt tabulky, a další metodu match, která přijímá dva objekty tabulky. Obě metody vrací kolekci objektů ComparisonVector, kde třída reprezentuje srovnávací vektor, který obsahuje informace o podobnosti jednotlivých atributů v záznamu.

8.3.4 Klasifikace

V procesu spojování záznamů a deduplikace je klasifikace často jednou z koncových a někdy i závěrečných fází. Klasifikace je technika, která pomocí srovnávacích vektorů získaných z porovnání záznamů určuje, zda se jedná o shodu jinými slovy, jestli se páry hodnot vztahují ke stejné entitě z reálného světa. Jádrem části knihovny zabývající se klasifikací je rozhraní Classifier. Toto rozhraní je kontraktem, který musí dodržovat všechny algoritmy, které chtějí být použity pro klasifikaci.

```
public interface Classifier {
    List<ClassifiedRecord> classify(Collection<ComparisonVector> result);

    default List<Match> matches(List<ComparisonVector> result) {
        return classify(result).stream()
                .filter(classified -> classified instanceof Match)
                .map(classifiedRecord -> (Match) classifiedRecord)
                .toList();
    }

    default List<NotMatch> notMatches(List<ComparisonVector> result){
        return classify(result).stream()
                .filter(classified -> classified instanceof NotMatch)
                .map(classifiedRecord -> (NotMatch) classifiedRecord)
                .toList();
    }
}
```

Zdrojový kód 9: Rozhraní Classifier, které implementují klasifikační algoritmy

Jak je znázorněno na ukázce Zdrojový kód 9, rozhraní Classifier obsahuje tři metody classify, matches a notMatches. Z nich metoda classify je jedinou abstraktní metodou, kterou je třeba implementovat, zatímco ostatní dvě metody jsou výchozí. Metoda classify přijímá kolekci instancí třídy ComparisonVector, vrácené z porovnávání záznamů třídou Matcher, a vrací kolekci instancí rozhraní ClassifiedRecord. Rozhraní ClassifiedRecord je speciální zalepené rozhraní, které přesně určuje, jaké třídy může dané rozhraní implementovat. V tomto případě jsou povolené třídy Match reprezentující shodu a NotMatch reprezentující neshodu. Tuto vlastnost poté využívají metody matches a notMatches, kde metoda matches vrací pouze shody a metoda notMatches pouze neshody.

```
public sealed interface ClassifiedRecord permits Match, NotMatch {
    Record first();
    Record second();
    ComparisonVector comparisonVector();
}
```

Zdrojový kód 10: Zalepené rozhraní, které reprezentuje klasifikovaný objekt

Knihovna nabízí tři implementace klasifikačních algoritmů. Prvním z těch algoritmů je ThresholdClassifier, který využívá přístup založený na limitních hodnotách. Tento klasifikátor funguje na principu, že sčítá hodnoty podobnosti ze srovnávacího vektoru do celkové hodnoty podobnosti a porovnává jí se zadaným limitem pro klasifikaci. Pokud hodnota vyhovuje limitu, je hodnota považována za shodu, pokud ne, považuje se za neshodu. Druhý klasifikátor implementovaný v knihovně je KMeansClassifier. Jedná se o neřízený klasifikátor, který je založený na shlukovacím algoritmu K-means. Posledním klasifikátorem implementovaným v knihovně je SVMClassifier. SVMClassifier je řízený klasifikátor, založený na metodě podpůrných vektorů. Všechny uvedené klasifikační algoritmy jsou podrobně popsány v teoretické části.

8.3.5 Deduplikace a spojování záznamů

V předchozích kapitolách byl představen základní průběh procesu deduplikace a spojování záznamů který byl zakončen klasifikací jednotlivých záznamů. Nicméně tento přístup nemusí vyhovovat všem scénářům, zejména uživatelům, kteří nemají odborné znalosti v oblasti spojování záznamů nebo deduplikace záznamů. Knihovna proto nabízí další přístup, kdy uživatelé musí zadat pouze jednu tabulku pro deduplikaci nebo dvě pro spojování záznamů a následně obdrží deduplikovanou nebo spojenou tabulku. Tuto funkcionalitu umožňuje knihovna pomocí tříd Deduplicator a Linker.

```
Deduplicator deduplicator = new Deduplicator();
Table deduplicated = deduplicator.deduplicate(FEBRLDataset1);
```

Zdrojový kód 11: Ukázka základního použití deduplikace pomocí třídy Deduplicator

Základní příklad použití třídy Deduplicator naleznete v Zdrojový kód 11. Použití třídy Deduplicator vyžaduje tři třídy a to Matcher, Classifier a Merger. Aby se usnadnilo používání začínajícím uživatelům, je možné vytvořit třídu bez zadání některé z uvedených tříd. V takovém případě jsou použity výchozí třídy, konkrétně třída Matcher s blokovacím algoritmem FullIndexer, třída ThresholdClassifier pro klasifikaci a třída RollTheDiceMerger pro fúzi dat. Třída Deduplicator obsahuje jedinou veřejnou metodu deduplicate, která přijímá instanci třídy

Table pro deduplikaci a vrací novou instanci třídy Table s odstraněnými duplicitními hodnotami.

```
Linker linker = new Linker();
Table linked = linker.link(FEBRLDataset4a, FEBRLDataset4b);
```

Zdrojový kód 12: Ukázka základního použití spojování záznamů pomocí třídy Linker

Ekvivalentem třídy Deduplicator pro spojování záznamů je třída Linker, jejíž příklad použití je vidět ve Zdrojový kód 12. Podobně jako třída Deduplicator závisí třída Linker na třídách Matcher, Classifier a Merger. K usnadnění použití pro začínající uživatele je možné vytvořit třídu i bez zadání některé z těchto zmíněných tříd, obdobně jak u třídy Deduplicator. V takových případech jsou využity výchozí třídy, které přesně kopírují nastavení třídy Deduplicator, a to třída Matcher s blokovacím algoritmem FullIndexer, třída ThresholdClassifier a třída RollTheDiceMerger. Třída Linker disponuje pouze jednou veřejnou metodou link, která přijímá dvě instance třídy Table pro spojování záznamů a vrací novou instanci třídy Table reprezentující výsledek procesu spojování.

Třídy Deduplicator i Linker využívají rozhraní Merger pro sloučení shodných záznamů do jediného záznamu, což je proces často označovaný jako fúze dat. Rozhraní Merger implementuje jediná třída, a to RollTheDiceMerger, která představuje implementaci fúze dat nazvanou „hod kostkou“ (roll the dice). Tento algoritmus a další algoritmy fúze dat jsou popsány v teoretické části.

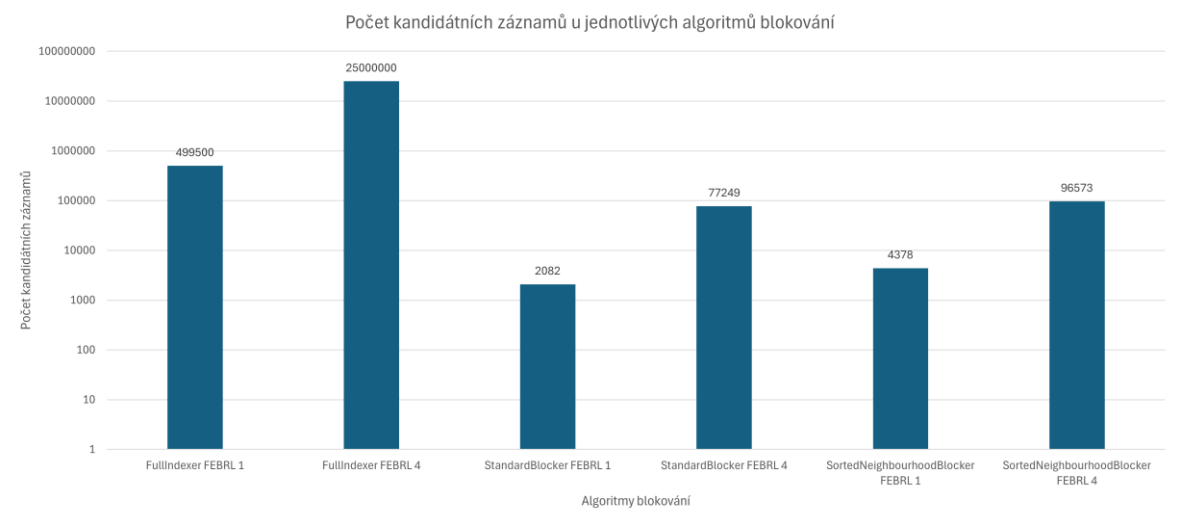
8.4 Využití knihovny nad daty

V této části bude předvedeno využití knihovny na testovacích datech. Testovací data budou využívat datasety, které poskytuje program Febrl, konkrétně datasety 1 a FEBRL 4. Dataset FEBRL 1 se skládá z 1000 záznamů, z nichž polovina jsou originály a polovina duplikáty, přičemž každý originál má přesně jeden odpovídající duplikát. Naproti tomu dataset FEBRL 4 se skládá ze dvou datasetů, a to dataset4a.csv, který obsahuje 5000 originálních záznamů, a dataset4b.csv obsahující 5000 duplicitních záznamů. Oba tyto datasety jsou speciálně vytvořeny pro testování deduplikace a spojování záznamů.

8.4.1 Blokování

Prvním představeným krokem procesu spojování záznamu a deduplikace na testovacích datech bude blokování. Jak již bylo uvedeno, knihovna obsahuje tři základní algoritmy FullIndexer, StandardBlocker a SortedNeighbourhoodBlocker. Použití těchto implementací na testovacích

datech je vidět na Obrázek 7 který znázorňuje počet dvojic kandidátních záznamů vrácených jednotlivými algoritmy.

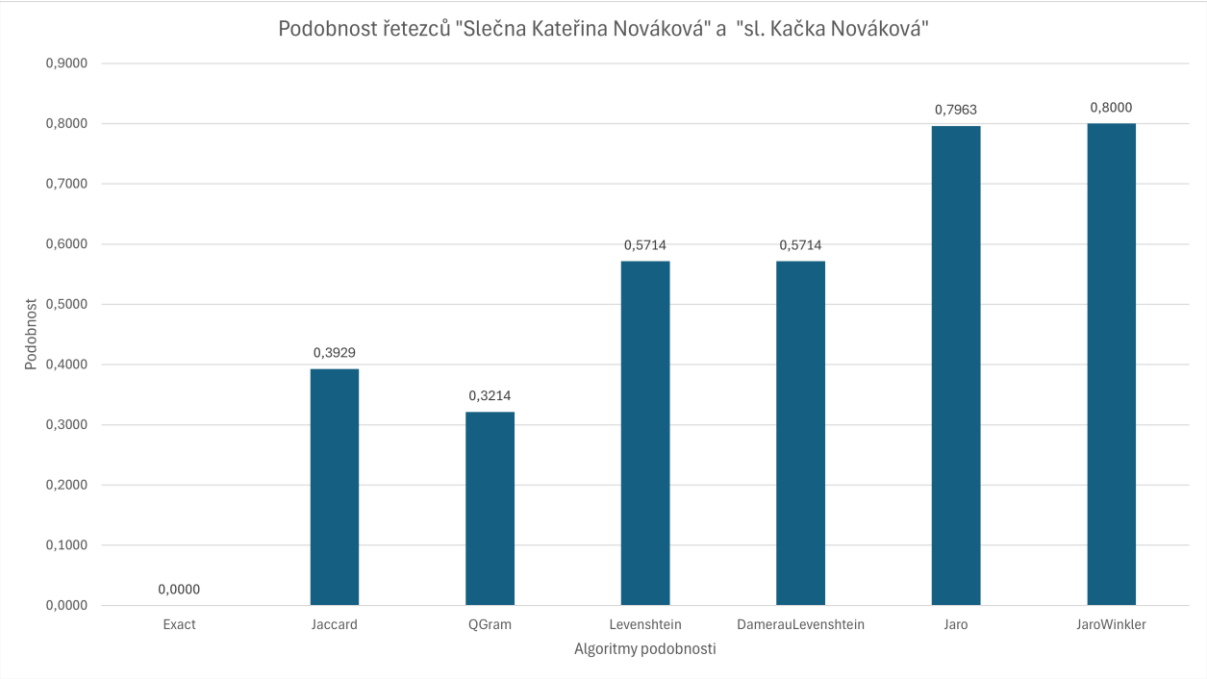


Obrázek 7: Graf počtu kandidátních záznamů u jednotlivých algoritmů blokování v logaritmickém měřítku

Třída FullIndexer poskytuje podle očekávání nejvyšší počet kandidátních záznamů, jelikož generuje všechny možné kombinace dvojic záznamů. Kromě toho graf ilustruje, že implementace tříd StandardBlocker a SortedNeighbourhoodBlocker vracejí ve srovnání s třídou FullIndexer výrazně méně kandidátských dvojic záznamů. Nicméně mezi těmito dvěma algoritmy vrací SortedNeighbourhoodBlocker vyšší počet kandidátních záznamů.

8.4.2 Porovnávací funkce

Porovnávací funkce nebudou demonstrovány na testovacích datech, ale na řetězcích, "Slečna Kateřina Nováková" a "sl. Kačka Nováková" jež reprezentují dva řetězce, které patří ke stejné reálně entitě. Toto nastavení nám umožňuje sledovat podobnost získanou jednotlivými algoritmy. Knihovna, jak již bylo uvedeno, obsahuje řadu algoritmů, včetně Exact, Jaccard, Jaro, JaroWinkler, Levenshtein, DamerauLevenshtein a QGram. Aplikace těchto implementací na testovací řetězce je znázorněna v grafu v rámci Obrázek 8, který ukazuje podobnost mezi příslušnými řetězci.

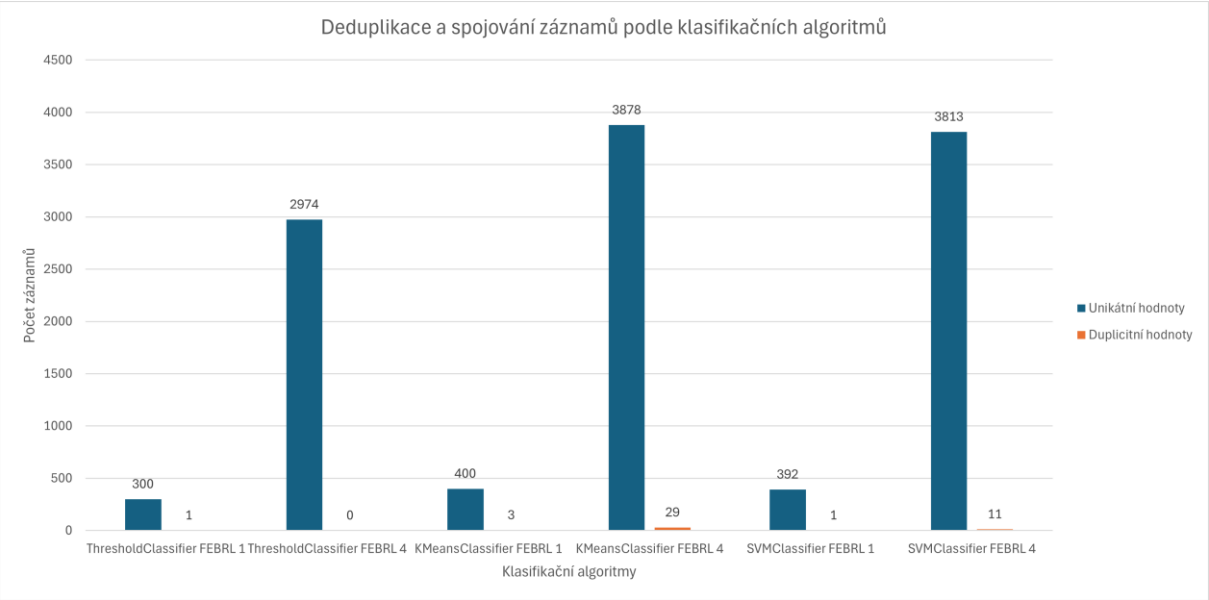


Obrázek 8: Graf podobnosti nad testovacími řetězci u jednotlivých algoritmů podobnosti

Z grafu je zřejmé, že algoritmus JaroWinkler vykazuje na daných řetězcích vyšší podobnost, díky čemuž byl zvolen jako výchozí algoritmus pro porovnávání textových atributů. Za tímto algoritmem z hlediska podobnosti následuje nevylepšená verze předešlého algoritmu, a to algoritmus Jaro. Tyto dva algoritmy prokázaly nejvyšší podobnost mezi dvěma testovacími řetězci, které patří ke stejné reálné entitě. Toto vysoké skóre však není překvapivé, protože oba algoritmy byly výslovně vytvořeny pro porovnávání jmen, což přesně odpovídá testovacím řetězcům.

8.4.3 Deduplikace a spojování záznamů

Dalším představením krokem procesu spojování záznamu a deduplikace na testovacích datech bude samotná deduplikace a spojování záznamů se zaměřením na použité klasifikační algoritmy. Již dříve bylo uvedeno, že knihovna zahrnuje tři základní algoritmy ThresholdClassifier, KMeansClassifier a SVMClassifier. Při použití těchto implementací na testovacích datech je na grafu v Obrázek 9 znázorněn počet unikátních deduplikovaných a spojených záznamů spolu s počtem duplicitních záznamů, které nebyly spojeny, avšak měly být.



Obrázek 9: Graf deduplikace a spojování záznamů podle klasifikačních algoritmů

Nad daty používanými pro klasifikaci byla provedeno sorted neighbourhood blokování podle jména z důvodu náročnosti některých klasifikačních metod , které nebyly pro tyto datasety použitelné. Z grafu vyplývá, že algoritmus ThresholdClassifier zaostává oproti ostatním klasifikačním algoritmům. Výhodou je však jeho jednoduchost díky čemuž je velmi rychlý a je ho možné použít i na větších datasetech. Naproti tomu klasifikační algoritmy KMeansClassifier a SVMClassifier poskytují srovnatelné výsledky, z pohledu rychlosti provádění samotné klasifikace je algoritmus KMeansClassifier obecně rychlejší než SVMClassifier.

ZÁVĚR

Diplomová práce se věnovala problematice spojování záznamů a souvisejícím algoritmům. Hlavním cílem práce bylo zkoumání možností, způsobů a metod, které se s pojmem deduplikace pojí. Z rozsáhlé analýzy zahrnující i široký sběr podkladů byla navržena, implementována a úspěšně otestována knihovna pro Java Match4j včetně testování implementovaných algoritmů nad kolekcí dat.

V teoretické části byl popsán celý proces deduplikace a spojování záznamů, zahrnující předzpracování dat, blokování, porovnávání záznamů a s tím souvisejících algoritmů, klasifikace záznamů a fúze dat. V práci byly mimo jiné zkoumány algoritmy určené pro řešení podobnosti řetězců, zahrnující mimo jiné Jaro-Winklerovu vzdálenost, Levenshteinovu vzdálenost, Damerau-Levenshteinovu vzdálenost, Jaccardův index a podobnost Q-gramů. Dále byly v práci popsány případy reálného využití deduplikace a spojování záznamů, práce se více zaměřila na open-source řešení, a to Splink, JedAI, Febrl, ale i na ostatní podobná řešení. Podrobné zmapování jednotlivých přístupů, možností, technik, metod a dalších oblastí v teoretické části přispělo k tvorbě praktického výstupu práce, jelikož byl díky tomu prozkoumám stanovený rámec řešené problematiky.

V praktické části byla vyvinuta knihovna v jazyce Java, která umožňuje provádění kompletního procesu deduplikace a spojování záznamů ve verzi LTS 21 a s využitím třech externích knihoven opencsv, logback-classic a smile-core. Pro ověření v průběhu vývoje knihovny byly použity knihovny JUnit5 a AssertJ. Celý proces byl v práci přehledně zaznamenán včetně výsledků testování klasifikačních algoritmů ThresholdClassified, KMeansClassifier a SVMClassifier.

I přes dosažené výsledky však existují oblasti, které by bylo možné v dalším vývoji rozšířit. Současné řešení by se mohlo rozvíjet o další algoritmy pro porovnávání řetězců a klasifikaci. Další potenciální rozšíření knihovny by mohlo zahrnovat možnost provádět výpočty na platformě Apache Spark nebo obdobných platformách, což by umožnilo provádění deduplikací a spojování záznamů i nad datasety, které jsou pro současné řešení moc rozsáhlé.

POUŽITÁ LITERATURA

[1] CHRISTEN, Peter. *Data matching: concepts and techniques for record linkage, entity resolution, and duplicate detection*. Data-centric systems and applications. New York: Springer, 2012. ISBN 9783642311642.

[2] MURRAY, Jared S. Probabilistic Record Linkage and Deduplication after Indexing, Blocking, and Filtering. Online. *Journal of Privacy and Confidentiality*. 2015, roč. 7. ISSN 2575-8527. Dostupné z: https://doi.org/10.29012/jpc.v7i1.643. [cit. 2023-12-01].

[3] CHRISTEN, Peter a GOISER, Karl. Quality and Complexity Measures for Data Linkage and Deduplication. In: GUILLET, Fabrice J. a HAMILTON, Howard J. (ed.). *Quality Measures in Data Mining*. Studies in Computational Intelligence. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, s. 127-151. ISBN 978-3-540-44911-9. Dostupné z: https://doi.org/10.1007/978-3-540-44918-8_6.

[4] BINETTE, Olivier a STEORTS, Rebecca C. (Almost) all of entity resolution. *Science Advances*. 2022, roč. 8, č. 12. ISSN 2375-2548. Dostupné z: https://doi.org/10.1126/sciadv.abi8021.

[5] RANDALL, Sean M; FERRANTE, Anna M; BOYD, James H a SEMMENS, James B. The effect of data cleaning on record linkage quality. *BMC Medical Informatics and Decision Making*. 2013, roč. 13, č. 1. ISSN 1472-6947. Dostupné z: https://doi.org/10.1186/1472-6947-13-64.

[6] SARFIN, Levy Rachel. Data Quality Dimensions: How Do You Measure Up?. Precisely. Online. 14.11.2022. Dostupné z: https://www.precisely.com/blog/data-quality/data-quality-dimensions-measure. [cit. 2023-11-20].

[7] GU, Lifang a BAXTER, Rohan. Adaptive Filtering for Efficient Record Linkage. Online. In: *Proceedings of the 2004 SIAM International Conference on Data Mining*. Philadelphia, PA: Society for Industrial and Applied Mathematics, 2004, s. 477-481. ISBN 978-0-89871-568-2. Dostupné z: https://doi.org/10.1137/1.9781611972740.50. [cit. 2023-12-01].

[8] ELIAS, Daniel a POON, Josiah. Neighbourhood Blocking for Record Linkage. Online. In: CHANG, Lijun; GAN, Junhao a CAO, Xin (ed.). *Databases Theory and Applications*. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2019, s. 57-78. ISBN 978-3-030-12078-8. Dostupné z: https://doi.org/10.1007/978-3-030-12079-5_5. [cit. 2023-12-02].

[9] CHRISTEN, Peter. A Survey of Indexing Techniques for Scalable Record Linkage and Deduplication. Online. *IEEE Transactions on Knowledge and Data Engineering*. 2012, roč. 24, č. 9, s. 1537-1555. ISSN 1041-4347. Dostupné z: https://doi.org/10.1109/TKDE.2011.127. [cit. 2023-11-20].

[10] ELZIKY, Mayada A.; IBRAHIM, Dina M. a SARHAN, Amany M. Improved Duplicate Record Detection Using ASCII Code Q-gram Indexing Technique. Online. *Arabian Journal for Science and Engineering*. 2018, roč. 43, č. 12, s. 7409-7420. ISSN 2193-567X. Dostupné z: https://doi.org/10.1007/s13369-018-3105-6. [cit. 2023-12-02].

[11] EFTHYMIOU, Vasilis; PAPADAKIS, George; PAPASTEFANATOS, George; STEFANIDIS, Kostas a PALPANAS, Themis. Parallel meta-blocking for scaling entity resolution over big heterogeneous data. Online. *Information Systems*. 2017, roč. 65, s. 137-157. ISSN 03064379. Dostupné z: https://doi.org/10.1016/j.is.2016.12.001. [cit. 2023-12-05].

[12] PAPADAKIS, George; KOUTRIKA, Georgia; PALPANAS, Themis a NEJDL, Wolfgang. Meta-Blocking: Taking Entity Resolutionto the Next Level. Online. *IEEE Transactions on Knowledge and Data Engineering*. 2014, roč. 26, č. 8, s. 1946-1960. ISSN 1041-4347. Dostupné z: https://doi.org/10.1109/TKDE.2013.54. [cit. 2023-12-08].

[13] PAPADAKIS, George; PAPASTEFANATOS, George a KOUTRIKA, Georgia. Supervised meta-blocking. Online. *Proceedings of the VLDB Endowment*. 2014, roč. 7, č. 14, s. 1929-1940. ISSN 2150-8097. Dostupné z: https://doi.org/10.14778/2733085.2733098. [cit. 2023-12-08].

[14] NAUMANN, Felix a HERSCHEL, Melanie. *An Introduction to Duplicate Detection*. Online. Synthesis Lectures on Data Management. Cham: Springer International Publishing, 2010. ISBN 978-3-031-00707-1. Dostupné z: https://doi.org/10.1007/978-3-031-01835-0.

[15] CHRISTEN, Peter. A Comparison of Personal Name Matching: Techniques and Practical Issues. Online. In: *Sixth IEEE International Conference on Data Mining - Workshops (ICDMW'06)*. Hong Kong: IEEE, 2006, s. 290-294. ISBN 0-7695-2702-7. Dostupné z: https://doi.org/10.1109/ICDMW.2006.2. [cit. 2023-12-12].

[16] DEZA, Michel Marie a DEZA, Elena. *Encyclopedia of Distances*. Online. 4. vydání. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016. ISBN 978-3-662-52843-3. Dostupné z: https://doi.org/10.1007/978-3-662-52844-0.

[17] DALIANIS, Hercules. *Clinical Text Mining*. Online. Cham: Springer International Publishing, 2018. ISBN 978-3-319-78502-8. Dostupné z: https://doi.org/10.1007/978-3-319-78503-5.

[18] GOMEDE, Everton. *The Damerau–Levenshtein Distance: A Powerful Measure for String Similarity*. Online. Medium. 10. 6. 2023. Dostupné z: https://medium.com/@evertongomede/the-damerau-levenshtein-distance-a-powerful-measure-for-string-similarity-22dc4b150f0a. [cit. 2023-11-20].

[19] MOHAMMED ADDOKALI, Belaid a ALI ELBURASE, Ebrahem. Using Levenshtein Distance Algorithm to Increase Database Search Efficiency and Accuracy. Online. *Libyan Society for Curriculum and Instructional Strategies*. 2022, č. 10. Dostupné z: https://uot.edu.ly/publication_item.php?pubid=5595. [cit. 2024-02-10].

[20] SAMIR, Fatema. *Unveiling String Distances: A Dive into Levenshtein and Jaro Distances in Databases*. Online. DEV Community. 14. 7. 2023. Dostupné z: https://dev.to/fatemasamir/unveiling-string-distances-a-dive-into-levenshtein-and-jaro-distances-in-databases-b6j. [cit. 2024-02-10].

[21] HALL, Tim. *UTL_MATCH : String Matching by Testing Levels of Similarity/Difference*. Online. ORACLE-BASE.com. 2023. Dostupné z: https://oracle-base.com/articles/11g/utl_match-string-matching-in-oracle. [cit. 2024-02-22].

[22] *Data Quality Operators*. In: Oracle [online]. 2023. Dostupné z: https://docs.oracle.com/en/database/oracle/oracle-database/23/sqlrf/data-quality-operators.html#GUID-C13A179C-1F82-4522-98AA-E21C6504755E. [cit. 2024-02-22].

[23] BARAKA, Ahmed. *Using Fuzzy Matching in Oracle Database 23c*. Online. Ahmedbaraka.com. 2023. Dostupné z: https://www.ahmedbaraka.com/fuzzy-matching-in-oracle-23c/. [cit. 2024-02-22].

[24] *Fuzzystrmatch — determine string similarities and distance*. In: The PostgreSQL Global Development Group [online]. 2024. Dostupné z: https://www.postgresql.org/docs/current/fuzzystrmatch.html. [cit. 2024-02-23].

[25] *Use the fuzzystrmatch extension to calculate the similarity between strings*. In: Alibaba Cloud [online]. 31. 3. 2023. Dostupné z: https://www.alibabacloud.com/help/en/rds/apsaradb-rds-for-postgresql/use-the-fuzzystrmatch-extension-to-calculate-the-similarity-between-strings. [cit. 2024-02-23].

[26] RAMSEY, Paul. *Fuzzy Name Matching in Postgres*. Online. Crunchy Data Solutions, Inc. 22. 2. 2021. Dostupné z: https://www.crunchydata.com/blog/fuzzy-name-matching-in-postgresql. [cit. 2024-02-24].

[27] WANG, Fei a WANG, Hongbo. Record Linkage Using the Combination of Twice Iterative SVM Training and Controllable Manual Review. Online. In: *2016 IEEE 14th Intl Conf on Dependable, Autonomic and Secure Computing, 14th Intl Conf on Pervasive Intelligence and Computing, 2nd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*. IEEE, 2016, s. 31-38. ISBN 978-1-5090-4065-0. Dostupné z: https://doi.org/10.1109/DASC-PICom-DataCom-CyberSciTec.2016.21. [cit. 2024-03-13].

[28] CHRISTEN, Peter. Automatic record linkage using seeded nearest neighbour and support vector machine classification. Online. In: *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. New York, NY, USA: ACM, 2008, s. 151-159. ISBN 9781605581934. Dostupné z: https://doi.org/10.1145/1401890.1401913. [cit. 2024-03-13].

[29] GOISER, Karl a CHRISTEN, Peter. Towards automated record linkage. Online, In: *AusDM '06: Proceedings of the fifth Australasian conference on Data mining and analystics.* 2006, roč. 61, s. 23-31. Dostupné z: https://dl.acm.org/doi/10.5555/1273808.1273812. [cit. 2023-12-13].

[30] ELFEKY, M.G.; VERYKIOS, V.S. a ELMAGARMID, A.K. TAILOR: a record linkage toolbox. Online. In: *Proceedings 18th International Conference on Data Engineering*. San Jose: IEEE Comput. Soc, 2002, s. 17-28. ISBN 0-7695-1531-2. Dostupné z: https://doi.org/10.1109/ICDE.2002.994694. [cit. 2023-12-14].

[31] GU, Lifang a BAXTER, Rohan. Decision Models for Record Linkage. Online. In: WILLIAMS, Graham J. a SIMOFF, Simeon J. (ed.). *Data Mining*. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2006, s. 146-160. ISBN 978-3-540-32547-5. Dostupné z: https://doi.org/10.1007/11677437_12. [cit. 2023-12-16].

[32] SHEARER, Michael. *Hands-On Entity Resolution: A Practical Guide to Data Matching with Python*. O'Reilly Media, 2024. ISBN 978-1098148485.

[33] DONG, Yongquan; DRAGUT, Eduard C. a MENG, Weiyi. Normalization of Duplicate Records from Multiple Sources. Online. *IEEE Transactions on Knowledge and Data Engineering*. 2019, roč. 31, č. 4, s. 769-782. ISSN 1041-4347. Dostupné z: https://doi.org/10.1109/TKDE.2018.2844176. [cit. 2024-02-26].

[34] BLEIHOLDER, Jens, 2010. Data Fusion and Conflict Resolution in Integrated Information Systems. Potsdam. Disertační práce. University of Potsdam. Faculty of Mathematics and Natural Sciences.

[35] CANALLE, Gabrielle Karine; SALGADO, Ana Carolina a LOSCIO, Bernadette Farias. A survey on data fusion: what for? in what form? what is next? Online. *Journal of Intelligent Information Systems*. 2021, roč. 57, č. 1, s. 25-50. ISSN 0925-9902. Dostupné z: https://doi.org/10.1007/s10844-020-00627-4. [cit. 2024-03-05].

[36] BAKHTOUCHI, Abdelghani. Data reconciliation and fusion methods: a survey. Online. *Applied Computing and Informatics*. 2020, roč. 18, č. 3/4, s. 182-194. ISSN 2634-1964. Dostupné z: https://doi.org/10.1016/j.aci.2019.07.001. [cit. 2024-03-05].

[37] CLEATON, Mary; HALL, Johanna; SHIPSEY, Rachel; WHITE, Zoe a XHAFERAJ, Kristina. A case study of using Splink: Census duplicate matching. *Proceedings of Statistics Canada Symposium*. 2022.

[38] *Splink: MoJ's open source library for probabilistic record linkage at scale.* Online. GOV.UK. 16.7.2021. Dostupné z: https://www.gov.uk/government/publications/joined-up-data-in-government-the-future-of-data-linking-methods/splink-mojs-open-source-library-for-probabilistic-record-linkage-at-scale. [cit. 2023-12-28].

[39] LINACRE, Robin; LINDSAY, Sam; MANASSIS, Theodore; SLADE, Zoe a HEPWORTH, Tom. Splink: Free software for probabilistic record linkage at scale. Online. *International Journal of Population Data Science*. 2022, roč. 7, č. 3. ISSN 2399-4908. Dostupné z: https://doi.org/10.23889/ijpds.v7i3.1794. [cit. 2023-12-28].

[40] PAPADAKIS, George; TSEKOURAS, Leonidas; THANOS, Emmanouil; GIANNAKOPOULOS, George; PALPANAS, Themis et al. The return of jedAI. Online. *Proceedings of the VLDB Endowment*. 2018, roč. 11, č. 12, s. 1950-1953. ISSN 2150-8097. Dostupné z: https://doi.org/10.14778/3229863.3236232. [cit. 2023-12-28].

[41] PAPADAKIS, George, Leonidas TSEKOURAS, Emmanouil THANOS, Nikiforos PITTARAS, Giovanni SIMONINI, Dimitrios SKOUTAS, Paul ISARIS, George GIANNAKOPOULOS, Themis PALPANAS a Manolis KOUBARAKIS. JedAI3 : beyond batch, blocking-based Entity Resolution [online]. 2020. *B.m.: OpenProceedings.org*. Dostupné z: https://doi.org/10.5441/002/EDBT.2020.74. [cit. 2023-12-29].

[42] CHRISTEN, Peter. Development and user experiences of an open source data cleaning, deduplication and record linkage system. Online. *ACM SIGKDD Explorations Newsletter*. 2009, roč. 11, č. 1, s. 39-48. ISSN 1931-0145. Dostupné z: https://doi.org/10.1145/1656274.1656282. [cit. 2024-01-30].

[43] CHRISTEN, Peter. Febrl: an open source data cleaning, deduplication and record linkage system with a graphical user interface. Online. In: *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. New York, NY, USA:

ACM, 2008, s. 1065-1068. ISBN 9781605581934. Dostupné z: https://doi.org/10.1145/1401890.1402020. [cit. 2024-02-01].

[44] CHRISTOPHIDES, Vassilis; EFTHYMIOU, Vasilis; PALPANAS, Themis; PAPADAKIS, George a STEFANIDIS, Kostas. An Overview of End-to-End Entity Resolution for Big Data. Online. *ACM Computing Surveys*. 2021, roč. 53, č. 6, s. 1-42. ISSN 0360-0300. Dostupné z: https://doi.org/10.1145/3418896. [cit. 2024-02-01].

[45] Opster Team. *Elasticsearch Text Analyzers – Tokenizers, Standard Analyzers, Stopwords and More*. Online. Opster. 28. 1. 2024. Dostupné z: https://opster.com/guides/elasticsearch/data-architecture/elasticsearch-text-analyzers/ [cit. 2024-05-05].

[46] Opster Team. *Elasticsearch Fuzzy Match: Advanced Techniques and Best Practices*. Online. Opster. 28. 1. 2024. Dostupné z: https://opster.com/guides/elasticsearch/search-apis/elasticsearch-fuzzy-match-techniques/ [cit. 2024-05-05].

[47] KONDA, Pradap; DAS, Sanjib; SUGANTHAN G. C., Paul; DOAN, AnHai; ARDALAN, Adel et al. Magellan: toward building entity matching management systems. Online. *Proceedings of the VLDB Endowment*. 2016, roč. 9, č. 12, s. 1197-1208. ISSN 2150-8097. Dostupné z: https://doi.org/10.14778/2994509.2994535. [cit. 2024-02-05].

[48] KUĆ, Rafal. *SLF4J Tutorial: Loggers, Levels & How to Configure for Java Applications with Examples*. Online. Sematext Group. 12. 3. 2021. Dostupné z: https://sematext.com/blog/slf4j-tutorial/. [cit. 2024-03-27].

[49] KUĆ, Rafal. *Logback Configuration Example: Tutorial on How to Use It for Logging in Java*. Online. Sematext Group. 12. 3. 2021. Dostupné z: https://sematext.com/blog/logback-tutorial/. [cit. 2024-03-27].

PŘÍLOHY

PŘÍLOHA A – ZDROJOVÝ KÓD KNIHOVNY MATCH4J

Obsah přílohy práce:

- Zdrojový kód knihovny Match4j