UNIVERZITA PAVLA JOZEFA ŠAFÁRIKA V KOŠICIACH
PRÍRODOVEDECKÁ FAKULTA

UNIFIKÁCIA METADÁT O PRODUKTOCH INTERNETOVÝCH OBCHODOV

2015 **Bc. Jozef Kuper**

Obsah

Úvod

Táto práca je súčasťou univerzitného projektu „kapsa“, ktorý okrem iného extrahuje informácie o produktoch z rôznych internetových zdrojov. Jedným z cieľov projektu je uchovať extrahované dáta o produktoch pod jedným portálom, pričom dáta je potrebné uchovávať v efektívnej dátovej štruktúre a samozrejme je vhodné vyhnúť uchovávaniu duplicít. Keďže takýchto obchodov a produktov v nich je obrovské množstvo, je potrebné dáta čo najpresnejšie špecifikovať alebo kategorizovať do akej domény a na ktorý atribút sa daná informácia vzťahuje a pod. Ak napríklad predpokladáme, že chceme získať dáta zo 100 rôznych internetových obchodov, pričom každý z nich má v priemere 100 produktov (čo je pri reálnych e-shopoch pomerne nízky odhad) a zároveň každý produkt obsahuje aspoň 10 atribútov, tak nakoniec získame 100 000 atribútov, ktoré je potrebné anotovať pred uložením do systému. Navyše ak internetový obchod znení ponuku produktov, tak je potrebné dáta aktualizovať. Keďže manuálna anotácia dát je v tomto prípade časovo náročná, bolo by vhodné vymyslieť systém, ktorý by takúto prácu uľahčil.

Hlavným cieľom práce je navrhnúť dátovú štruktúru a postupy na odhaľovanie a unifikáciu atribútov a domén produktov extrahovaných z internetových obchodov. Keďže nie je možné vytvoriť automatizovaný program na unifikáciu dát, tak sme sa zamerali na problém identifikácie podobných atribútov a domén produktov Následne je potrebné navrhnuté štruktúry a aplikáciu implementovať v prostredí Java. Na odhaľovanie podobností atribútov a domén sme využili fulltextové vyhľadávanie. Pomocou takéhoto vyhľadávania je možné identifikovať podobnosti metadát produktov v rozsiahlych dátových setoch. V práci sme sa zaoberali vytváraním vhodného formátu, ktorý by sme mohli využiť pri indexovaní dát do fulltextového vyhľadávača. Na základe formátu indexovaných dát sme následne vytvárali vhodné vstupy pre fulltextový vyhľadávač.

Implementované riešenia je následne vhodné otestovať na extrahovaných dátach z rozličných internetových obchodov. V práci sme program testovali na dátových setoch z dvoch rôznych internetových zdrojov a následne sme vyhodnocovali efektivitu implementovaného riešenia.

1 Unifikácia - definícia problému

V práci sa nebudem zaoberať extrakciou dát a ich následným spracovávaním a ukladaním do prijateľnej štruktúry , pretože sa to netýka problematiky tej to práce. Extrakcia dát je inou časťou projektu. V práci však budeme pracovať z dátami, ktoré mi ponúkne už existujúci nástroj na extrahovanie dát. Tento nástroj nám ponúkne štruktúrované dáta s ktorými budeme pracovať.

1.1 Unifikácia domén

Prvým zo základných problémov je zjednotenie domén. Ak chceme aby produkty spadajúce pod jednu doménu mali aj v atribúte „doména“ rovnaký reťazec, je potrebné im ho priradiť. Je zrejmé, že ak sa jedná o rovnaký produkt, tak bude spadať aj pod rovnakú doménu. Po extrahovaní dát z rôznych zdrojov, však tieto domény môžu mať rôznu reprezentáciu.

Príklad: Predpokladajme, že máme dané dve relácie R a S v štruktúrovanej forme. Dáta boli extrahované z rôznych zdrojov a majú rozdielnu reprezentáciu domén.

Tabuľka 1: relácia R

ID	Názov	Doména	Atrinút1	Atribút2	Atribút3	...
1	Canon EOS 20D	Fotoaparát	???	???	???	...

Tabuľka 2: relácia S

ID	Názov	Doména	Atrinút1	Atribút2	Atribút3	...
1	Canon EOS 20D	Elektronika	???	???	???	...

Ak chceme zjednotiť dva rovnaké produkty, tak je asi neprehľadné rozdeliť ich do dvoch rôznych domén. Predpokladajme, že nástroj na extrahovanie dát nám ponúkne štruktúrované dáta, v ktorých je potrebné unifikovať doménu. Asi najefektívnejšou možnosťou je pozrieť na domény, ktoré sme už spracovávali a manuálne prepísať doménu produktu. Ako však takýto proces automatizovať? Jednou z možností je vybrať si jednu a použiť ju. Problém však nastáva pri väčšom množstve relácii, ktoré majú

spornú doménu. Ďalšou z možností je pozrieť sa do štruktúry dát a zistiť pri ktorej doméne sa daný produkt už vyskytol. Čo však má systém spraviť vtedy, ak nenašiel žiadnu zhodu? Asi by bolo vhodné vytvoriť si vlastnú množinu domén, ktorá bude obsahovať jednotné názvy. Jedným z hlavných problémov však nastáva vtedy, keď začneme uvažovať hierarchiu domén. Akým spôsobom mám prehlásiť, že produkt, ktorý má doménu „*elektronika*“ v podstate patrí do subdomény „*fotoaparát*“.

Najefektívnejší prístup je pravdepodobne manuálne priradenie konkrétnej domény alebo subdomeny. Ak predpokladáme, že sme takéto priradenie už v minulosti robili, alebo že máme dostatočne rozsiahlu štruktúru, v ktorej nájdeme rovnaké produkty s už priradenou doménou, tak je možné nájsť postupy, ktorými by sme proces priraďovania domén mohli zjednodušiť.

1.2 Unifikácia atribútov

Druhým základným problémom je zjednotenie atribútov. Jedným príkladom môžu byť rovnaké atribúty s rôznymi označeniami pri rovnakej doméne. Napríklad tablet môže mať v jednom internetovom obchode atribúty „*typ procesora, rozlíšenie displeja, operačný systém*“, v druhom obchode to môže byť „*processor, display, operating system*“, a v treťom „*procesor, displej, OS*“. Je teda potrebné aby program vedel klasifikovať atribúty „*OS, operating system a operačný systém*“ do jedného atribútu, ktorého názov je defaultne daný. Ďalším problémom môžu byť rovnaké atribúty pri produktoch rôznych domén, napríklad spotreba energie, farba, alebo cena sa nachádzajú pri veľkom množstve produktov rôznych domén. Pre zjednodušenie práce s dátami je tiež potrebné čo najpresnejšie identifikovať dátový typ hodnoty atribútu. Dátové typy atribútov sú v súčasnej štruktúre definované väčšinou ako reťazec (*string*), no pre jednoduchšiu prácu s dátami je užitočné ich definovať čo najpresnejšie. Je pomerne jednoduché definovať dátový typ atribútu cena, no komplikovanejšie je to napríklad s atribútmi, kde je určená aj číselná aj slovná hodnota (napríklad pre rozlíšenie je to číslo a jednotka „px“, ktorá môže a nemusí byť na konkrétnom zdroji uvedená).

Príklad: Predpokladajme, že máme dané dve relácie R a S a že po extrahovaní majú tie dáta štruktúrovanú formu. Každá relácia bola extrahovaná z rozdielneho zdroja. Cieľom je spojiť tieto dve relácie, keďže sa jedná o rovnaký produkt, tak aby sme získali jednotné (unifikované) atribúty.

Tabuľka 3: relácia R

ID	Názov	Doména	Cena	K dispozícii rozlíšenie	Externý blesk	...
1	Canon EOS 20D	Fotoaparát	20,00	18Mpix	áno	...

Tabuľka 4: relácia S

ID	Názov	Doména	Cena bez DPH	Rozlíšenie	Závit na statív	...
1	Canon EOS 20D	Fotoaparát	17,00	18 megapixelov	áno	...

Jednoduchým spojením relácii by sme získali tabuľku, ktorá by obsahovala skoro dva krát viac atribútov, ktorých význam by bol rovnaký, čo je v rozpore s naši cieľom. Keďže sa jedná o rovnaký produkt z dvoch rôznych zdrojov, tak je vhodné zvážiť, či budeme výslednú reláciu reprezentovať ako dva záznamy závislé od zdroja, alebo ako jeden záznam. Problémom je, že nie všetky hodnoty atribútov sa rovnajú. Napríklad atribút „cena“, alebo atribút „cena bez DPH“ predstavuje síce rovnaký, alebo podobný atribút, no ich hodnoty sú rôzne. Bolo by teda vhodné uchovávať cenu produktu ako informáciu závislú od zdroja. Rovnako napríklad atribúty „farba“, „spotreba“ a mnoho ďalších. Takéto atribúty si teda môžeme označiť ako *zdrojovo nezávislé*. Ako som spomínal ďalším problémom sú síce rovnaké atribúty, ale s rozdielnou reprezentáciou reťazca. Atribúty „*k dispozícii rozlíšenie*“ a „*rozlíšenie*“ majú rozdielnu reprezentáciu, napriek tomu, že sú rovnaké. Ak predpokladáme že použijeme funkciu, ktorá prehlási tieto dva reťazce za totožné tak sa dostávame ku ďalšiemu problému. Tým je rozdielna reprezentácia hodnoty atribútu. Keďže predpokladáme, že sa jedná o rovnaký produkt, tak rovnako môžeme predpokladať, že hodnoty rovnakých atribútov sú totožné (platí len v prípade, že atribúty sme neprehlásili za *zdrojovo nezávislé*). Takýto predpoklad nám v tomto prípade zjednoduší prácu. Jedným zo spôsobov reprezentácie by teda mohol byť ten, že v takomto prípade vezmem jeden z reťazcov reprezentujúcich atribút a jeden z dvojice reprezentujúcich hodnotu atribútu. Takýto postup by však nebol bezchybný, keďže závisí od výberu porovnávacej funkcie a rovnako od samotných porovnávaných reťazcov. Napríklad už spomínané atribúty „*OS*“ a „*operačný systém*“ by boli

prehlásené za rozdielne. Rovnako hodnota tohto atribútu nemusí byť rovnaká (OS nie je závislí od produktu). Zistili sme teda, že pri unifikácii zdrojovo nezávislých atribútov je veľké množstvo okrajových prípadov. Našim cieľom je však nájsť proces, ktorým chceme takéto chyby minimalizovať. Vráťme sa teda ku problému, či je lepšie reprezentovať dvojicu relácii ako jeden záznam, alebo ako dva záznamy, ktoré majú rozdielny zdroj. Ak by sme chceli mať len jeden záznam pre jeden produkt, tak je potrebné unifikovať názvy atribútov a zároveň aj hodnoty atribútov. Je však rozumné zasahovať do hodnôt pri takom množstve okrajových prípadov? Pravdepodobne to závisí od konkrétneho procesu zjednotenia hodnôt atribútov. Aby sme však predišli znehodnoteniu dát, je rozumnejšie zachovať obidva záznamy pod zjednotenými atribútmi pre rôzne zdroje. Tým sme problém spájania relácii R a S zredukovali na zjednotenie atribútov.

2 Identifikácia

V predošlej kapitole sme už popisovali základnú problematiku unifikácie domén a atribútov. Pri unifikácii je však potrebné v prvom kroku identifikovať doménu alebo atribút, s ktorým následne unifikujeme. Pri veľkom množstve dát môže byť tento krok pomerne náročný. Cieľom je teda ponúknuť používateľovi potenciálne podobnú doménu alebo atribút uložený v systéme. Podľa akých kritérií však efektívne vytvárať zoznam domén alebo atribútov na unifikáciu? Problém unifikácie by sme v prvom kroku mohli rozdeliť na dva podproblémi. Prvým je identifikácia podobných domén a atribútov a druhým je samotná unifikácia. Pri identifikácií je však predpoklad, že systém už obsahuje nejaké dáta, ktoré je možné prehlásiť za potenciálne zhodné. Je preto potrebné anotovať extrahované dáta. Je zrejmé, že ak systém neobsahuje žiadne dáta, tak nie je s čím unifikovať.

2.1 Anotácia metadát

V automatizovanom spracovávaní dát sa doména priradí buď na základe zhody z už existujúcou doménou alebo sa vytvorí nová doména. Takéto priradenie však slúži len na zjednodušenie práce. Preto je potrebná finálna anotácia domény. Po anotácii domén máme identifikovanú doménu pod ktorú dané produkty spadajú. Hodnoty atribútov a názvy atribútov však ešte nemáme anotované. Cieľom anotácie je charakterizovať, alebo anotovať, vlastnosti konkrétneho atribútu pre danú doménu. Automatizovaný

proces anotácie nám však môže „*predanotovať*“ atribút. To znamená, že ak systém už obsahuje daný názov pre atribút (má totožný reťazec v názve), tak mu pridelí identifikátor. Tieto atribúty však ešte nemôžeme prehlásiť za finálne, preto ich budeme označovať za atribúty identifikované wraperom. Ako som spomínal pri doménach, anotácia danej domény predstavovala vyplnenie jej popisu (v slovenskom aj anglickom jazyku). Anotácia atribútov je o niečo zložitejšia. Budeme však uvažovať dva prípady, ktoré môžu pri anotácii nastať. Prvým je, že ideme anotovať atribút, ktorý je nový. To znamená že neexistuje žiadny atribút v štruktúre dát, ktorý by sme využili pri unifikácii. Dôsledkom je to, že bez ohľadu na prácnosť anotácie musíme popísať všetky jeho vlastnosti. Druhý prípad nastáva vtedy, keď anotujeme atribút, ktorý sa už nachádza v štruktúre dát. Presnejšie povedané v štruktúre sa už nachádza atribút, ktorý je podobný anotovanému atribútu a teda je použiť vlastnosti tohto atribútu. Používateľovi boli teda ponúknuté dáta pre anotáciu a následne atribút môže prehlásiť anotovaný. V prípade, že systém neponúkol korektné dáta, je potrebné ich upraviť pred ukončením anotácie.

Administrátorský systém taktiež ponúka možnosť prehliadania doménovo nezávislých atribútov, ktoré sa už nachádzajú v štruktúre dát. Tie sú kategorizované podľa domén do ktorých spadajú. Požívateľ je teda schopný prezerať vlastnosti atribútov podobných domén. Takéto prehliadanie dát však slúži len ako pomoc pri anotácií.. Cieľom je aby systém vo finálnej podobe čo najviac uľahčoval prácu používateľovi pri anotácii domén a atribútov. Zjednodušene povedané chceme znížiť počet klikov na minimum.

2.2 Identifikácia domén produktov

V súčasnom procese sa doména priraďuje na základe už existujúcej domény, alebo sa vytvorí nová doména. Takýto systém však nie je dokonalý. Ak v štruktúre domén uvažujeme ich hierarchiu, tak systém napríklad nie je schopný zistiť, že doména „*notebook*“ je potomok domény „*elektronika*“ (ak takáto vlastnosť nebola v minulosti zadefinovaná používateľom). Rovnako systém nie je schopný zistiť to, že ak nám zo vstupnej tabuľky prídu produkty domény „*PC*“ a zároveň štruktúra obsahuje doménu s názvom „*počítače*“, tak neviem, že sa jedná o rovnakú doménu. V takomto prípade je možné v systéme vytvoriť novú doménu s názvom „*PC*“ (pokiaľ sa používateľ nerozhodne, že chce domény unifikovať). Ak predpokladáme, že v štruktúre dát je veľké množstvo domén, tak je pomerne prácne a zdĺhavé manuálne prechádzať domény

a hľadať zhodnú doménu. Bol by teda užitočný systém, ktorý automaticky prezrie domény v systéme a ponúkne používateľovi domény, ktoré sú potenciálne zhodné s naším vstupom. Ako ale identifikovať podobné domény? Jedným zo spôsobov je, ako som spomínal, porovnanie názvov dvoch domén. Tento prístup by však bol pomerne neefektívny napríklad pri našej doméne „*PC*“. Ďalším problémom pri tomto prístupe sú cudzojazyčné stránky. Ak vyextrahujeme stránku napríklad v angličtine, tak je zrejmé, že pre doménu „*computer*“ neidentifikujeme správnu doménu (ak predpokladáme, že v systéme už máme doménu „*počítače*“). Bolo by teda vhodné použiť univerzálnejší prístup identifikácie domén. Jednou z možností by mohlo byť hľadanie podobností na úrovni metadát pomocou fulltextového vyhľadávanie. Metadáta domény predstavujú konkrétne atribúty produktov a ich hodnoty. Teda samotné produkty z danej domény a daného zdroja. Otázkou je, aké metadáta použiť alebo aké metadáta majú pre nás dostatočnú informačnú hodnotu pri identifikácií podobných domén. Asi je zrejmé, že rovnaké produkty môžu mať rôzne hodnoty v rovnakých atribútoch z niekoľkých dôvodov. Jedným je napríklad fakt, že produkty z rôznych zdrojov majú rôzne reprezentácie pre hodnoty atribútov, alebo sú reťazce chybne extrahované. Pri názvoch atribútov nastáva podobný problém. Predpokladáme totiž, že produkty domény, ktorá je v systéme, má anotované atribúty. Ich názvy teda môžu byť rôzne. Rovnako nevieme nájsť všeobecný identifikátor v metadátach, ktorý by nám pomohol pri identifikácií domén. Napriek tomu je však predpoklad, že náš systém by pomocou fulltextového vyhľadávania za určitých podmienok vedel identifikovať správnu doménu pri porovnávaní názvov, alebo hodnôt atribútov.

Príklad: Predpokladajme, že máme dve relácie R a S. Relácia R predstavuje jeden vyextrahovaný produkt z ľubovoľného zdroja. Relácia S predstavuje produkt v anotovanej doméne s anotovanými atribútmi z rozdielneho zdroja ako produkt relácie R. Chceli by sme, aby nám systém z množiny anotovaných domén ponúkol doménu *Fotoaparát* ako vhodného kandidáta na unifikáciu.

Tabuľka 5: relácia R

ID	Názov	doména	Cena bez DPH	K dispozícií Rozlíšenie	Externý blesk	...
4	Canon EOS 20D	Elektronika	20,00	18Mpix	áno	...

Tabuľka 6: relácia S

ID	Názov	Doména	Cena	Rozlíšenie	Závit na statív	...
8	Canon EOS 20D	Fotoaparát	17,00	18 Mpix	áno	...

V príklade je možné vidieť, že produkty majú totožné názvy (teda sa jedná o ten istý produkt), napriek tomu nie je možné prehlásiť oba produkty za totožné, pretože majú rozdielne metadáta. Ak by sme však v systéme hľadali podobné produkty ako produkt z neanotovanej domény, je pravdepodobnosť, že systém nájde práve produkt z tej domény, ktorú potrebujeme. Je pravdepodobné, že systém bude identifikovať podobné domény, len v prípade, že už boli anotované domény podobných produktov. V opačnom prípade nám systém, hľadajúci podobností v metadátach, ponúkne zlú alebo žiadnu doménu.

2.3 Identifikácia atribútov produktov

Rovnako ak pri doménach, aj pri atribútoch je vhodné identifikovať potenciálne podobné atribúty. Pri anotácií atribútov je to dokonca potrebnejšie keďže jedna doména môže mať veľké množstvo atribútov a samotná anotácia je podstatne náročnejšia a prácnejšia ako anotácia domén. Na rozdiel od domén pri atribútoch neuvažujeme žiadnu hierarchiu. Napriek tomu je identifikácia atribútov, na rozdiel od domén, o niečo náročnejšia. Ak zvolíme prístup jednoduchého porovnania reťazcov názvov atribútov, tak sa môže stať, že pri veľkom množstve anotovaných dát bude tento proces trvať dlhšie. Pre urýchlenie môžeme prehľadávať len atribúty z danej domény, no takýto spôsob je neefektívny, keďže daný atribút sa môže nachádzať v nesúvisiacej doméne. Zároveň sa pri tomto prístupe môže vyskytnúť rovnaký problém ako pri doménach. Ak anotujeme atribút z názvom „*OS*“ a zároveň máme v systéme atribút s názvom „*operačný systém*“ (pričom predpokladáme, že ich hodnoty sa líšia) tak nám systém neponúkne to najlepšie čo môže. Skúsme teda uvažovať prístup analogický s identifikáciou domén, teda fulltextové vyhľadávanie. Budeme teda hľadať produkty, ktoré sú podobné ako náš atribút. Problémom je, že pri doméne sme mohli hľadať konkrétny produkt, pričom pri atribútoch hľadáme len produkty podobné ako náš jeden atribút. Pri takomto hľadaní nastáva problém, kedy prehlásiť produkt za podobný. Predpokladajme, že na vstupe máme názov a hodnotu atribútu. Ak však atribút

nadobúda len jednu hodnotu, tak môžeme dostať veľké množstvo zlých výsledkov. Mohli by sme však využiť fakt, že po extrahovaní produktov z danej domény máme pre jeden atribút minimálne jednu hodnotu. Pri hľadaní podobných produktov môžeme teda použiť všetky hodnoty, ktoré atribút nadobúda a hľadať podobnosti na základe hodnôt.

Príklad: Podobne ako pri identifikácií domén máme daný produkt s anotovanými atribútmi v anotovanej doméne (relácia R). Zároveň máme atribút, ktorý nadobúda množinu hodnôt (relácia S). Cieľom je nájsť potenciálne podobné atribúty.

Tabuľka 7: relácia R

ID	Názov	Farba	Cena	Rozlíšenie displeja	Externý blesk	...
4	Canon EOS 20D	čierna	20,00 €	18Mpx	áno	...

Tabuľka 8: relácia S

ID	Názov	Hodnota1	Hodnota2	Hodnota3	Hodnota4	...
8	rozlíšenie	5Mpx	600x800	18 Mpx	1024x600	...

Ako je vidieť v relácii S, vytvorili sme množinu hodnôt, ktoré atribút nadobúda a hľadáme produkty, ktoré obsahujú daný atribút, alebo jednu z hodnôt v niektorom atribúte. Predpokladáme, že systém vráti všetky atribúty z daného produktu, v ktorých nastala zhoda. Týmto spôsobom je však potrebné prechádzať všetky atribúty a všetky hodnoty atribútov. Problém rýchleho vyhľadávanie vo veľkom množstve dát je riešený takzvaným fulltextovým vyhľadávaním. Rovnako ako pri doménach aj pri atribútoch je pravdepodobné, že systém bude identifikovať podobné atribúty, len v prípade, že už boli anotované atribúty podobných produktov. V opačnom prípade nám systém ponúkne nesprávny alebo žiadny atribút.

3 Fulltextové vyhľadávanie

Fulltextové vyhľadávanie je spôsob vyhľadávania vo vopred pripravených databázach alebo textových súboroch (dokumentoch), aby bolo možné nájsť ľubovoľný reťazec znakov v čo najkratšom možnom čase. Ak chceme nájsť reťazec v malom množstve dokumentov, tak stačí aby sme prehľadali množinu dokumentov

a porovnávali hľadaný reťazec s časťou konkrétneho dokumentu. V prípade, že je množina dokumentov príliš rozsiahla, je vhodné použiť spôsob vyhľadávania, ktorým sa, čo možno najviac, urýchli časová odozva pri vyhľadávaní.

Pri fulltextovom vyhľadávaní vyhľadávací algoritmus porovnáva vstup so všetkými reťazcami v uložených dokumentoch a pokúša sa vyhodnotiť ich zhodu. Takéto vyhľadávanie sa bežne používalo už v sedemdesiatych rokov napríklad na vyhľadávanie v databáze kníh. Rovnako v súčasnosti je tento prístup vyhľadávania využívaný vo veľkom množstve webových stránok a aplikácii. Pravdepodobne každý kto využíva internet sa stretol s fulltextovým vyhľadávačom, ktorý umožňuje používateľovi klásť „otázky“ bez ohľadu na to v akom morfologickom tvare ho zadá. V praxi je ťažké rozoznať ako konkrétny vyhľadávací algoritmus pracuje, keďže sa tieto algoritmy len málokedy zverejňujú.

Ako som spomínal, pri malom množstve dokumentov je možné vyhľadávať reťazec priamo. Pokiaľ je však množstvo dokumentov podstatne väčšie, ako kapacita vyhľadávacieho algoritmu, je pre udržanie rýchlej časovej odozvy dôležité rozdeliť vyhľadávanie na dve časti: indexovanie a vyhľadávanie. Pri indexovaní sa spracováva text všetkých dokumentov, pričom je vytváraný takzvaný „index“. Index predstavuje zoznam kľúčových slov, alebo kľúčových dát (nie nutne len reťazce), v ktorých sa následne vo vyhľadávacej fáze prechádza pripravený index, namiesto originálnych dokumentov. Indexer vytvára index pre každé slovo, alebo reťazec v dokumente, pričom obvykle ignoruje tzv. „*stop-words*“, ktoré nemajú pri vyhľadávaní žiadnu informačnú hodnotu. Obvykle sú to spojky a predložky. V našom prípade tieto slová neuvažujeme, pretože tieto slová boli odfiltrované nástrojom na extrakciu produktov (respektíve boli extrahované len užitočné informácie o produktoch) Niektoré indexery prevádzajú reťazce v dokumentoch na „*korene slov*“, čiže nepracujú so slovami v odvodenom tvare. Na takúto deriváciu slov pritom využívajú skloňovanie alebo časovanie slov. Pri takejto redukcii je však potrebná rozsiahla lexikálna databáza slov, čo môže podstatne spomaliť proces indexovania. Vo všeobecnosti fulltextové vyhľadávače vždy odrážajú aktuálny stav dokumentov (taký, aký bol v čase posledného indexovania). Rôzne vyhľadávače aktualizujú svoj index v rôznych časových intervaloch. Samotné indexovanie je však časovo náročné pri veľkom množstve dát. Preto je potrebné zvážiť, kedy aktualizovať index, aby bol systém efektívny a ponúkol nám vždy to najlepšie, čo sa dá. V našom prípade indexujeme produkty internetových

obchodov. Aj keď je týchto obchodov veľké množstvo, predpokladáme že veľké množstvo produktov ponúka niekoľko obchodov. Tým pádom sa počet indexovaných produktov nerovná počtu extrahovaných produktov. Zároveň jeden indexovaný produkt neobsahuje rozsiahle reťazce informácií (ako napríklad webová stránka indexovaná webovým vyhľadávačom). Preto predpokladáme, že indexovania našich dát nebude trvať extrémne dlho (napríklad niekoľko dní). Dôležitou vlastnosťou fulltextového vyhľadávača je presnosť a návratnosť. Návratnosť je miera kvantity relevantných dát získaných vyhľadávačom, kým presnosť je miera kvality vrátených výsledkov. Návratnosť vieme získať ako podiel relevantných výsledkov vrátených vyhľadávačom a všetkých relevantných výsledkov v indexe. Presnosť získame podielom relevantných výsledkov získaných vyhľadávačom a všetkých výsledkov vrátených vyhľadávačom. Pomocou týchto informácií by sme mohli získať informáciu o efektívnosti fulltextového vyhľadávača. Problémom však je, že často nemáme vedomosť o tom aké dáta boli indexované, alebo či dáta vrátené vyhľadávačom sú práve tie, ktoré potrebujeme. Pri našich dátach však môžeme predpokladať, že nevzniknú problémy, alebo chyby pri vyhľadávaní, ktoré sú spôsobené nejednoznačnosťou prirodzeného jazyka.

3.1 Elasticsearch

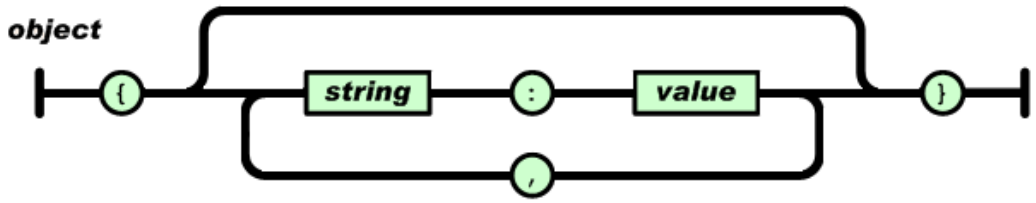
Program elasticsearch ponúka veľké množstvo funkcií, pomocou ktorých vieme vyhľadávať alebo filtrovať dokumenty, ktoré sú pre nás zaujímavé alebo podobné s našim dokumentom na vstupe. Všetky dopyty a filtre sú definované pre formát JSON. Samotný program elasticsearch neobsahuje vlastné algoritmy podporujúce prácu s dátami, ale využíva existujúcu knižnicu *„Apache Lucene Core“*, ktorá túto funkcionalitu podporuje. Táto knižnica je súčasťou open-source projektu zameraného na vyhľadávanie. Knižnica Lucene podporuje uchovávanie, indexovanie, vyhľadávanie a spravovanie dát. Môžeme sa teda spýtať prečo by sme mali používať elasticsearch, keď prácu s dátami vykonáva samotná knižnica. Hlavným dôvodom používania programu elasticsearch je ten, že ponúka lepšiu funkcionalitu pre tvorbu webových aplikácií, napríklad http protokol, čiže sa dá využiť pri rôznych platformách. Taktiež poskytuje spätnú kompatibilitu.

Na to aby sme mohli indexovať dáta do jednotlivých indexov, je potrebné ich previesť do JSON formátu. JSON (JavaScript Object Notation) je pomerne jednoduchý dátový formát. To znamená že je ľahko čitateľný pre človeka a rovnako je pomerne

ľahké ho generovať a analyzovať. JSON je textový formát, ktorý je úplne nezávislý, no používa konvencie, ktoré sú zrozumiteľné. Tieto vlastnosti robia z formátu JSON ideálny prostriedok na výmenu dát medzi programátorskými jazykmi. Tento formát sa skladá z dvoch základných štruktúr:

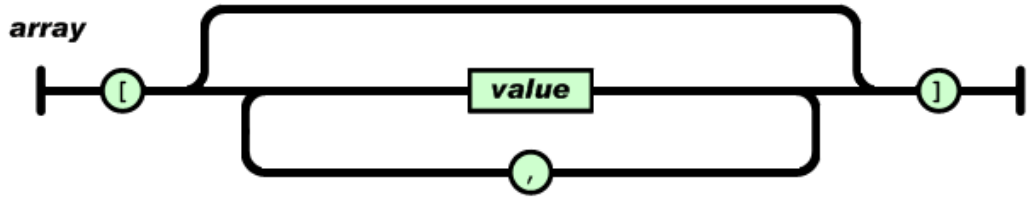
- Množina párov názov/hodnota. V rôznych jazykoch je táto množina reprezentovaná buď ako objekt, záznam, štruktúra alebo asociatívne pole.
- Usporiadaný zoznam hodnôt. Vo väčšine jazykoch reprezentovaný ako pole, vektor, list alebo sekvencia.

Objekt predstavuje neusporiadanú množinu dvojíc názov/hodnota. Začiatok a koniec objektu je reprezentovaný zátvorkami { }, názov a hodnota sú oddelené dvojbodkou a jednotlivé páry sú oddelené čiarkou.



Obr. 1 Schéma objektu JSON

Pole, vektor, list alebo sekvencia (podľa reprezentácie v danom jazyku) je usporiadaná množina hodnôt oddelených čiarkou.



Obr. 2 Schéma poľa objektu JSON

Okrem poľa je možné do objektu vkladať aj hodnoty iných typov. Napríklad reťazec, číslo, iný objekt, boolean hodnoty (*true, false*), alebo hodnotu *null*. Hodnota typu reťazec je sekvencia Unicode znakov obalených úvodzovkami.

3.1.1 Návrh štruktúry JSON objektov pre indexovanie

V predchádzajúcej kapitole bol popísaný princíp fulltextového vyhľadávania. Pomocou programu Elasticsearch budeme indexovať dáta a následne v týchto dátach vyhľadávať alebo identifikovať potenciálne podobné metadáta. V prvom kroku je teda

potrebné indexovať dáta. Ako som spomínal program elasticsearch pracuje s dátami vo formáte JSON. Je teda potrebné všetky dáta, ktoré budeme indexovať a vstupy pre dopyty, previesť do tohto formátu. Pred samotným indexovaním si musíme uvedomiť aké dáta budeme indexovať a hlavne čo má daný indexovaný dokument obsahovať. V kapitole identifikácia domén a atribútov som popísal základný princíp vyhľadávania. Budeme teda indexovať jednotlivé extrahované produkty, pričom doména a atribúty týchto produktov už musia byť anotované. Následne budeme v týchto anotovaných a indexovaných dátach hľadať podobné produkty s naším vstupom. V súčasnej štruktúre dát sú jednotlivé produkty uložené v tabuľkách podľa domén do ktorých patria. Tieto dáta z tabuliek budeme prevádzať do formátu JSON. Jeden produkt v tomto formáte bude predstavovať jeden JSON dokument. Čo všetko však takýto dokument musí obsahovať? Ako výstup pri vyhľadávaní, alebo výsledok dopytu, dostaneme jednotlivé, potenciálne podobné, dokumenty. Mali by sme byť teda schopný s tohto výstupu vyextrahovať to čo potrebujeme, teda doménu a atribút. Dokument by teda mal obsahovať doménu, do ktorej produkt patrí a zároveň by mal obsahovať všetky dvojice atribút a hodnota, ktoré konkrétny produkt nadobúda. Ako som spomínal, problémom pri vyhľadávaní sú napríklad cudzojazyčné stránky. Tento problém je možné vyriešiť tak, že ak nájdeme atribút, ktorý má v štruktúre zároveň cudzojazyčný názov, tak tento názov taktiež vložíme do JSON dokumentu. Hodnota takéhoto atribútu pritom predstavuje rovnakú hodnotu ako pri pôvodnom názve. Je zrejmé, že takýto systém správne definuje nový cudzojazyčný atribút alebo doménu len v prípade, že takýto názov už bol v minulosti definovaný alebo anotovaný.

Predpokladajme teraz, že máme v systéme indexované dáta, nad ktorými vieme vyhľadávať. Keď anotujeme nové dáta je potrebné ich znova indexovať. Chceme pritom predísť duplicitám v indexovaných dátach. Pre správne fungovanie systému na identifikáciu dát, je problémom ak sú v systéme indexované rovnaké produkty a to aj za predpokladu, že sú produkty z rôzneho zdroja. Aj keď sa takéto produkty môžu líšiť v názvoch atribútov a v reprezentácií hodnôt atribútov, samotná štruktúra dát projektu takéto duplicity nepovoľuje. Jeden produkt, ktorý je vyextrahovaný z dvoch rôznych zdrojov je uchovávaný ako jeden produkt, pričom jeho atribúty a hodnoty sú zjednotené. Ak sa teda takýto produkt zmení, je anotovaný nový atribút, je potrebné ho z indexu vymazať a následne ho znova indexovať. Takéto mazanie je možné vďaka jedinečnému identifikátoru, ktorý je produktu pridelený systémom uchovávajúcim tieto dáta. Otázkou ostáva kedy tieto dáta obnoviť. Jednou z možností je obnoviť dáta vždy

keď sa zmení ľubovoľný atribút, alebo druhou možnosťou je obnoviť indexované dáta po súhlase administrátora.

Poďme sa teda pozrieť na konkrétny príklad indexovaného produktu. Objekt JSON dokumentu a samotnú štruktúru a tvar takéhoto objektu som už spomínal v predošlej kapitole. Pre pripomenutie sú to dvojice *atribút:hodnota*, pričom hodnota môže predstavovať aj množinu hodnôt. JSON objekt (jeden produkt) pre doménu bicykle vyzerá nasledovne:

```
{
"domena":"Bicykle",
"idDomain":2,
"id_object":"40964",
"id_attributes":[29,30,31,32,33,34],
"Cena":"620",
"Price":"620",
"Rám": hiten 26“,
"Frame": hiten 26“
"Brzdy": V Brzdy,
"Prešmýkač": SHIMANO TY18 (28.6),
"Menič": SHIMANO TY18,
"Radenie": SHIMANO Revoshift (18)
}
```

Ako môžeme vidieť objekt obsahuje informáciu o doméne, identifikátor produktu, identifikátory jednotlivých atribútov a samotné atribúty aj v cudzojazyčných názvoch. Máme teda definovanú štruktúru JSON objektu pre indexovanie produktov. Ako ale definovať objekt pre vyhľadávanie? Základná myšlienka bola spomenutá pri identifikácií domén a atribútov. Chceme teda nájsť podobné produkty ako náš vstup.

4 Funkcie pre dopyt fulltextového vyhľadávania

Základnú funkcionalitu fulltextového vyhľadávania v programe Elasticsearch by sme mohli rozdeliť na dve kategórie. Prvou sú dopyty pomocou ktorých vieme vyhľadávať indexované dáta a druhou sú filtre, ktoré sú podobné ako dopyty, no ponúkajú len obmedzenú funkcionalitu oproti dopytom.

4.1 Filtre a dopyty

Filtre je možné použiť ako náhradu dopytu v prípade že potrebujeme len vyfiltrovať podobné dokumenty. Jedných zo základných rozdielov medzi dopytom a filtrom je, že filter neráta skóre. Skóre je číselná miera podobnosti medzi hľadaným dokumentom a indexovaným dokumentom. Dôsledkom toho je, že filtre sú podstatne rýchlejšie ako dopyty. Výstup filtra je teda len množina podobných dokumentov, ktoré spĺňajú zadanú požiadavku. Sú používané ako vnorené funkcie pre dopyty. To znamená, že do dopytu, ktorý použijeme, vnoríme filter. Ten nám vráti len množinu dokumentov, ktoré spĺňajú našu požiadavku (napr. aby pod názvom *doména* bola hodnota *fotoaparát*). Výsledky filtra sú zároveň uložené v pamäti. Ak teda používame filter v dopyte viac ako jeden krát, sú použité dáta z pamäte. Fakt, že filtre sú rýchlejšie, a to, že výstup je uchovávaný v pamäti, je dôsledkom, že vyhľadávanie je podstatne rýchlejšie. Program obsahuje veľké množstvo filtrov, ktoré je možné použiť. Ukážeme si dva základné filtre *term filter* a *range filter*, ktoré je možné napísať priamo v JSON formáte, alebo pomocou Java API pre elasticsearch.

```
{
"filter" : {
        "term" : { "doména" : "fotoaparát"},
        }
}

{
"filter" : {
        "range" : {
                "megapixel" : {
                        "gte" : 5,
                        "lte" : 8
                        }
                }
        }
}
```

Dopyty by sme mohli rozdeliť na základné a komplexné. Základné dopyty sa vzťahujú na konkrétnu hodnotu (resp. množinu slov) pre daný názov v objekte JSON. Pri tomto dopyte teda hľadáme konkrétne hodnoty definované pre jeden atribút JSON objektu. Komplexné dopyty je možné použiť pri vyhľadávaní, kde na vstupe máme

neznámy dokument a snažíme sa nájsť podobné bez toho, aby sme definovali konkrétnejšie atribúty, ktoré by mali indexované dokumenty obsahovať (napríklad pri dopyte *more_like_this*). Nemusíme teda explicitne dopytu povedať, v ktorých atribútoch objektu vyhľadávať. V tomto prípade dopyt vyhľadáva vo všetkých atribútoch, ktoré sa nachádzajú v indexovaných dokumentoch. Je možné taktiež vyskladať zložené dopyty zo základných dopytov, alebo vytvoriť dopyt s niekoľkými vnorenými dopytmi alebo filtrami podľa našich potrieb.

Príklad: Predpokladajme, že chceme nájsť všetky také dokumenty, ktoré spĺňajú nasledujúce podmienky: v dvojici z názvom *doména* bola hodnota *fotoaparát,* v názve *megapixel* boli hodnoty od *5* do *8* a v názve *pamäťová karta* bola hodnota *áno.* Zároveň chceme aby bola prvá a druhá podmienka splnená a tretia podmienka môže a nemusí byť splnená. Zložený dopyt pre túto požiadavku by sme teda mohli vyskladať zo základných dopytov nasledovne:

```
{
"query" : {
"bool" : {
        "must" : {
                "term" : { "doména" : "fotoaparát"},
        "must" : {
                "filter" : {
                        "range" : {
                                "megapixel" : {
                                        "gte" : 5,
                                        "lte" : 8
                                        }
                                }
                        }
                }
         }
         "should": {
                "term": { "pamäťová karta" : "áno"}
     }
   }
}
}
```

Môžeme vidieť, že pre názvy *doména* a *pamäťová karta* sme použili základný vnorený dopyt *termQuery* (je možné použiť aj *termFilter*) a pri názve megapixel sme použili *rangeFilter*. Ďalej môžeme vidieť, že celý dopyt je obalený pomocou

BooleanQuery dopytu. Celý dopyt nám teda vráti všetky také dokumenty, ktoré spĺňajú zadané požiadavky, pričom sú zoradené podľa skóre od najpodobnejších po najmenej podobné. Takýmto spôsobom je možné vyskladať zložené dopyty pomocou základných dopytov. Čo však v prípade, že nechceme nájsť konkrétne dokumenty na základe podmienok, ale chceme nájsť všetky také dokumenty, ktoré sú podobné s hodnotami, ktoré nemajú explicitne definovaný atribút. V tomto prípade je možné použiť jeden z dopytov *more-like-this* alebo *fuzzy-like-this*. Tieto dopyty majú na vstupe množinu hodnôt a nájdu dokumenty „približne" rovnaké s hľadaným vstupom, pričom ako som spomínal, nemusíme definovať atribúty (ale môžeme).

4.1.1 More-like-this dopyt

Tento dopyt podporuje aj vyhľadávanie v poliach (množinách hodnôt pre daný názov). Dopyt je v skutočnosti vyskladaná *BooleanQuery* funkcia, ktorá je použitá na konkrétne názvy a hodnoty. Tieto názvy a hodnoty sú však vybrané na základe *tf-idf* jednotlivých hodnôt (čiže nie všetky hodnoty zo vstupného dokumentu sú brané do úvahy pri vyhľadávaní). Taktiež je možné zadefinovať rôzne parametre vyhľadávania ako napríklad minimálny a maximálny počet výskytov danej hodnoty, minimálny a maximálny počet dokumentov, v ktorých sa daná hodnota musí vyskytnúť (pri ktorých bude hodnota braná do úvahy pri vyhľadávaní), maximálna a minimálna dĺžka slov, ktoré budú porovnávané. Tiež sa dá zadefinovať percentuálny počet klauzúl v dopyte, ktoré musia byť splnené, aby bol dokument prehlásený za podobný.

Pre použitie dopytu by však bolo vhodné bližšie popísať princíp fungovania. Na základe vedomostí o fungovaní konkrétneho dopytu by sme následne mohli definovať prípady, kedy je konkrétny dopyt vhodné použiť, prípadne kedy je použite menej efektívne. Princíp fungovania je rovnako užitočné poznať pri definovaní parametrov dopytu, ktoré boli spomínané. Poďme sa teda pozrieť, čo a ako vlastne dopyt *more_like_this* ráta a využíva. Ako som spomínal, tento dopyt zo vstupu vyberie len niektoré hodnoty. Pointa dopytu je teda vo výbere vhodných hodnôt atribútov a priradení vhodných názvov atribútov týmto hodnotám. Tie budú použité pri samotnom vyhľadávaní a rátaní skóre s indexovanými dokumentmi. Uvažujme, že hľadáme všetky dokumenty, ktoré sú podobné s našim vstupom:

```
{
	"doména" : "fotoaparát",
	"výrobca" : "Nikon",
	"rozlíšenie" : "24.2 Mpix",
	"hnotnosť" : "505 g",
	"popis" : "Jednooká zrkadlovka Nikon D3200 je ideálny pre začiatočníkov s jednoduchým ovládaním. Tento model je vybavený režimom Sprievodca najnovšej generácie s podporou videa. S fotoaparátom Nikon D3200 zachytíte verne atmosféru všetkých dôležitých okamihov. "
}
```

Máme teda definovaných 5 atribútov a ich hodnoty. Predpokladáme však, že hľadáme podobné produkty na základe podobností hodnôt atribútov. Zároveň, pre tento dopyt nie je nutné definovať názvy atribútov. Vstupom teda budú len hodnoty a názvy atribútov môžeme odfiltrovať. Vstup pre dopyt teda môže vyzerať nasledovne:

```
fotoaparát, Nikon, 24.2 Mpix, 505 g, Jednooká zrkadlovka Nikon D3200 je ideálny pre začiatočníkov s jednoduchým ovládaním. Tento model je vybavený režimom Sprievodca najnovšej generácie s podporou videa. S fotoaparátom Nikon D3200 zachytíte verne atmosféru všetkých dôležitých okamihov.
```

Dopyt v prvom kroku vyberie jednotlivé hodnoty a vytvorí vektor termov (jedno slovo v hodnote označme ako term). Takýto vektor obsahuje všetky termy, ktoré sa nachádzali v hodnote atribútu a počet jeho výskytov. Vektor bude teda vyzerať nasledovne:

```
Nikon[3], D3200[2], fotoaparát[1] 24.2[1], Mpix[1], 505[1], g[1] Jednooká[1], zrkadlovka[1], ideálny[1], začiatočníkov[1], jednoduchým[1], ovládaním[1], model[1], vybavený[1], režimom[1], Sprievodca[1], najnovšej[1], generácie[1] podporou[1], videa[1], fotoaparátom[1], zachytíte[1], verne[1], atmosféru[1], dôležitých[1], okamihov[1]
```

Po vytvorení vektora, je tento prevedený na mapu, ktorá je použitá ako vstup pre metódu *createQueue*. V tejto metóde sa vykonáva základ celého dopytu. Pre každý term v mape, vytvorenej z hodnôt sa následne ráta skóre pre jednotlivé termy. V prvom kroku zistí počet všetkých dokumentov, ktoré boli indexované v danom indexe. Následne iteruje cez každý term v mape, pričom vyhodí tie, ktoré sa nevyskytli dostatočný počet krát (parameter dopytu *min_term_freq*). Potom postupne hľadá daný term v jednotlivých atribútoch indexovaných dokumentov a zisťuje v koľkých

dokumentoch sa term vyskytol. Cyklus teda neprehľadáva len počet výskytov v dokumente, ale počíta počet výskytov v každom atribúte každého indexovaného dokumentu. Napríklad term *„Nikon“* sa vyskytol v 10-tich dokumentoch v atribúte „*popis*“, ale zároveň sa vyskytol v 15-tich dokumentoch v atribúte „*výrobca*“. Metóda si teda zapamätá maximálny počet výskytov (teda *docFreq*=15) a zároveň si zapamätá atribút, v ktorom sa hodnota vyskytla. Následne filtruje slová, ktoré sa nevyskytli v dostatočnom počte dokumentov (parameter *min_doc_freq*). Čiže ak je parameter nastavený na hodnotu *5* (defaultná hodnota), maximálny počet výskytov je v atribúte „popis“ a jeho hodnota je *4*, tak tento term bude ignorovaný. V poslednom kroku metóda vyráta *idf* a *skóre* pre každý „úspešný“ term. Hodnota *idf* sa ráta z celkového počtu dokumentov a hodnoty *docFreq*. Výstupom metódy je rad (*PriorityQueue*), v ktorom sú zoradené termy podľa skóre, ktoré nadobudli. Pre jednu vstupnú mapu by teda vytvorí rad, ktorý môže vyzerať nasledovne:

Tabuľka 1: výstup metódy createQueue

	topAttribute	*Score*	*idf*	*docFreq*	*tf*
zrkadlovka	Typ fotoaparátu	2,67	0,89	17	3
nikon	výrobca	2,5	1,25	15	2
fotoaparátom	popis	0,58	0,58	9	1
...					

Tento výstup však nie je finálny. V ďalšom kroku sa ráta *boost*, ktorý predstavuje niečo ako mieru dôležitosti daného termu. Tá je však tiež parametrom dopytu, takže môže a nemusí byť braná do úvahy. *Boost* faktor sa ráta ako podiel skóre, ktoré nadobudol daný term a najlepšieho skóre, ktoré bolo nadobudnuté niektorým termom vo vstupnej mape. Nakoniec sa na základe termov, ktoré sú v rade vyskladá *BooleanQuery* pomocou *should* operátora a dopytu *TermQuery*. Výsledný dopyt pre daný vstup by preto mohol vyzerať takto:

```
{
```

```
"query" : {
"bool" : {
        "should" : {
                "term" : { "výrobca" : "nikon"},
        "should" : {
                "term" : { "typ fotoaparátu" : "zrkadlovka"},
        "should": {
                "term": { "popis" : "fotoaparátom"}

        ....


        }
        }
}
```

Na základe vedomostí o štruktúre indexovaných dát, vstupu a toho ako funguje tento dopyt by bolo vhodné pozmeniť niektoré parametre. Keďže nemáme vedomosť o tom, ktoré atribúty alebo hodnoty sú viac či menej dôležité (zaujímavé) pri vyhľadávaní, tak pravdepodobne nemá zmysel používať *boostFaktor*. Rovnako nemá zmysel definovať pole atribútov, nad ktorými budeme zisťovať *docFreq* pre daný term, keďže to je práve tá informácia o ktorú sa usilujeme. Na druhej strane by bolo vhodné pozmeniť parameter *min_term_freq* (defaultne je nastavený na hodnotu 2). Keďže indexované dokumenty a vstupné dáta neobsahujú rozsiahlejší text, je potrebné nastaviť tento parameter na hodnotu 1. V praxi by to znamenalo, že vyskladaný *BooleanQuery* bude obsahovať väčšie množstvo dopytov *TermQuery* (stále nebude obsahovať termy s nízkou frekvenciou výskytov naprieč všetkými dokumentmi). Rovnako by mohlo byť zaujímavé definovať *Stop words* pre dopyt *more_like_this.* Tie predstavujú „nezaujímavé“ termy a teda dopyt ich neberie do úvahy. Takýmito termami by mohli byť napríklad spojky.

4.1.2 Fuzzy-like-this dopyt

Rovnako nájde všetky „podobné“ dokumenty s hľadaným, avšak na rozdiel od *more-like-this* tento dopyt neberie do úvahy *tf-idf* ale *fuzzy* hodnotu. Podľa fuzzy funkcie teda vyberie *n*-najlepších hodnôt z ktorých vytvorí *BooleanQuery* funkciu. Vo všeobecnosti je tento dopyt pomalší oproti predošlému (keďže musí rátať fuzzy zhodu hodnôt).

Rovnako ako dopyt *more_like_this* ani dopyt *fuzzy_like_this* nie je podrobnejšie popísaný v dokumentácií knižnice lucene. Preto bolo potrebné sa bližšie pozrieť na zdrojové kódy tohto dopytu. Tieto informácie nám taktiež môžu pomôcť pri definovaní parametrov. Rovnako nám tieto informácie pomôžu definovať okrajové prípady, kedy je a kedy nie je vhodné použiť práve tento dopyt. Ako som spomínal dopyt vyberá zo vstupu len niekoľko najlepších termov. Tie sú určované na základe fuzzy prahu. Rovnako ako pri predchádzajúcom dopyte, aj tu sa využíva prioritný rad, ktorý obsahuje termy zoradené podľa skóre, ktoré im bolo pridelené na základe definovanej podobnostnej funkcie. Knižnica lucene využíva funkciu *edit_distance* podľa ktorej vyhodnocuje prah (fuzzy hodnotu). Pri tomto dopyte je základ dopytu implementovaný v metóde *addTerms*, ktorá pridáva termy do prioritného radu. Rovnako ako pri *more_like_this* aj tento dopyt priraďuje termom atribúty z ktorých následne vyskladá dopyt. V prvom kroku prehľadáva všetky atribúty a pre každý hľadá zhodu zo vstupom v celom indexe. Vstup sa pri tomto dopyte neredukuje, ako to bolo v predošlom dopyte. Pracujeme teda s každým termom, ktorý bol na vstupe. Najprv zistíme celkový počet dokumentov v indexe. Pre každý atribút, v ktorom ideme hľadať, vyberieme všetky termy, ktoré daný atribút nadobúda v celom indexe. Pre každý vstupný term metóda hľadá jeho *varianty* na základe fuzzy zhody. V cykle je vyrátaná fuzzy zhoda pre každý term v prehľadávanom atribúte. Tie sú následne ukladané do prioritného radu podľa toho akú zhodu nadobudli (parameter *MAX_VARIANTS_PER_TERM* udáva maximálny počet podobných termov, ktoré budú zachované). V podstate, ak je nájdená zhoda podľa *edit_distance* funkcie s prahom *0.5*, tak je daný term prehlásený za podobný a uložený do radu. Zároveň je pre každý podobný term vyrátané skóre. To sa ráta pomocou *boost* faktora, ktorý sa definuje pri indexovaní (defaultne nastavený na 1) a pomocou *idf* hodnoty ako súčin druhej mocniny *boost* faktora a *idf* hodnoty. Hodnota *idf* závisí od celkového počtu dokumentov a hodnoty *df* (počet dokumentov, v ktorých sa vstupný term vyskytol). Ak je hodnota *df* = 0 (vstupný term sa nevyskytol v indexovaných dokumentoch), tak sa *idf* počíta pomocou priemernej frekvencie výskytov podobných termov (podiel počtu podobných termov a sumy frekvencie jednotlivých podobných termov). Každý podobný term je uložený do prioritného radu s informáciou o skóre termu a vstupnom terme. Tento proces sa vykoná pre každý vstupný term a pre každý atribút v indexovaných dátach (pokiaľ nedefinujeme pole atribútov, v ktorých má vyhľadávať).

Predpokladajme teda, že pre každý vstupný term máme uložené všetky jeho podobné varianty v prioritnom rade. V ďalšej fáze je potrebné vyskladať dopyt, podľa ktorého budeme vyhľadávať v indexovaných dátach. Najprv je prioritný rad prevedený na mapu, kde kľúč predstavuje vstupný term a hodnota je zoznam fuzzy podobných termov. V poslednom kroku prehľadávam mapu a pre každý kľuč v mape sa vyskladá pomocou *BooleanQuery* a operátora *should* pre všetky hodnoty, ktoré kľúč nadobúda. Toto sa vykoná pre každý kľúč, čiže získame niekoľko dopytov. To je následne potrebné spojiť do finálneho dopytu, znova pomocou *BooleanQuery* a operátora *should*. Získame teda dopyt s niekoľkými vnorenými dopytmi.

Pri *fuzzy_like_this* dopyte je podstatne menej parametrov ako to bolo pri dopyte *more_like_this*. Napriek tomu by bolo vhodné zvážiť, aký prah (parameter *fuzziness*) použiť na to aby sme dostali to najlepšie čo sa dá z indexovaných dát. Zaujímavým parametrom je aj *prefix_length* (defaultne nastavený na hodnotu 0), ktorý predstavuje dĺžku požadovaného prefixu termu na to, aby bol prehlásený za podobný (zároveň musí spĺňať aj fuzzy prah). Takýmto parametrom by sme totiž mohli odfiltrovať len slová, ktoré sú napríklad skloňované. Vo všeobecnosti je však tento dopyt efektívny napríklad v prípade, že by dáta o produktoch internetových obchodov boli chybne extrahované.

5 Návrh štruktúry vstupných objektov pre vyhľadávanie

Pri hľadaní podobných domén a atribútov máme k dispozícií zoznam produktov, ktoré zatiaľ neboli anotované. Tieto dáta boli extrahované z rôznych zdrojov a majú štruktúrovanú formu. Sú uložené v jednej tabuľke z názvom *wrap*. Táto tabuľka predstavuje náš vstup. Pre nás podstatné stĺpce sú *attribute_value* a *attribute_name*, ktoré obsahujú názov a hodnotu atribútu pre jeden konkrétny produkt. Jeden produkt je teda uložený v niekoľkých riadkoch s rovnakým identifikátorom v stĺpci *id_product*. Každý produkt taktiež obsahuje názov domény z ktorej bol extrahovaný. Názov domény a hodnotu *attribute_value* je potrebné identifikovať s hodnotami, ktoré sme indexovali. Pomocou hodnôt vo *wrap* tabuľke budeme teda vytvárať náš vstup pre dopyt a tak identifikovať podobnosti s indexovanými dátami. Na nasledujúcom obrázku môžeme vidieť niektoré hodnoty a atribúty jedného produktu. Produkt spadá pod neanotovanú

doménu „*telefóny/mobilny-telefon*“, ktorá je v stĺpci *attribute_value* pod názvom *attribute_29.* Neanotované domény a atribúty majú v stĺpci *id_attribute* a *id_domain* priradený identifikátor s hodnotou *null.*

id_download	id_product	attribute_name	attribute_value	id_attribute	id_domain
110	3334	attribute_23	Typ OSFotoaparátVstav...	23	NULL
110	3334	attribute_24	31509	24	NULL
110	3334	attribute_26	Alcatel One Touch 2005...	26	NULL
110	3334	attribute_29	telefony/mobilny-telefon	29	NULL
110	3334	attribute_4	45.99 €	4	NULL
110	3334	Bluetooth	Áno	NULL	NULL
110	3334	Dotykový displej	nie	NULL	NULL
110	3334	EDGE	Áno	NULL	NULL
110	3334	Fotoaparát	2 Mpx	NULL	NULL
110	3334	GPRS	Áno	NULL	NULL
110	3334	GPS	Nie	NULL	NULL
110	3334	Hmotnos? vrátan...	85 g	NULL	NULL
110	3334	Kapacita batérie	850 mAh	NULL	NULL

Obr. 3wrap tabuľka

5.1 Tvorba vstupov pre identifikáciu domén

V kapitole [3.2 identifikácia domén produktov] sme popísali základnú myšlienku toho ako by mal vyzerať vstup. Každý indexovaný dokument obsahuje aj informáciu o doméne do ktorej spadá. Ako som spomínal budeme hľadať podobné produkty ako náš vstup. Základnou úlohou je však vytvoriť taký vstup, aby sme dostali tie najpodobnejšie produkty. Pri práci sme uvažovali niekoľko prístupov vytvárania vstupu pre identifikáciu domén. Predpokladajme, že chceme anotovať doménu, ktorej produkty sú uvedené ako podmnožina produktov danej domény a zdroja vo *wrap* tabuľke. Pri komplexných dopytoch sú pre nás zaujímavé len hodnoty atribútov. Aké hodnoty však vybrať, aby sme našli podobné produkty? Ak doména obsahuje napríklad 200 produktov a každý produkt obsahuje napríklad 20 atribútov, tak máme k dispozícii 4000 hodnôt. Pre urýchlenie vyhľadávania je preto vhodné nevyberať všetky hodnoty.

5.1.1 Výber náhodného produktu

Jednou z možností, ktorú sme uvažovali ako prvú, bola tá, že sme z *wrap* tabuľky vybrali náhodný (respektíve prvý nájdený) produkt. Hodnoty nájdeného produktu sme následne použili ako vstup pre komplexný dopyt. Pri tomto prístupe však nastáva niekoľko problémov. Prvým je ten, že vybraný produkt môže obsahovať malé množstvo atribútov. Ak by sme teda vybrali náhodný produkt s dvoma atribútmi:

```
{
"Cena":"620",
"Rám": hiten 26“,
}
```

Je nepravdepodobné že ako výstup fulltextového vyhľadávania dostaneme to najlepšie čo sa dá. Rovnako sa môže stať, že takéto atribúty alebo hodnoty budú chybne extrahované. Reťazce by sa teda nezhodovali zo žiadnymi indexovanými produktmi. Takýto prístup sme preto vyhodnotili ako neefektívny.

5.1.2 Výber najlepšieho produktu

Druhou možnosťou by mohol byť výber najlepšieho produktu z tabuľky. To znamená, že vyberieme produkt, ktorý má čo najväčší počet atribútov pre jednu doménu. Všetky hodnoty atribútov následne použime ako vstup pre náš dopyt. Týmto spôsobom by sme teda odstránili problém, ktorý mohol nastať pri predchádzajúcom prístupe. Jedným z problémov pri tomto prístupe vytvárania objektu je ten, že tabuľka ktorú prehľadávame môže byť veľmi rozsiahla a takýto dopyt by mohol byť zdĺhavý. Otázkou ostáva, či takýto spôsob vytvorenia objektu pre vstup do dopytu je najlepší možný. Dostaneme pri takomto vstupe to najlepšie čo môžeme z indexovaných dát? Môže sa totiž stať, že síce máme produkt s najväčším počtom atribútov, ale nemáme všetky atribúty, ktoré produkt z danej domény môže nadobúdať. Napríklad bicykel s najväčším počtom atribútov 21 nemá atribút z názvom „*radenie“*. Ak by tento atribút obsahoval, tak je predpoklad, že fulltextový vyhľadávač dá na výstup „o niečo“ lepší výsledok. Ako by sme teda mohli takýto vstup zdokonaliť?

5.1.3 Vytvorenie pseudo produktu

Nakoniec sme si ako najlepší prístup vytvárania objektu zvolili vytváranie pseudo objektov. Pseudo objekt predstavuje fiktívny produkt, ktorý má najväčší možný počet atribútov. Vo *wrap* tabuľke sa pozrieme na všetky produkty z danej domény a následne vyberieme množinu všetkých atribútov, ktoré jednotlivé produkty nadobúdajú. Problémom ešte ostáva aké hodnoty týmto atribútom priradiť. Jednou z možností je priradiť jednému atribútu všetky hodnoty, ktoré sú mu priradené pri jednotlivých produktoch. Tento spôsob však môže spôsobiť, že výsledný vstup bude príliš rozsiahli a teda aj fulltextové vyhľadávanie bude trvať dlhšie. V práci sme preto zvolili prístup, v ktorom jednému atribútu priradíme náhodnú hodnotu (respektíve prvú nájdenú hodnotu). Hodnoty atribútov pseudo produktu použijeme ako vstup pre dopyt. Náš predpoklad je ten, že takýto produkt je najvhodnejší pri vyhľadávaní, pričom berieme do úvahy štruktúru indexovaných objektov. Týmto spôsobom taktiež predídeme problému pokrytia všetkých atribútov, ktorý nastáva pri prístupe výberu najlepšieho produktu.

5.2 Tvorba vstupu pre identifikáciu atribútov

Pri tvorbe vstupu pre identifikáciu atribútov sme zvolili rozdielny prístup ako pri doménach. Indexované dáta pritom majú rovnakú štruktúru, čiže vyhľadávame v rovnakom indexe. A rovnako výstup bude zoznam podobných produktov. Základná myšlienka bola všeobecne spomenutá v kapitole [3.3 identifikácia atribútov produktov]. Predpokladáme, že hľadáme atribút „*radenie*“ pre doménu bicykle. Jedným spôsobom je vytvoriť jednoduchý vstup, ktorý bude predstavovať jediná hodnota, predstavujúca náš hľadaný atribút pre daný produkt. Znova sa môžeme opýtať, či pomocou takéhoto objektu nájdeme to najlepšie čo sa dá, alebo či sa tento vstup nedá zdokonaliť, aby bol výsledok dopytu lepší. Je zrejmé, že atribút, ktorému chceme nájsť jemu podobný, nadobúda v tabuľke *wrap* niekoľko hodnôt (ak sa v množine produktov pre danú doménu vyskytol aspoň raz). Vstup pre dopyt jednoducho vytvoríme tak, že atribútu priradíme ako hodnotu pole hodnôt, ktoré sa vyskytnú pri danom reťazci vo *wrap* tabuľke. Ak máme napríklad atribút „*radenie“* pri doméne bicykle, a zároveň *wrap* tabuľka pre doménu bicykle obsahuje 10 produktov, z ktorých dva nadobúdajú hodnotu pre reťazec „*radenie*“, vstup pre dopyt by mohol vyzerať nasledovne:

```
{
 „riadenie“:[SHIMANO revoshift (18), SHIMANO Altus (24)],
}
```

6 Spracovanie JSON objektov ako výstupu dopytu pre identifikáciu domén a atribútov

V kapitole [4.1.1 Návrh štruktúry JSON objektov pre indexovanie] sme popísali formát vstupu objektov ktoré indexujeme. Proces indexovania máme teda ukončený. Nasleduje Proces samotného fulltextového vyhľadávania. Niektoré dopyty programu elasticsearch boli popísané v kapitole [5.1.3 Elasticsearch dopyty a filtre]. Po aplikovaní niektorého z dopytov dostaneme množinu „potenciálne zhodných“ produktov. Z nich je potrebné vyextrahovať informáciu o doméne a atribúte.

6.1 Extrahovanie podobných domén

Následný proces extrakcie potenciálne zhodnej domény je pomerne jednoduchý. Ako výsledok dopytu dostaneme zoznam produktov, ktoré sú podobné ako náš pseudo produkt. V poslednom kroku identifikácie domény preto prechádzame všetky objekty vrátené dopytom a extrahujeme z nich informáciu o doméne, do ktorej jednotlivé produkty patria. Získanú množinu domén ponúkneme používateľovi v administrátorskom prostredí ako potenciálne zhodné. Ten si následne môže jednu z ponúknutých domén vybrať a unifikovať ju s doménou na vstupe. Samozrejme je možné brať do úvahy aj počet výskytov domén vo výstupe a podľa toho v ponuke podobných domén uprednostniť tú pravdepodobnejšiu možnosť.

```
//vstupom je doména
public List<DBODomain> getListOfUnificableDomain(DBODomain domain) {
        //vytvoríme pseudo objekt JSON
        JSONObject json = jsonFormatCreate
                        .getJsonFormatInputForDomainUnification(domain);
        //pomocou fulltextového vyhľadávania získame podobné produkty
```

```
        ArrayList<JSONObject> jsonObjects = new ArrayList<>(
                elasticSearchUnification.searchDocumentInAllDocuments(json));
        //z objektov vyberieme domény
        Map<String, Integer> map = new HashMap<String, Integer>();
                for (JSONObject json  : jsonObjects) {

                        if (!map.containsKey(json.getDomain())){
                                map.put(json.getIdDomain(),json.getDomain());
                        }
                }
        //vytvoríme objekty domén s ktorými je možné unifikovať
        List<DBODomain> domains = new ArrayList<DBODomain>(
                mysqlProductData.getListOfUnificableDomain(map));
        //na výstup vrátime objekty typu DBODomain
        return domains;
}
```

6.2 Extrahovanie podobných atribútov

Extrakcia potenciálne podobných atribútov z výstupu dopytu je zložitejšia. Už vieme ako vyzerá vstup pre dopyt. Výstup dopytu je zoznam podobných produktov. Predpokladáme že dostaneme produkty, ktoré obsahujú podobný atribút, alebo podobné hodnoty ako nás vstup. Na rozdiel od domén však z výstupu extrahujeme podstatne väčšiu množinu, čiže všetky atribúty všetkých produktov. V tejto množine sa vyskytujú aj atribúty, v ktorých nenastala zhoda pri fulltextovom vyhľadávaní. Je to teda len filtrovanie produktov, ktoré môžu obsahovať atribút, unifikovateľné so vstupom. Asi by nebolo vhodné ponúknuť používateľovi administrátorského prostredia všetky atribúty všetkých produktov. Je preto potrebné znova filtrovať jednotlivé atribúty. Na to sme použili funkciu na porovnávanie reťazcov. Atribút, ktorý bol použitý ako vstup, porovnáme so všetkými atribútmi nájdených produktov. A rovnako všetky hodnoty atribútu na vstupe porovnáme so všetkými hodnotami atribútov jednotlivých produktov. Pri porovnávaní používame porovnávaciu funkciu Levenstein, ktorej prah sme definovali na hodnotu 0,3. Ak je teda pri porovnávaní splnená podmienka podobnosti v atribúte alebo v hodnote, tak daný atribút prehlásime za potenciálne zhodný a ponúkneme ho používateľovi administrátorského prostredia.

```
//vstupom je atribút
public List<DBOAttribute> getListOfUnificableAttributes(DBOAttribute
attribute) {
	//vytvoríme objekt JSON pre vstup atribútu
	JSONObject inputJson = jsonFormatCreate
			.getJsonFormatInputForAttributeUnification(attribute);
	//pomocou fulltextového vyhľadávania získame podobné produkty
	ArrayList<JSONObject> jsonObjects = new ArrayList<>(
	elasticSearchUnification.searchDocumentInAllDocuments(inputJson));
	Map<String, Integer> map = new HashMap<String, Integer>();
	for (JSONObject json  : jsonObjects) {
		//hľadáme zhodu v názvoch atribútov
		for (String attName  : json.getAttributeNames()) {
			if (Levenstein(inputJson.getName(), attName) > 0.3){
				map.put(getIdOfAttribute(attName), attName);
			}
		}
		//hľadáme zhodu v hodnotách atribútov
		for (String attVlaues  : json.getAttributeValues()) {
			for (String inputValues : inputJson.getValues()){
				if (Levenstein(attVlaues  , inputValues) > 0.3){
					map.put(getIdOfAttribute(attVlaues),
					attVlaues);
				}
			}
		}
	}
	//vytvoríme objekty atributov s ktorými je možné unifikovať
	List<DBOAttribute> attributes = new ArrayList<DBOAttribute>(
		mysqlProductData.getListOfUnificableAttributes(map));
	//na výstup vrátime objekty typu DBOAttribute
	return attributes;
```

7 Testovanie identifikácie

Pri práci sme identifikáciu otestovali len na pomerne malých dátových sadách. Konkrétne sme mali k dispozícii produkty z dvoch zdrojov (Eloshop.sk, Tpd.sk). Je

zrejmé že systém je možné dôkladne otestovať len v prípade, že máme veľké množstvo anotovaných dát ktoré následne indexujeme pre fulltextové vyhľadávanie. Taktiež je zrejmé, že ak chceme identifikovať napríklad doménu s názvom „*bicykle*“ a v systéme nie sú produkty takejto domény, tak program neponúkne správnu doménu (respektíve neponúkne žiadnu). Program tiež nie je možné automaticky testovať. Testovanie preto prebiehalo len ako vizuálna kontrola pri anotácií dát. Ak anotujeme doménu „*bicykle“* a zároveň v systéme máme indexované produkty tejto domény, tak predpokladáme, že program nám ponúkne túto doménu ako potenciálne podobnú. Ak sa tak stane, považujeme tento konkrétny test za úspešný.

Viem teda čo je na vstupe a čo očakávam v ponuke pre unifikáciu. Rovnako pri atribútoch, ak v systéme daný atribút nie je indexovaný, tak nám systém neponúkne atribút pre unifikáciu (respektíve ponúkne nesprávny). Nevýhodou nášho postupu pri vyhľadávaní podobných atribútov je fakt, že vyhľadávame podobné produkty a z nich následne extrahujeme atribúty na základe podobnosti (funkcie Levenstein). Pri doménach máme na vstupe podstatne rozsiahlejšie dáta pre vyhľadávanie na rozdiel od atribútov. Pri atribútoch sa často stane, že na vstupe máme je jednu hodnotu. Navyše táto hodnota sa môže vyskytovať pri mnohých nesúvisiacich atribútoch. Napríklad malé prirodzené čísla a boolean hodnoty a vyskytujú pri veľkom počte atribútov To môže spôsobiť situáciu, že fulltextové vyhľadávanie nájde podobné produkty aj v nesúvisiacich doménach, v ktorých sa hľadaný atribút nevyskytuje.

Príklad: predpokladajme, že máme na vstupe atribút *displej* z domény *Tablety*. Hodnota atribútu je len jedna (5), čo predstavuje uhlopriečku tabletu. Napriek tomu, že v systéme máme anotované domény *mobilné telefóny, GPS navigácie,* ktoré obsahujú atribút *veľkosť displeja* (s hodnotami rozdielnymi od „5“), systém nám ponúkne aj nesprávne atribúty nesúvisiacich domén ako „*počet dverí*“ alebo „*počet airbagov*“. Hodnota „5“ sa totiž často vyskytuje v doméne *autá.* Takéto chyby vyhľadávania je však ťažké eliminovať, keďže samotný vstup je tvorený len hodnotami daného atribútu, používané dopyty zároveň priradzujú názvy atribútov automatizovane podľa podobnosti hodnôt.

7.1 Anotácia dátových setov

V administrátorskom prostredí sme neanotovali všetky atribúty. Anotovali sme len tie, ktoré boli pre nás zaujímavé. Napríklad pri doméne *Reproduktory* sme neanotovali atribúty, ktoré neobsahovali dostatočnú informačnú hodnotu. Atribút *Farba mriežky* napríklad obsahoval len jednu hodnotu (-), čo síce môže byť korektná informácia a konkrétnom produkte, no obsahuje pre nás nepodstatnú hodnotu. V našom prípade je tento atribút nezaujímavý. Chceme totiž získať užitočné dáta o všetkých produktoch. V indexovaných dátach, rovnako aj v štruktúre dát, táto informácia predstavuje len jednu hodnotu jedného atribútu (resp. jednu hodnotu v stĺpci tabuľky). Takéto ignorovanie niektorých atribútov môžeme urobiť bez toho, aby nám to negatívne ovplyvnilo identifikáciu.

Anotovali sme dáta z jedného zo spomínaných zdrojov (Eloshop.sk). Plus dve domény z predchádzajúceho testovania už boli v systéme anotované. Pri zdroji *Eloshop* sme anotovali 12 domén. Pri každej z nich sme tiež anotovali väčšinu ich atribútov. V nasledujúcej tabuľke sú domény a počet atribútov, ktoré sme anotovali. Celý zoznam anotovaných atribútov je priložený v prílohe práce.

Tabuľka 2: Anotované domény

Anotované domény	Počet anotovaných atribútov	Počet produktov
Reproduktory	15	19
GPS navigácie	22	11
Chladničky-mrazničky	35	41
Mobilné telefóny	38	31
Tablety	25	16
Autorádiá	16	10
Autá	12	3339
Byty	9	27655
Notebooky	49	12
Televízory LCD/LED	35	39
Fotoaparáty	24	6
Digitálne záznamníky	10	4
Video kamery do auta	18	5
Rúry	41	37

Pri anotovaní sme vyberali domény, pre ktoré bola v systéme podobná doména. Chceme totiž, aby anotovaná doména mala v extrahovaných dátach podobnú alebo rovnakú doménu, nad ktorou môžeme následne vykonať testovanie. Napríklad doména „*chladničky-mrazničky“* mala v zdroji tpd.sk podobnú doménu „*americké chladničky“*, na ktorej sme program testovali. Pre väčšinu z vybraných domén na testovanie sme očakávali, že nám program ponúkne podobnú doménu a zároveň, že testovaná doména obsahuje atribúty, ktoré sú rovnaké ako naše indexované atribúty. Pre testovanie sme vybrali nasledovné domény s uvedeným počtom produktov v danej doméne a zdroji:

- Ultrabooky – 6 produktov (Eloshop.sk)
- Americká chladnička – 6 produktov (tpd.sk)
- Bezdrôtové audio systémy – 25 produktov (tpd.sk)
- Hifi veže – 45 produktov (tpd.sk)
- Vstavané rúry – 150 produktov (tpd.sk)
- Smartfóny - 191 produktov (tpd.sk)
- Tablety – 102 produktov (tpd.sk)
- Telefóny – 1 produkt (tpd.sk)

Domény boli zámerne vyberané z rôznych zdrojov, až na doménu *Ultrabooky*. Dôvodom je, že pri rôznych zdrojoch sú atribúty nazvané rôzne. Ak ale máme v systéme len atribúty jedného zdroja a identifikujeme atribúty podobnej domény rovnakého zdroja, tak je identifikácia jednoduchšia. Jeden internetový odchod má totiž relatívne zjednotenú množinu atribútov, ktoré popisujú podobné produkty. Ak teda chceme identifikovať atribúty domény *Ultrabooky* a v systéme máme indexované dáta o doméne *Notebooky* z rovnakého zdroja, tak tieto domény majú veľmi podobné atribúty, čo sa týka počtu, názvu alebo rozdelenia vlastností produktu. Obidve domény majú napríklad parameter *procesor* rozdelený na atribúty *procesor(taktovanie), procesor(výrobca), procesor(cache pamäť.)* pričom pri doméne *tablety* zo zdroja tpd.sk je jediný atribút *procesor* popisujúci všeobecné údaje.

V prvom kroku sme teda anotovali všetky spomínané domény a indexovali všetky produkty z daných domén. Následne sme testovali identifikáciu domén a atribútov na vybraných doménach a ich atribútoch.

7.2 Testovanie identifikácie domén

Pri doménach je postup získania potenciálne podobných domén jednoduchý. V administrátorskom prostredí si najprv vyberieme neanotovanú doménu a následne môžeme spustiť fulltextové vyhľadávanie podobných domén pomocou *fuzzy_like_this* alebo *more_like_this* dopytu. Pri testovaní identifikácii domén by však bolo potrebné mať v systéme indexované väčšie množstvo domén. Pri identifikácií domén je tiež potrebné si uvedomiť, či je pre danú neanotovanú doménu potrebne unifikovať, alebo vytvoriť novú doménu. Napríklad pri spomínanej doméne *„Americká chladnička“* je otázne, či túto doménu unifikovať s doménou *„cladničky-mrazničky“*, alebo vytvoriť novú doménu. Každopádne sú však tieto domény podobné a teda očakávame, že nám pri tomto vstupe, program ponúkne doménu *„chladničky-mrazničky“*. Program totiž nehľadá len rovnaké domény, ale snaží sa ponúknuť potenciálne zhodné. Ak nám teda takúto doménu ponúkne, tak považujeme konkrétny test za úspešný. Nasledujúca tabuľka zobrazuje ponúknuté potenciálne zhodné domény, pre naše testované domény:

Tabuľka 3 výsledky testovania pre dopyt fuzzy_like_this

Neanotovaná doména	Ponúknuté podobné domény	Skóre
Ultrabooky	Notebooky	7.2162395
	Mobilné telefóny	0.69494104
	Tablety	0.652154
	byty	0.35133487
	GPS navigácie	0.2515042
	Fotoaparáty	0.11468128
	Autorádiá	0.080826506
	Televízory LCD/LED	0.056258634
	auta	0.055324003
Americká chladnička	Chladničky-mrazničky	5.50562
	Rúry	3.3259797
	Tablety	1.0373498
	Mobilné telefóny	0.70056564
	Fotoaparáty	0.4858405
	Video kamery do auta	0.3915813
	Autorádiá	0.33327544
	Televízory LCD/LED	0.15638112
	Notebooky	0.15215813

Vstavané rúry	Rúry	4.69559
	Chladničky-mrazničky	1.5214827
	Fotoaparáty	0.7055877
	Video kamery do auta	0.58472717
	Mobilné telefóny	0.54805696
	Televízory LCD/LED	0.4145479
	Notebooky	0.12027543
	Digitálne záznamníky	0.09743683
	Autorádiá	0.09257443
Smartfóny	Mobilné telefóny	0.69952905
	Tablety	0.34102863
	Notebooky	0.18107426
	GPS navigácie	0.16388255
	Rúry	0.11982722
	byty	0.090440184
	Digitálne záznamníky	0.0655646
	auta	0.054274797
	Chladničky-mrazničky	0.041745923
	Fotoaparáty	0.04064196
	Autorádiá	0.029397821
	Video kamery do auta	0.009066051
	Televízory LCD/LED	0.009038674
Tablety	Mobilné telefóny	1.0782424
	Tablety	0.9037733
	Notebooky	0.65600103
	Rúry	0.2983555
	Chladničky-mrazničky	0.15187608
	Fotoaparáty	0.14450203
	Televízory LCD/LED	0.12488546
	Video kamery do auta	0.12315068
	GPS navigácie	0.09664688
	Digitálne záznamníky	0.052855186
	byty	0.022108926
	auta	0.018227993
Telefóny	Mobilné telefóny	1.3300648
	Tablety	0.4008851
	Fotoaparáty	0.3607477
	Notebooky	0.29877365
	Rúry	0.24366166

	Televízory LCD/LED	0.102760024
	Video kamery do auta	0.0673046
	Digitálne záznamníky	0.055415515
	Chladničky-mrazničky	0.042149905
	byty	0.03933628
	auta	0.03580302

Tabuľka 4 Výsledky testovania pre dopyt more_like_this

Neanotovaná doména	Ponúknuté podobné domény	Skóre
Ultrabooky	Notebooky	7.387376
	Tablety	6.880407
	Mobilné telefóny	5.0761414
	Televízory LCD/LED	2.2582784
	Autorádiá	1.506651
	Fotoaparáty	1.4056674
	GPS navigácie	0.33943698
	Chladničky-mrazničky	0.29898718
Americká chladnička	Rúry	3.413213
	Chladničky-mrazničky	3.3581405
	auta	2.2057734
	Televízory LCD/LED	1.920394
	Tablety	1.7560257
	Autorádiá	1.6484237
	Mobilné telefóny	1.53252
	Fotoaparáty	1.5150161
	Notebooky	1.1600156
	GPS navigácie	1.0901036
Vstavané rúry	Rúry	8.594109
	Chladničky-mrazničky	5.8256297
	Televízory LCD/LED	2.5789785
	Notebooky	2.332253
	Mobilné telefóny	1.9963098
	Fotoaparáty	1.6139462
	Reproduktory	0.62414324
	Tablety	0.33949572
Smartfóny	Mobilné telefóny	12.860161
	Rúry	5.8503175
	Tablety	4.74605

	Fotoaparáty	1.3541433
	Notebooky	1.1391613
	Televízory LCD/LED	0.94526756
	Autorádiá	0.44656637
	Chladničky-mrazničky	0.3977588
	GPS navigácie	0.36010292
Tablety	Mobilné telefóny	9.574222
	Tablety	5.7906256
	Notebooky	4.8453546
	Televízory LCD/LED	3.363922
	Fotoaparáty	1.6863539
	Rúry	1.1100332
	GPS navigácie	0.8768078
	Chladničky-mrazničky	0.8495779
	Autorádiá	0.5198906
Telefóny	Mobilné telefóny	13.689914
	Rúry	4.0121307
	Tablety	2.094346
	Notebooky	1.6086028
	Fotoaparáty	0.87870026
	Chladničky-mrazničky	0.7901001
	Televízory LCD/LED	0.7725333
	Reproduktory	0.4127854
	auta	0.26128882
	GPS navigácie	0.2550816

V tabuľkách je možné vidieť, že v ponúknutých doménach sa vyskytuje hľadaná doména, s pomerne vysokým skóre, pre všetky hľadané domény. Skóre pre doménu je vyrátané ako súčet skóre hodnôt pre jednotlivé produkty z danej domény. Ak nám teda fulltextový vyhľadávač vrátil 4 produkty z domény *Tablety*, tak výsledné skóre je súčet štyroch hodnôt týchto produktov. V administrátorskom prostredí sú ponúknuté domény zoradené od najviac po najmenej podobné. Vďaka tomu vieme pri anotácii vidieť, ktoré domény majú vyššiu mieru zhody z hľadanou doménou. Skóre pre jeden produkt je rátaný programom elasticsearch a predstavuje mieru podobnosti daného produktu s naším vstupom.

Je pomerne komplikované na základe týchto dát počítať percentuálnu úspešnosť identifikácie domén. Zároveň je vidieť, že program nám taktiež ponúkol veľa

nesúvisiacich domén, čo môže byť spôsobené malým množstvom produktov v podobných doménach. Pri komplexných dopytoch totiž nie je možné nastaviť prah zhody, pod ktorým už indexovaný produkt nie je prehlásený za podobný. Bolo teda potrebné nastaviť počet výsledkov, ktoré má fulltextové vyhľadávanie vrátit. Pre optimalizáciu vyhľadávania v malom množstve produktov by bolo vhodné upraviť počet produktov, ktoré má dopyt vrátiť. Takéto nastavenie však nie je efektívne. V prípade, že by systém obsahoval väčší počet produktov je predpoklad, že ponúknutých domén bude menej. V súčasnom stave ale nemáme k dispozícií rozsiahlejšie dáta. Napriek tomu, že nám systém ponúkol veľké množstvo nesúvisiacich domén, môžeme prehlásiť, že program nám ponúkol aj správne identifikované domény s naším vstupom. Z dát, ktoré máme k dispozícii je možné zistiť mieru presnosti a návratnosti nášho programu na dátach, ktoré máme k dispozícií. Tieto hodnoty rátame podľa klasického *Precesion – recall* scenáru. Uvažujeme pritom štyri základné hodnoty:

- **True positive** – počet požadovaných hodnôt, ktoré nám program vrátil
- **True negative** – počet hodnôt, ktoré program neponúkol a ani nemal ponúknuť
- **False positive –** počet nesprávne identifikovaných hodnôt
- **False negative** – počet hodnôt, ktoré program mal ponúknuť pre unifikáciu, ale neponúkol

Na základe týchto hodnôt teda môžeme zrátať presnosť a návratnosť nášho programu podľa nasledovných vzorcov:

$$\boldsymbol{precesion} = \frac{\mathbf{tn}}{\mathbf{tn} + \mathbf{fp}}$$

$$\boldsymbol{recall} = \frac{\mathbf{tp}}{\mathbf{tp} + \mathbf{fn}}$$

Z otestovaných dát sme vypočítali spomínané štyri základné hodnoty. Tie sme počítali pomocou celkového počtu anotovaných domén, ktoré sú v systéme indexované. Ako bolo spomenuté, celkový počet indexovaných domén je 14. V nasledujúcej tabuľke sú celkové hodnoty, ktoré platia per celú testovanú sadu (teda pre všetkých 6 domén) a

ktoré sme získali pomocou podmienok, ktoré sú popísané vyššie. Zároveň uvažujeme, že očakávaná hodnota, ktorú mal program vrátiť je len jedna konkrétna hodnota:

Tabuľka 5 klasifikačné hodnoty pre použité dopyty

	True positive	**True negative**	**False positive**	**False negative**
more_like_this	6	30	40	0
fuzzy_like_this	6	21	57	0

Výsledné percentuálne hodnoty presnosti a návratnosti vyrátané podľa spomínaných vzťahov sú nasledovné:

Tabuľka 6 presnosť a návratnosť pre testované dopyty

	Presnosť	**Návratnosť**
more_like_this	42.86%	100%
fuzzy_like_this	26.92%	100%

Z hodnôt v tabuľke je zrejmé, že na presnejšie vyčíslenie hodnôt je potrebné otestovať väčšiu množinu domén. Môžeme však povedať, že na testovaných dátach máme vysokú návratnosť pri obidvoch dopytoch. V praxi to znamená, že program identifikoval všetky domény správne, alebo ponúkol aj doménu, ktorú sme očakávali. Na druhej strane sme dosiahli nízku presnosť, čo znamená, že program nám ponúkol veľké množstvo nesúvisiacich domén. Predpokladáme však, že pri rozsiahlejšom otestovaní stúpne presnosť a návratnosť mierne klesne.

7.3 Testovanie identifikácie atribútov

Testovanie identifikácie atribútov je o niečo komplikovanejšie ako pri doménach. Na druhej strane však vieme vyčísliť percentuálnu úspešnosť korektnej identifikácie podľa počtu správne identifikovaných atribútov pre danú doménu. Je však potrebné si uvedomiť niekoľko vecí. Ako som spomínal, pri atribútoch je vstupom množina hodnôt, ktoré atribút nadobúda. Samotné hodnoty však hovoria veľa aj o konkrétnom atribúte. Predstavme si, že máme atribút „*bluetooth*“, ktorý nadobúda hodnoty *áno/nie*. Pri inej

doméne však tieto hodnoty môžu byť konkrétnejšie. Napríklad v hodnote atribútu „*bluetooth"* môže byť presnejšie špecifikované o aký bluetooth ide (napríklad 4.0). Je potrebné si uvedomiť, že ide o dva rozdielne atribúty. A teda ak nám ponúkne takýto atribút, ktorý sa zhoduje v názve, ale má rozdielne hodnoty, tak ich nie je možné unifikovať.

Pred samotným testovaním je taktiež potrebné si uvedomiť, aké atribúty budeme testovať (resp. či budeme testovať na všetkých atribútoch). Ako som spomínal, ak nemáme indexovaný atribút, s ktorým je možné unifikovať, tak nám systém ponúkne len nesúvisiace alebo žiadne atribúty. Táto skutočnosť je pri testovaní nežiaduca a teda atribúty, ktoré nemajú v systéme indexovaného dvojníka nebudeme uvažovať. Zo spomínaných domén vyberieme len atribúty, o ktorých vieme, že už sú v systéme a následne porovnávame ponúknuté atribúty s našim atribútom (vstupom). Ak nám systém ponúkne atribút, ktorý sme očakávali, tak tento konkrétny test považujeme za úspešný. Takýmto spôsobom vyhľadávame atribúty v testovaných doménach a vyhodnocujeme úspešnosť vyhľadávania pre každú z domén. Atribúty, ktoré boli testované sú uvedené v prílohe. Pre každú doménu je priložený súbor, v ktorom je uvedený testovaný atribút, ponúknuté možnosti pre unifikáciu a informácia o tom, či v ponúknutých atribútoch bola aj očakávaná (korektná) hodnota. Nasledujúca tabuľka obsahuje len konečné percentuálne vyhodnotenie pre jednotlivé domény. Stĺpec „počet správne identifikovaných atribútov" obsahuje informáciu o tom, koľko z hľadaných atribútov bolo identifikovaných správne. Čiže zhodný atribút, s ktorým je možné unifikovať, sa vyskytol vo výslednej množine.

Tabuľka 7 Testovanie atribútov pre dopyt fuzzy_like_this

Testované domény	**Počet hľadaných atribútov**	**Počet správne identifikovaných atribútov**	**Percentuálny počet správne identifikovaných atribútov**
Ultrabooky	35	22	62.86%
Americká chladnička	16	8	50%
Bezdrôtové audio systémy	18	15	83.3%
Hifi veže	13	9	69.23%
Vstavané rúry	19	15	78.95%

Smartfóny	33	26	78.79%
Tablety	20	14	70%
Telefóny	18	13	72.2%
SPOLU	172	123	71.51%

Tabuľka 8 Testovanie atribútov pre dopyt more_like_this

Testované domény	**Počet hľadaných atribútov**	**Počet správne identifikovaných atribútov**	**Percentuálny počet správne identifikovaných atribútov**
Ultrabooky	35	11	33.43%
Americká chladnička	16	8	50%
Bezdrôtové audio systémy	18	12	66.6%
Hifi veže	13	8	61.54%
Vstavané rúry	19	15	78.95%
Smartfóny	33	23	69.7%
Tablety	20	13	65%
Telefóny	18	9	50%
SPOLU	172	99	57.56%

Pri porovnaní výsledkov testovania pre rôzne dopyty je možné vidieť, že dopyt *fuzzy_like_this* má minimálne také výsledné percentuálne hodnoty ako dopyt *more_like_this*. Dopyt, ktorý využíva fuzzy funkciu je teda o niečo lepší pri identifikácii atribútov. Tento fakt je spôsobený tým, že dopyt *fuzzy_like_this* využíva funkciu *edit distance* (alebo fuzzy funkciu) pre reťazce. Na základe tejto podobnosti z hodnotami sú vyberané atribúty pre dopyt. Na rozdiel od toho dopyt *more_like_this* vyberá atribúty len pomocou *tf* a *idf*, ktoré počíta ako počet výskytov termu. Je teda zrejmé, že hodnota sa musí v indexovaných dátach explicitne nachádzať v rovnakom tvare. Ak ale predpokladáme, že v systéme sa hľadané slovo nachádza v inom tvare tak tieto nie sú považované za rovnaké, ako je to pri fuzzy funkcii. Problémom sú tiež chybne extrahované dáta. V prípade, že chceme nájsť atribút „*Farba“* s hodnotou „*?ierna“* pomocou dopytu *more_like_this* a v systéme sa nachádza produkt z hodnotou „*Čierna“*, tak tieto hodnoty sú považované za rozdielne.

Pri identifikácii sa vyskytlo aj niekoľko neočakávaných atribútov. Hlavným dôvodom nájdenia aj nechcených atribútov sú ich hodnoty. Konkrétne tzv „kameňom úrazu“ môžu byť *boolean* hodnoty a celočíselné hodnoty pri atribútoch. Asi nie je prekvapujúce, že pri väčšine produktov sú hodnoty atribútov typu *áno/nie*. Tento fakt môže negatívne ovplyvniť celý proces identifikácie. Pri testovaní sa napríklad vyskytol problém, že pri hľadaní atribútu, ktorý mal takéto hodnoty, nám program ponúkol veľké množstvo podobných atribútov (keďže po získaní podobných produktov v týchto dokumentoch následne hľadáme zhodu podľa hodnôt a podľa názvu atribútu). Niekedy program, pri takomto vstupe ponúkol rádovo desiatky podobných atribútov.

Rovnako ako pri doménach aj tu rátame skóre jednotlivých atribútov, vďaka čomu vieme v administrátorskom prostredí ponúknuť pravdepodobnejšiu možnosť. To môže uľahčiť anotáciu v prípade, že systém ponúkne desiatky podobných atribútov. Skóre pre jeden atribút je vyrátaný ako súčin dvoch hodnôt. Prvou je hodnota, ktorú získal produkt pri fulltextovom vyhľadávaní. Druhou je hodnota podobnosti získaná funkciou edit distance. Zároveň filtrujeme atribúty, ktoré majú hodnotu podobnosti menšiu ako 0.3. Ak na stala zhoda pri rovnakom atribúte viac krát, tak do výsledku vložíme atribút, ktorý získal vyššie skóre. Zároveň zohľadňujeme boolean hodnoty. To znamená, že ak hľadaný atribút obsahuje boolean hodnoty, tak považujeme zhodu v názve atribútu za relevantnejšiu. V praxi to znamená, že ak má boolean hodnoty a nastane zhoda v názve atribútu, tak skóre ktoré nadobúda zdvojnásobíme. Zároveň berieme do úvahy aj situáciu, že nastala zhoda aj v názve, aj v hodnote. V tomto prípade hodnoty sčítavame.

Vzhľadom k tomu, že v súčasnosti nemáme k dispozícii rozsiahlejšie dáta z viacerých zdrojov, nie je možné otestovať to, ako sa bude systém správať pri veľkom množstve *boolean* hodnôt. Pretože dopyt (ako bolo popísané) hľadá podobností na základe zhody v hodnotách.

V administrátorskom prostredí požadujeme takzvanú „*real time*“ odozvu, aby administrátor nemusel čakať na ponuku pre unifikáciu. Alebo aby sa odozva programu aspoň blížila „*real time*“ odozve. Pri hľadaní atribútov, výsledky fulltextového vyhľadávania znova prehľadávame. Tento čas opätovného hľadania je možné považovať za konštantný , pretože je zadefinovaný počet dokumentov, ktoré má vrátiť elasticsearch a teda prehľadávame vždy rovnaký počet dokumentov pri hľadaní domény alebo atribútov. Otázna je časová odozva programu elasticsearch pri veľkom množstve indexovaných produktov.

Záver

V úvode práce sme si problém unifikácie metadát rozdelili na dva základné pod problémy. Prvým je identifikácia a druhým je samotná unifikácia. Zaoberali sme sa hlavne problémom identifikácie podobných metadát, s ktorými je následne možné vykonať jednoduchý proces zlúčenia. Na riešenie problému identifikácie domén a atribútov sme v práci použili fulltextové vyhľadávanie, konkrétne program elasticsearch, ktorý takéto vyhľadávanie podporuje. Použili sme dva dopyty, pomocou ktorých sme vyhľadávali podobnosti v produktoch. Zvažovali sme pritom niekoľko spôsobov vytvárania vstupov pre tieto dopyty, aby sme z anotovaných dát pomocou vyhľadávača získali čo najpodobnejšie hodnoty. Zároveň bolo potrebné vytvoriť taký formát dokumentov pre indexovanie aby bolo možné v týchto dokumentoch efektívne vyhľadávať. Formát dát, ktoré indexujeme dostávame aj na výstupe fulltextového vyhľadávača. Preto bolo potrebné analyzovať výstup vyhľadávača a získať z neho tie najlepšie hodnoty. Základom analýzy týchto dát bol hlavne filtrovanie podobností pri atribútoch a doménach. Druhou fázou bol výpočet skóre hodnoty pre jednotlivé domény a atribúty, pomocou ktorých vieme v administrátorskom systéme ponúknuť pravdepodobnejšiu možnosť.

Pri testovaní domén sme zistili, že program nám síce vracia tie domény, ktoré potrebujeme, alebo ktoré sme očakávali, ale vracia tiež veľké množstvo nesúvisiacich domén. Tento fakt je spôsobený malým počtom indexovaných produktov. Môžeme ale povedať, že na testovaných dátach funguje efektívne a pomocou skóre vieme administrátorovi ponúknuť aj pravdepodobnosť zhody. Pri testovaní atribútov pomocou rôznych dopytov sa ukázalo, že dopyt *fuzzy_like_this* vracia o niečo lepšie hodnoty. Konkrétne bol tento dopyt v konečných číslach oproti dopytu *more_like_this* lepší o približne 14%. Táto hodnota je rozdielom výsledkov úspešnosti identifikácie naprieč celým testovaným dátovým setom. Tento rozdiel môže byť spôsobený hlavne požadovaním presnej zhody reťazcov pri priraďovaní atribútov dopytu *more_like_this*. Dopyty môžu ovplyvniť taktiež chybne extrahované hodnoty atribútov z internetových zdrojov.

Vytvorili sme program, pomocou ktorého vieme administrátorovi uľahčiť anotáciu dát, čo bolo hlavným cieľom tejto práce. Zároveň sme tento program implementovali do súčasného administrátorského prostredia.

Zoznam použitej literatúry

[1] Peter Christen: Data matching. Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate detection. Springer, 2012, ISBN 978-3-642-31164-2

[2] Venkatesh Ganti, Anish Das Sarma: Data Cleaning. A Practical Perspective. Morgan & Claypool publishers, 2013, ISBN 978-1-608-45678-9