ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Framework pro vyhodnocování úspěšnosti algoritmů častých sekvencí v doporučovacích systémech
Student:	Michal Bajer
Vedoucí:	Ing. Tomáš Řehořek
Studijní program:	Informatika
Studijní obor:	Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2017/18

Pokyny pro vypracování

Proveďte rešerši algoritmů pro hledání častých sekvencí v transakčních datech. Seznamte se s problematikou doporučovacích systémů a kolaborativního filtrování, zejména pak s vyhodnocováním úspěšnosti doporučovacích algoritmů. Navrhněte způsob, jak využít vytěžené časté sekvence k tvorbě doporučovacího modelu. Dále navrhněte modulární framework, který zastřeší vytěžování sekvencí, aplikaci modelů na testovací uživatele a vyhodnocení jejich úspěšnosti. Implementujte framework v jazyku Java a začleňte do něj zvolené algoritmy pro každou část celého procesu. Framework otestujte a změřte výsledky na alespoň dvou dodaných testovacích datasetech a naměřené výsledky diskutujte.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 4. ledna 2017

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Bakalářská práce

Framework pro vyhodnocování úspěšnosti algoritmů častých sekvencí v doporučovacích systémech

Michal Bajer

Vedoucí práce: Ing. Tomáš Řehořek

8. května 2017

Poděkování

V první řadě bych rád poděkoval svému vedoucímu bakalářské práce, panu Ing. Tomáši Řehořkovi, za jeho vstřícný přístup a podporu během tvorby této práce. Dále bych rád poděkoval společnosti Recombee za poskytnutí testovacího serveru.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 8. května 2017 .....

Odkaz na tuto práci

Bajer, Michal. *Framework pro vyhodnocování úspěšnosti algoritmů častých sekvencí v doporučovacích systémech.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

Tato bakalářská práce pojednává o analýze sekvenčních dat, hledání častých vzorů v časově uspořádaných datech a jeho využití k doporučování produktů zákazníkům internetových obchodů na základě jejich historie nákupů.

Cílem práce je navrhnout a implementovat framework, který umožňuje zpracovat nasbíraná data, najít v nich vzory a určit na jejich základě pravděpodobné pokračování sekvence nákupů zákazníka. Dalším cílem je umožnit vyhodnocení úspěšnosti několika různých algoritmů a vybrat nejvhodnější pro specifický účel.

Výsledkem práce je návrh, diskuze možností a implementace funkčního frameworku pro doporučování produktů a vyhodnocení úspěšnosti. Na konec proběhla analýza kvality doporučení za použití implementovaných algoritmů. Implementace práce proběhla v jazyce Java.

Klíčová slova doporučovací systémy, sekvenční data, sequence mining, internetové obchody, nákupní vzory

Abstract

This bachelor thesis deals with the analysis of sequential data, the search of frequent patterns in time-based data and its use for the recommendation of products to the customers of Internet shops based on their history of purchases.

The aim of the thesis is to design and implement a framework that allows the processing of collected data, to find patterns and to determine the probable continuation of the customer purchase sequence. Another goal is to allow the evaluation of the success of several different algorithms and to choose the most suitable for a specific purpose.

The result of the thesis is the design, discussion of options and implementation of a functional framework for product recommendation and success evaluation. At the end, the quality of the recommendations was analysed using the implemented algorithms. The implementation of the work was done in Java.

Keywords recommendation systems, sequential data, sequence mining, internet shops, purchases patterns

Obsah

Seznam obrázků

Seznam tabulek

Úvod

V dnešní době se často mluví o analýze dat a jejím využití pro podporu podnikání, získání lepších podkladů pro rozhodování a v mnoha dalších oblastech. Jedna z oblastí, kde se podobné technologie již s úspěchem využívají, je doporučování produktů, stránek k navštívení, videí ke zhlédnutí a dalších druhů obsahu. Obecně se tato oblast označuje jako doporučovací systémy. Díky velkému potenciálu pro některé firmy již existuje množství různých řešení.

Každý takový systém dělá určité předpoklady o tom, jak se bude zákazník chovat. Většinou se předpokládá, že bude mít zájem o podobné věci, jako další lidé s podobnou historií (nákupů, zhlédnutí atd.). V některých situacích (např. při sledování seriálů) je velice zajímavé i to, v jakém pořadí uživatel díly sledoval. V této práci se budu věnovat tomu, jak tuto myšlenku implementovat a využít k doporučování.

Praktická část mé práce je založena na implementaci frameworku, který by dokázal vyhledat zajímavé sekvence v historických datech o uživatelích a vytvořit na jejich základě doporučení. Tyto doporučení je poté potřeba porovnat s původními daty a vyhodnotit úspěšnost. Tento framework musí mít dostatečně obecnou strukturu, aby bylo možné použít různé implementace konkrétních částí.

V první části popíši obecnou problematiku doporučování a základní přístupy. Dále popíši základní algoritmy pro hledání častých sekvencí. Nakonec vysvětlím podrobněji přístup, který jsem zvolil pro svoji implementaci.

V druhé části bude popsána struktura frameworku a také návrh algoritmu pro doporučování, který jsem implementoval na základě standardního přístupu upraveného pro tento účel. V další části popíši podrobnosti o implementaci jednotlivých částí frameworku.

V poslední části provedu testování na skutečných datech a popíši různé pokusy o nalezení vhodných parametrů a výsledky těchto pokusů. Na konec se z nich pokusím vyvodit závěr o vhodnosti tohoto přístupu a silných a slabých stránkách této implementace.

Kapitola 1

Cíl práce

Cílem práce je provést analýzu problematiky doporučování a potenciálu využití algoritmů pro hledání častých sekvencí v této oblasti. Výstup z rešerše poté využít k návrhu frameworku, který bude umět najít časté sekvence s pomocí algoritmů nalezených v literatuře.

Druhá část tohoto frameworku bude obsahovat algoritmy, které využijí nalezené sekvence pro doporučení. Algoritmy budou navrženy na základě rešerše a přizpůsobeny pro tento účel. Návrh musí být dostatečně obecný a dobře rozšiřitelný, aby umožnil testování a vyhodnocení různých algoritmů a implementací.

V praktické části je cílem implementovat pro každou část frameworku jeden nebo více algoritmů a vyhodnotit jejich úspěšnost při použití na skutečných datech. Testy a vyhodnocení algoritmů s různými parametry podle běžných metrik budou tvořit poslední část práce.

Kapitola 2

Analýza

2.1 Doporučovací systémy

Systémy pro doporučování obsahu začaly být běžné s nástupem velkých internetových obchodů před cca 15 lety. Obchody nabízely velké množství produktů, ale většina uživatelů se zajímá pouze o malou část z nich. Otázka toho, jak nabídnout uživateli právě to, co ho zajímá, začala být v komerční sféře mnohem výraznější než dřív. Účelem doporučovacích systémů je z velkého množství produktů vybrat několik, které mají velkou šanci uživatele zaujmout.

Tato sekce popisuje základní rozdělení problematiky doporučování tak, jak je obvykle prezentováno v literatuře. Jedná se o rozdělení především na základě toho, jaké informace se k doporučování používají. Některé systémy se zaměřují pouze na uživatele a jejich historii, jiné se snaží hledat souvislosti mezi produkty.

Přístup zvolený v této práci využívá pouze historii uživatelů a z tohoto hlediska se tedy blíží nejvíce systémům z následující kategorie.

2.1.1 Kolaborativní filtrování

Tento přístup využívá toho, že uživatelé, kteří mají podobnou nákupní historii, pravděpodobně mají zájem o podobné oblasti. Stačí tedy najít uživatele s podobnou historií a doporučit položky, které si tito uživatelé koupili, ale ten, pro kterého doporučení děláme, zatím ne.

Tento přístup patří k nejstarším a úspěšně se používá v různých oblastech. Hlavní výhodou je velká robustnost, protože není nutné vědět nic o produktech ani o zákaznících, kromě jejich historie nákupů. Algoritmus pracuje pouze s maticí uživatelů a produktů, v které je tato historie zaznamenána. Na druhou stranu lze tuto vlastnost považovat i za nevýhodu, protože často je k dispozici více informací, které by potenciálně mohly být využity k získání lepších výsledků.

Postupně se ukázalo, že výpočetní náročnost těchto algoritmů je obrovská, jelikož velké internetové obchody mohou mít miliony zákazníků a stovky tisíc produktů. Matici vytvořenou pro tento algoritmus pak nelze zpracovat během několika stovek milisekund, které jsou pro nalezení doporučení k dispozici. Zároveň má tento přístup i vysokou paměťovou náročnost, protože tato matice je rozsáhlá, ale přitom v praxi velice řídká. Běžný zákazník totiž koupí z nabízených tisíců produktů pouze jednotky.

Tento problém se řeší různými způsoby. První možností je použít téměř stejný algoritmus na produkty. Místo hledání podobných uživatelů lze hledat podobné produkty a to analogickým přístupem. Hledají se produkty, které se společně nacházejí v historii nákupů více uživatelů a vytvoří se tzv. matice podobnosti. Poté stačí najít produkty, které už daný zákazník koupil a k nim v této matici najít několik nejpodobnějších. Hlavní výhodou tohoto přístupu je to, že matici podobnosti je možné vypočítat předem a při požadavku na doporučení pro konkrétního zákazníka už v ní pouze vyhledávat.

Moderní implementace toho přístupu využívají tzv. faktorizaci matic. Velice zjednodušeně řečeno se zde používají poznatky z lineární algebry k nalezení menší matice (a to v praxi podstatně menší, jelikož původní matice je extrémně řídká), která ale stále obsahuje většinu informací o vztazích mezi produkty a zákazníky. Podle výsledků různých soutěží se zdá, že tento přístup má velký potenciál.

2.1.2 Doporučení založené na obsahu

V tomto přístupu se využívají pouze data o historii nákupů jednoho uživatele v kombinaci s informacemi o produktech. Typicky se využívají kategorie produktů, případně klíčová slova. Obojí umožní vytvořit skupiny podobných produktů.

Algoritmy založené na tomto přístupu obvykle využívají hodnocení, které uživatel dal produktům, s kterými již interagoval. Pomocí známých vazeb mezi nimi pak doporučují ty produkty, které jsou podobné těm, které uživatel hodnotil pozitivně.

Zajímavou vlastností toho přístupu je, že díky využití předchozích hodnocení je možné uživateli vysvětlit, na základě čeho bylo doporučení nalezeno. To značně zvyšuje důvěryhodnost doporučení. Další výhodou je to, že je možné doporučovat i nové produkty, které ještě nikdo nekoupil, pouze na základě zařazení do kategorie.

Tento přístup má naopak velký problém s doporučováním pro nové uživatele. Pro smysluplné doporučení je totiž nutné, aby uživatel ohodnotil určitý počet produktů. Proto se v praxi většinou kombinuje s jinými přístupy, které umožní v takové situaci najít doporučení jiným způsobem.

2.1.3 Doporučení založené na znalostech

Tento přístup se zásadně liší od předchozích, protože nevyužívá již sesbíraná data o uživatelích. Parametry zadává uživatel sám a systém podle nich vyhledává produkty. To umožňuje doporučování na základě mnohem většího množství parametrů.

Využívá se v oblastech, kde nelze očekávat, že zákazník bude mít dostatečnou nákupní historii nebo dostatečný počet hodnocení k tomu, aby mohly být využity předchozí metody. Typickým příkladem je prodej aut, nemovitostí, zájezdů atd.

Zároveň se tento přístup využívá v kombinaci s ostatními pro doporučování novým uživatelům, o kterých systém založený na jiných metodách ještě nemá žádné informace.

2.1.4 Hybridní techniky

Jak již bylo naznačeno v předchozích sekcích, každý z těchto přístupů má svoje nedostatky. Častým problémem je neschopnost pracovat s novými uživateli či novými položkami v nabídce.

Možným řešením těchto problémů je využít několik druhů doporučování a výsledky z nich sloučit. To lze dělat různými způsoby. Nejjednodušeji tak, že se z každého systému vybere určitý počet položek a prezentují se zákazníkovi společně.

Další jednoduchou možností je spouštět různé systémy v určitém pořadí, dokud některý nevyprodukuje požadovaný počet doporučení. V tomto přístupu lze zajistit úspěch tím, že první systémy jsou založeny na specifických přístupech, které nemusí nutně vyprodukovat žádný výsledek, ale jejich výsledky jsou často přesnější a naopak poslední může být jednoduchý seznam bestsellerů, které lze vrátit vždy.

Sofistikovanější systémy mohou využívat vážený součet hodnocení položek v různých systémech a doporučí ty, které mají nejvyšší sloučené hodnocení. Nalezení vah v takovémto systému je samo o sobě poměrně náročné.

Hybridní techniky umožňují dosahovat velkých zlepšení, především pokud se vyladí na míru pro dané použití. Tyto systémy byly také velice úspěšné v různých soutěžích.

Tato sekce je založena na poznatcích z knihy [1] a výtahu zpracovaném v [2].

2.1.5 Soutěž Netflix prize

Jednou z největších příležitostí, kde došlo k porovnání různých přístupů a algoritmů pro doporučování, byla soutěž vyhlášená společností Netflix v roce 2006. Netflix poskytl účastníkům rozsáhlý vzorek vlastních dat a nabídl cenu jeden milion dolarů pro tým, kterému se podaří dosáhnout nejvyšší úspěšnosti

doporučení. Podmínkou bylo, že se úspěšnost doporučení zvýší alespoň o 10% oproti stávajícím systémům.[3]

Vítězem se stal tým BellKor's Pragmatic Chaos, který dokázal v září 2009 posunout úspěšnost o 10,06%. Jejich algoritmus je kombinací cca stovky různých přístupů. Hlavní částí byl kolaborativní algoritmus využívající maticovou faktorizaci. Další části zpracovávaly informace o čase zhlédnutí a různé další dostupné informace. Autoři popsali celý systém ve vlastním článku. [4]

2.2 Hledání častých sekvencí

Problematika hledání častých sekvencí je jedna z méně významných oblastí analýzy dat. Byla poprvé popsána v článku R. Agrawala a R. Srikanta [5] z roku 1995. Lze zde nalézt obecnou definici problému a tři navrhované algoritmy pro řešení. Tyto konkrétní algoritmy jsou z hlediska efektivity již překonané, ale myšlenky na kterých stojí se stále využívají.

Vzhledem k tomu, že se jedná o poměrně specifickou oblast, existují především teoretické články, které představují několik základních algoritmů. Ty se pak v praxi upravují pro konkrétní využití. O specifických implementacích použitých v doporučovacích systémech je v dostupné literatuře obtížné něco najít.

Obecně se nejčastěji tento přístup využívá jako jeden z algoritmů v hybridních systémech. (Zajímavé pokusy proběhly například v [6]) Je potřeba poměrně velké množství dat, aby se nalezlo dostatečné množství vzorů, které je pak možné využít. Použití komplikuje i to, že nákupní historie uživatele nemusí odpovídat žádnému běžnému vzoru a v tom případě tento systém není schopný doporučit žádnou položku. Proto se výstupy často kombinují s jiným systémem.

2.2.1 Definice a základní principy

R. Agrawal a R. Srikant definují problém hledání častých sekvencí takto:

> *„Given a database of sequences, where each sequence consists of a list of transactions ordered by transaction time and each transaction is a set of items, sequential pattern mining is to discover all sequential patterns with a user-specified minimum support, where the support of a pattern is the number of data-sequences that contain the pattern.“* ([5])

Tato definice je velmi obecná. Díky tomu je možné výsledné algoritmy aplikovat v různých oblastech, včetně například analýzy řetězců DNA. Pro účely této práce je vhodné si povšimnout několika věcí.

1. Sekvence jsou uspořádané podle času.

2. Skládají se z množin prvků. Díky tomu je problém definován velmi robustně. Zároveň to ovšem komplikuje používané algoritmy.

3. Proces hledání má jeden parametr - minimální podporu sekvence (*„minimum support“*). Jedná se o počet výskytů daného sekvence v databázi, který je potřebný k tomu, aby byla sekvence považována za častou.

Základní princip všech používaných algoritmů je podobný. V prvním kroku dojde ke generování kandidátů a poté se pro každého kandidáta určí podpora (počet výskytů dané sekvence v datech). Pokud je vyšší než minimální podpora, pak se daná sekvence uloží.

Pro omezení počtu vygenerovaných kandidátů lze využít tzv. Apriori principle. Ten říká, že pokud je určitá sekvence častá, pak jsou časté i všechny její podsekvence. Neboli pokud určitá sekvence není častá, pak žádná sekvence, kterou lze vytvořit jejím rozšířením, není častá.

2.2.2 Používané přístupy k hledání

Tato sekce obsahuje stručný přehled základních přístupů. Podrobný rozbor existujících algoritmů lze nalézt např. v [7].

2.2.2.1 Apriori algoritmy

První apriori algoritmy byly navrženy již v původním článku Agrawala a Srikanta. O rok později pak jejich rozvojem vytvořili první významný algoritmus v této oblasti nazvaný GSP [8].

Tento algoritmus postupuje iterativně. V každém kroku vytvoří nové sekvence zřetězením dvojic sekvencí z předchozí iterace a vytvoří nové kandidáty. Poté odstraní takové sekvence, v kterých nalezne podsekvenci, která nesplňuje minimální podporu. Algoritmus skončí, pokud během iterace neobjeví žádné nové časté sekvence.

V průběhu generování a ověřování kandidátů se používá datová struktura zvaná *„hash-tree“*. Díky tomu je tento algoritmus poměrně složitý na implementaci a v porovnání s novějšími algoritmy má horší výkon. Obsahuje ale možnosti, které tyto algoritmy nemají. Jedná se o tři funkce:

1. Umožňuje omezit čas mezi následujícími transakcemi. Pokud je mezera mezi nimi příliš krátká nebo příliš dlouhá, nezapočte se takový výskyt při počítání podpory vygenerované sekvence.

2. Transakce je definována pomocí posuvného okénka. Za jednu transakci se nepovažuje pouze jeden záznam v databázi, ale dojde ke sloučení všech, které proběhly v určitém intervalu.

3. Lze pracovat nejen se sekvencemi jednotlivých produktů, ale i sekvencemi kategorií produktů. K tomu je nutné, aby uživatel zadefinoval tzv.

taxonomii produktů. Taxonomie je stromová struktura, která reprezentuje příslušnost produktů do kategorií a případně i kategorií do abstraktnějších kategorií.

Postupně se ukázalo, že pro mnoho praktických využití tyto pokročilejší funkce nejsou nutné, případně je lze doplnit předzpracováním dat. Byly vytvořeny algoritmy, které poskytují vyšší výkon a navíc jim stačí přečíst databázi pouze jednou, což vede k dalšímu podstatnému zrychlení. Hlavními zástupci jsou SPAM [9] a SPADE [10]. K implementaci v této práci jsem zvolil SPAM, proto bude jeho princip popsán podrobněji v příští sekci.

Společnými rysy těchto algoritmů je to, že z tzv. horizontálního pohledu na data přechází na vertikální. Jinak řečeno, jejich vnitřní reprezentace dat je organizována z pohledu produktů a toho, v kterých transakcích se vyskytují. V případě GSP jsou data organizována z pohledu uživatelských sekvencí a transakcích v nich obsažených.

Princip generování kandidátů se také změnil. GSP využívalo ke generování nových sekvencí prohledávání do šířky, tyto algoritmy využívají prohledávání do hloubky. Pro delší sekvence se takový přístup ukázal být méně náročný.

Slabinou těchto algoritmů je to, že vyžadují všechna data načtená v paměti. Přesněji řečeno SPADE umožňuje rozdělit prohledávaný prostor na více částí a prohledávat je odděleně, SPAM vyžaduje všechna data dostupná zároveň. Vzhledem k tomu, že dlouhodobě klesá cena operační paměti, v dnešní době již není tento problém příliš kritický.

2.2.2.2 Pattern growth algoritmy

Apriori algoritmy mají v nejhorším případě exponenciální složitost. To je dáno tím, že v nich dochází ke generování kandidátů. Teprve poté se pro tyto kandidáty hledá podpora v datech.

Pattern growth algoritmy byly vyvinuty tak, aby neobsahovaly tento krok a díky tomu mají lepší algoritmickou složitost. Na velkých datech mohou být v některých případech výkonnější. Cenou za to je fakt, že jsou podstatně složitější na implementaci. Hlavními zástupci této kategorie jsou FREESPAN [11] a PREFIXSPAN [12].

Základní myšlenka je taková, že algoritmus projde databázi a nalezne podporu všech jednotlivých produktů. Poté se produkty seřadí podle jejich podpory od nejvyšší po nejnižší. Poté se postupně prochází jednotlivé transakce a produkty z nich se vkládají do stromové struktury, tak aby sekvence, které obsahují stejné produkty, byly ve stromě právě jednou. Nakonec se tato struktura prochází od listů směrem ke kořeni a získávají se z ní časté sekvence.

2.3 Vybrané algoritmy pro implementaci

2.3.1 SPAM

Tento algoritmus byl představen v roce 2002 [9]. Patří mezi Apriori algoritmy, využívá tedy klasickou strukturu těchto algoritmů. Pracuje rekurzivně, přičemž při každém zanoření dochází ke generování kandidátů a poté počítání jejich podpory. Do další úrovně se předají ti kandidáti, jejichž podpora je vyšší než zvolená minimální podpora.

Do své práce jsem si tento algoritmus vybral především díky jeho velmi dobrému výkonu. Zajímavé porovnání s algoritmem PREFIXSPAN, který je v literatuře také považován za velmi rychlý, lze nalézt zde [13]. Na datech použitých v tomto článku byl SPAM výrazně rychlejší.

2.3.1.1 Základní princip

Obecně pokud existuje sekvence o délce n a množina produktů, které jsou vhodné k prodloužení sekvence, o velikosti k, pak existuje k sekvencí o délce $n+1$, které se stávají kandidáty. Díky tomu má problém generování kandidátů stromovou strukturu. Na každé pozici v sekvenci je možné vyzkoušet určitý počet produktů a na základě toho, který je zvolen, existuje určitá skupina produktů pro další pozici. Tento proces je analogický k procházení stromu od kořene. V původním článku se strom nazývá *„lexikografický strom“* [9].

Vzhledem k obecné definici sekvence jako posloupnosti množin produktů, existují dva způsoby, jak vytvořit novou delší sekvenci. Nový produkt je možné přidat jako první prvek nové množiny na konec sekvence nebo jako další prvek poslední množiny v sekvenci. Díky tomu má lexikografický strom v každém uzlu dvě skupiny potomků (jednu pro produkty, které prodlužují sekvenci jako nová množina, druhou pro ty, které prodlužují sekvenci přidáním do poslední množiny).

Dvě metody prodlužování znamenají také dvě množiny produktů použitelných pro prodloužení. Během iterací algoritmu se obě množiny průběžné zmenšují díky aplikaci prořezávání, které je postavené na Apriori principu.

2.3.1.2 Příprava a reprezentace dat

Údaje o nákupech jsou obvykle uložené tak, že každá transakce je jeden záznam v databázové tabulce (znázorněno v 2.1). Nákup v tomto příkladu tvoří ovoce. Pro účely zpracování tímto algoritmem je nutné jednotlivé transakce seřadit podle zákazníka a podle času, v jakém proběhly (zde reprezentováno vzrůstajícím id). Výsledkem je uspořádaný seznam transakcí pro každého uživatele (znázorněno v 2.2).

SPAM pro reprezentaci sekvencí využívá bitmapy. V prvním kroku se pro každý produkt, který se objeví v databázi, vytvoří bitmapa, která obsahuje jeden bit pro každou proběhlou transakci (znázorněno v 2.3).

ID zákazníka	ID transakce	Množina produktů
1	1	{jablko, banán}
1	2	{pomeranč}
1	3	{hruška, třešeň}
2	4	{banán}
2	5	{hruška, třešeň}
2	6	{jablko}

Tabulka 2.1: Forma dat uložených v databázi

ID zákazníka	Sekvence
1	{jablko, banán}, {pomeranč}, {hruška, třešeň}
2	{banán}, {hruška, třešeň}, {jablko}

Tabulka 2.2: Forma dat použitá v algoritmu

Pozice v bitmapě jsou uspořádány podle id uživatele a pro každého uživatele vzestupně podle id transakce. Vzestupné uspořádání transakcí je důležité, protože reprezentuje časové uspořádání transakcí. Výskyt produktu v transakci je reprezentován bitem s hodnotou jedna.

Při generování kandidátů bitovými operacemi vznikají nové bitmapy, které reprezentují celou sekvenci. Platí, že počet bitů s hodnotou jedna odpovídá podpoře dané sekvence (včetně sekvence délky jedna, tedy jednoho produktu). To umožňuje velmi rychlé výpočty podpory.

IDZ	IDT	Jablko	Banán	Pomeranč	Hruška	Třešeň
1	1	1	1	0	0	0
1	2	0	0	1	0	0
1	3	0	0	0	1	1
1	4	0	1	0	0	0
1	5	0	0	0	1	1
1	6	1	0	0	0	0

Tabulka 2.3: Bitmapy reprezentující jednotlivé položky

2.3.1.3 Průběh iterace

V každé iteraci algoritmus přijme sekvenci a dvě množiny produktů, určené pro prodlužování sekvence oběma způsoby. Pro všechny varianty prodloužení se vygeneruje kandidát a určí se jeho podpora. Pokud je podpora vyšší než zadaná minimální podpora, kandidát se uloží jako častá sekvence a rekurzivně se na něj zavolá stejná funkce. Během průběhu této funkce zároveň dochází k sestavování množin produktů pro další iterace.

Ukládání výsledků se realizuje pomocí stromové struktury, do které tato rekurzivní funkce na začátku uloží produkt, kterým byla naposled prodloužena. Tím vznikne struktura podobná lexikografickému stromu. Průchodem tímto stromem od kořene do všech uzlů lze získat všechny časté sekvence nalezené během průběhu tohoto algoritmu. Kořen toho stromu nereprezentuje žádný produkt, ale umožňuje snazší práci s výslednou datovou strukturou.

Kapitola 3

Návrh

V rámci této kapitoly jsou myšlenky popsané v analýze rozpracovány a využity k návrhu frameworku. Dále je zde popsán navržený způsob využití nalezených častých sekvencí k doporučování.

3.1 Doporučování na základě sekvencí

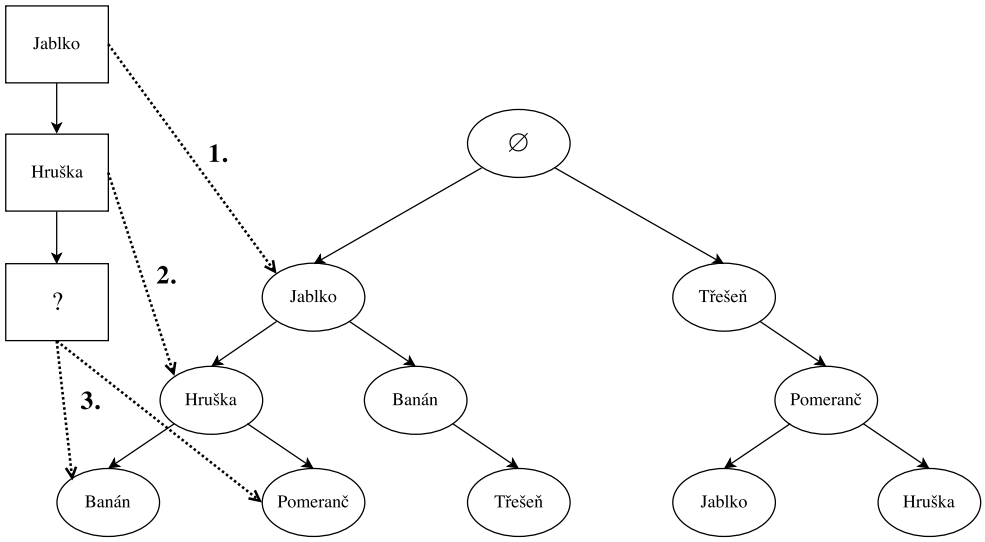
Hledání častých sekvencí vytvoří stromovou strukturu, která obsahuje všechny časté sekvence nalezené v datech. Tato struktura tvoří model, který se během doporučování využije k nalezení pravděpodobných produktů, které mohou uživatele zajímat.

Druhým vstupem doporučovacího algoritmu je uživatelská sekvence. Cílem algoritmu je nalézt určitý počet produktů, které by se mohly, s větší pravděpodobností než většina, objevit na následujícím místě v sekvenci.

3.1.1 Základní princip

Hlavní myšlenkou algoritmu je najít ve stromě častých sekvencí danou uživatelskou sekvenci a doporučit produkty, které tuto sekvenci prodlužují. Algoritmus začne v kořeni stromu a pokusí se najít mezi jeho potomky první produkt v uživatelské sekvenci. Poté se přesune do nalezeného uzlu a mezi jeho potomky hledá následující produkt v sekvenci. Takto pokračuje, dokud uživatelská sekvence obsahuje další produkty. Poté vrátí všechny potomky posledního uzlu. Proces je naznačen v obrázku 3.1.

Pokud se stane, že během procházení stromem následující produkt v sekvenci není mezi potomky aktuálního uzlu, algoritmus skončí. Tuto situaci je možné řešit různě. Nejjednodušším řešením je vrátit potomky aktuálního uzlu bez ohledu na to, že nenavazují přímo na poslední produkt v uživatelské sekvenci. Další možností je přeskočit produkt v sekvenci a pokusit se mezi potomky hledat ten následující.



Obrázek 3.1: Ilustrace mapování uživatelské sekvence na strom a nalezení doporučení

Další situací, která může nastat, je to, že aktuální uzel ve stromě již nemá žádné potomky. V takové chvíli není co doporučit. Vzhledem k tomu, že obě tyto situace v praxi nastávají často a výsledkem je buď žádné doporučení, nebo doporučení, které doopravdy neodpovídá uživatelské sekvenci, je vhodné zvolit robustnější řešení, které dokáže doporučit i za těchto okolností. Jedna z možností je popsána v následující sekci.

3.1.2 Metoda hlasování

Tento přístup spočívá v tom, že se doporučení nehledá ve stromě častých sekvencí přímo, ale během procházení se sbírají hlasy a výsledným doporučením se stane určitý počet produktů s největším počtem hlasů.

První možností je v každém uzlu stromu, který algoritmus navštíví, připočítat všem potomkům určitý počet hlasů. To umožní vytvořit doporučení i pokud algoritmus narazí na list předtím, než skončí uživatelská sekvence. Stejně tak i v situaci, kdy uzel nemá potřebného potomka a standardní procházení nemůže pokračovat.

Určení počtu hlasů, které by měl produkt obdržet, je možné provádět různými způsoby. Nejjednodušší možností je připočítat jeden hlas pokaždé, když se během průchodu daný produkt vyskytne. Tento přístup nicméně v určitém smyslu zahazuje výsledky hledání sekvencí, neboť nijak nezávisí na tom, v jakém místě se produkt vyskytl.

Jednoduchým rozšířením tohoto přístupu je určit počet hlasů na základě

hloubky uzlu ve stromu. Tím dojde k zvýhodnění produktů, které se vyskytnou v hlubších uzlech a tedy souvisí s větší částí uživatelské sekvence. Existují i sofistikovanější přístupy, které využívají další informace, které byly během hledání uloženy do uzlů stromu.

Mezi výskyty produktů ve větvi stromu existuje určitý vztah, protože produkty, které se objeví jako potomci uzlu, se zároveň objeví jako sourozenci uzlu. Proto není jednoduché přičítání hlasů vhodné. Lepší alternativou je uchovávat nejvyšší počet hlasů, které produkt získal kdekoliv během běhu algoritmu.

3.1.2.1 Hlasování pomocí podpory

Během hledání sekvencí je možné bez výrazné úpravy algoritmu ukládat do každého uzlu podporu sekvence, která v tomto uzlu končí. Tyto hodnoty je možné využít jako hlasy, které se přičtou danému produktu, pokud na něj algoritmus doporučování narazí. Hodnotu podpory je možné přičítat přímo, nebo nejdříve odečíst minimální podporu, čímž dojde ke zvýraznění rozdílů mezi sekvencemi, jejichž podpora se příliš neliší.

3.1.2.2 Hlasování pomocí důvěryhodnosti

Další hodnotou použitelnou pro hlasování je důvěryhodnost. Pro uzel v hloubce $n + 1$ určí se jako

$$\text{Confidence}_{n+1} = \frac{\text{Support}_{n+1}}{\text{Support}_n} \tag{3.1}$$

a vzhledem k tomu, že podpora rodiče je vždy vyšší než podpora potomka, výsledná hodnota se pohybuje v intervalu $(0, 1\rangle$. Smyslem této hodnoty je vyjádřit, jak často je předposlední položka v sekvenci následována tou poslední a tedy jak silná je implikace těchto dvou produktů.

Tuto myšlenku je možné rozšířit tak, aby byla více vázána na vlastnosti celé časté sekvence, především její délky a důvěryhodnosti každé položky v ní. Je možné sečíst důvěryhodnost všech položek až do současné pozice. Důvěryhodnost sekvence končící uzlem v hloubce $n + 1$ vypadá takto:

$$\text{Confidence}_{n+1} = \sum_{i=1}^{n} \frac{\text{Support}_{i+1}}{\text{Support}_i} \tag{3.2}$$

Dalším zajímavým údajem, který by bylo možné využít během doporučování, je čas mezi transakcemi v sekvencích, které tvořily podporu dané časté sekvence uložené ve stromu. Přidání času do algoritmu hlasování by mohlo vést ke zlepšení výsledků.

3.1.3 Určení délky použité sekvence

Strom častých sekvencí obvykle není příliš hluboký. Naopak sekvence aktivního uživatele může být poměrně dlouhá. Díky tomu, že hledání sekvencí pro-

bíhá na sekvencích, které pocházejí ze stejné databáze, jako ta, pro kterou vzniká doporučení, je téměř jisté, že značná část uživatelských sekvencí bude delší než výsledné časté sekvence.

Vzhledem k tomu, že proces doporučování závisí na hledání uživatelské sekvence ve stromu, pokud by algoritmus začal na začátku sekvence, ve velké části případů by se zastavil v listu stromu dříve než na posledním produktu sekvence. Tím by vzniklo doporučení ekvivalentní tomu, které by vzniklo pro zkrácenou sekvenci. To ale není účelem.

Je tedy vhodné začínat v uživatelské sekvenci pouze několik produktů od konce. Kolik je nutné zjistit praktickým testováním. Díky použití metody hlasování je možné spouštět proces opakovaně pro různě dlouhé části uživatelské sekvence a výsledné hlasy sloučit.

3.2 Vyhodnocení výsledků

Pro vyhodnocení úspěšnosti doporučovacího algoritmu je v této práci použita metoda křížové validace. Princip této metody spočívá v tom, že na začátku dojde k rozdělení vstupní množiny na dvě části:

1. Trénovací část, nad kterou poběží algoritmus hledání sekvencí a na základě těchto dat vznikne doporučovací model.

2. Testovací část, nad kterou poběží doporučovací algoritmus a na základě těchto dat dojde k vyhodnocení úspěšnosti.

Rozdělení do těchto dvou množin proběhne náhodně v určitém poměru, s tím že většina dat by měla směřovat do trénovací množiny. Náhodným výběrem se snažíme docílit toho, že obě množiny budou reprezentativní části vstupních dat.

Aby bylo možné algoritmy porovnat přímo, umožňuje framework spustit v jednom běhu nad stejným stromem různé algoritmy s různými parametry. Pro porovnání a vyhodnocení úspěšnosti budou použity dvě metriky.

3.2.1 Recall

Vyjadřuje úspěšnost doporučení. Určí se tak, že před zahájením doporučení pro danou sekvenci se skryje poslední položka této sekvence a doporučovací algoritmus se pokusí pro tuto položku vytvořit doporučení. Pokud se mezi doporučenými produkty objeví ten, který se na skryté pozici nachází, považuje se toto doporučení za úspěšné. Recall je poměr počtu úspěšných doporučení vůči všem.

3.2.2 Coverage

Vyjadřuje schopnost algoritmu doporučovat různé produkty (a ne například pouze bestsellery). Určí se jako poměr počtu produktů, které byly doporučeny pro libovolnou sekvenci, a počtu všech produktů, které se vyskytují v datech.

3.3 Základní struktura frameworku

Na nejvyšší úrovni se framework skládá z několika základních částí:

- Parser, který načítá data z databáze (případně z jiného zdroje) a převádí je do vnitřní reprezentace.
- Miner, v kterém je implementováno vyhledávání sekvencí. Tato část přijímá zákaznické sekvence a vrací model v podobě stromu častých sekvencí.
- Recommender, který obsahuje algoritmus doporučování pomocí stromu a jedné uživatelské sekvence.
- Cross validator, který rozděluje data na trénovací a testovací, a poté vyhodnocuje úspěšnost doporučení.
- Pomocné třídy, které slouží k obalení některých modulů, podpoře vyhodnocování a ukládání výsledků.

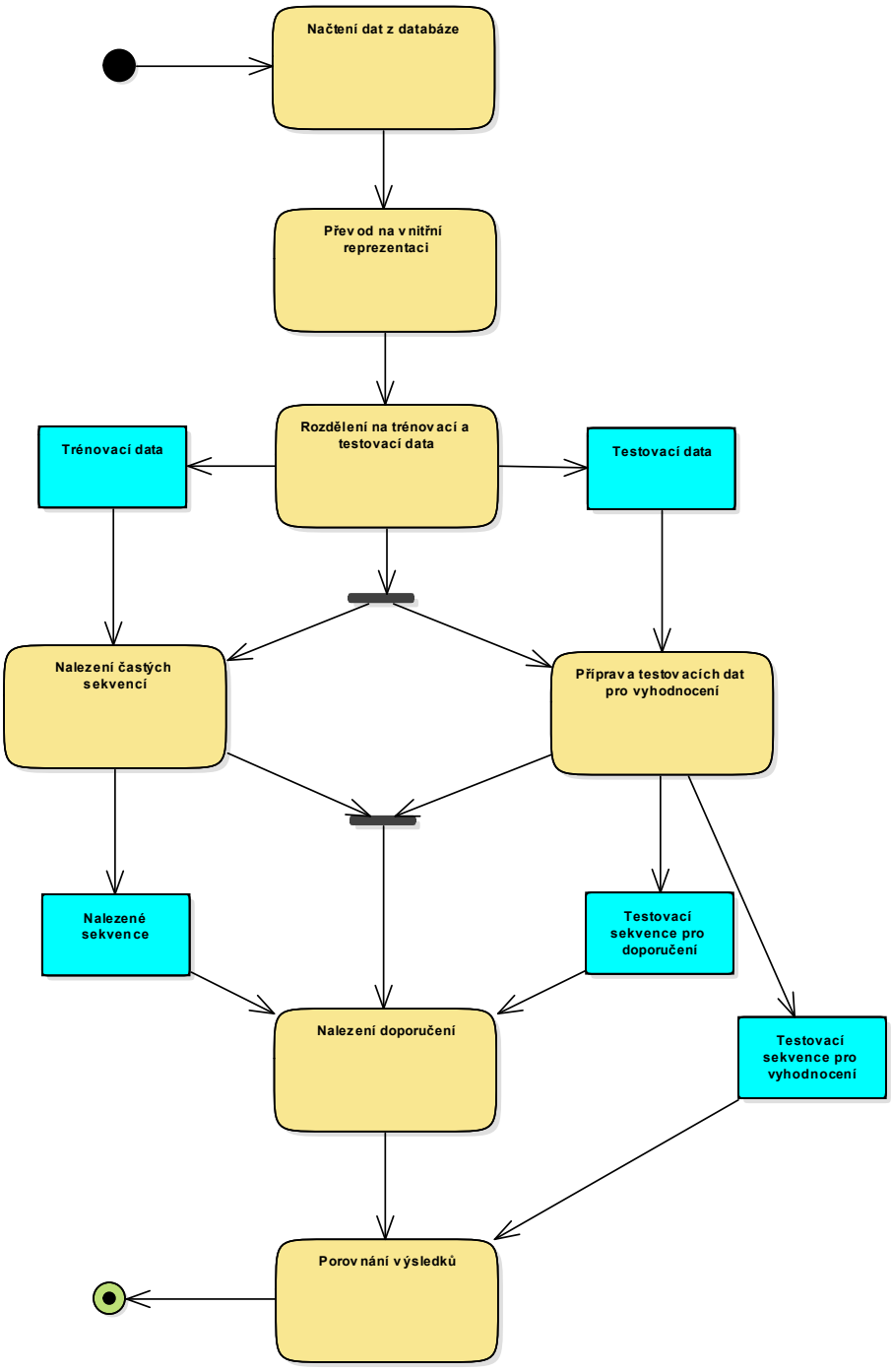
Jednotlivé části jsou obvykle reprezentovány rozhraním, které pak implementuje jedna nebo více tříd s konkrétními algoritmy. Princip zpracování a průchod jednotlivými částmi je znázorněn v diagramu 3.2.

3.4 Návrh tříd

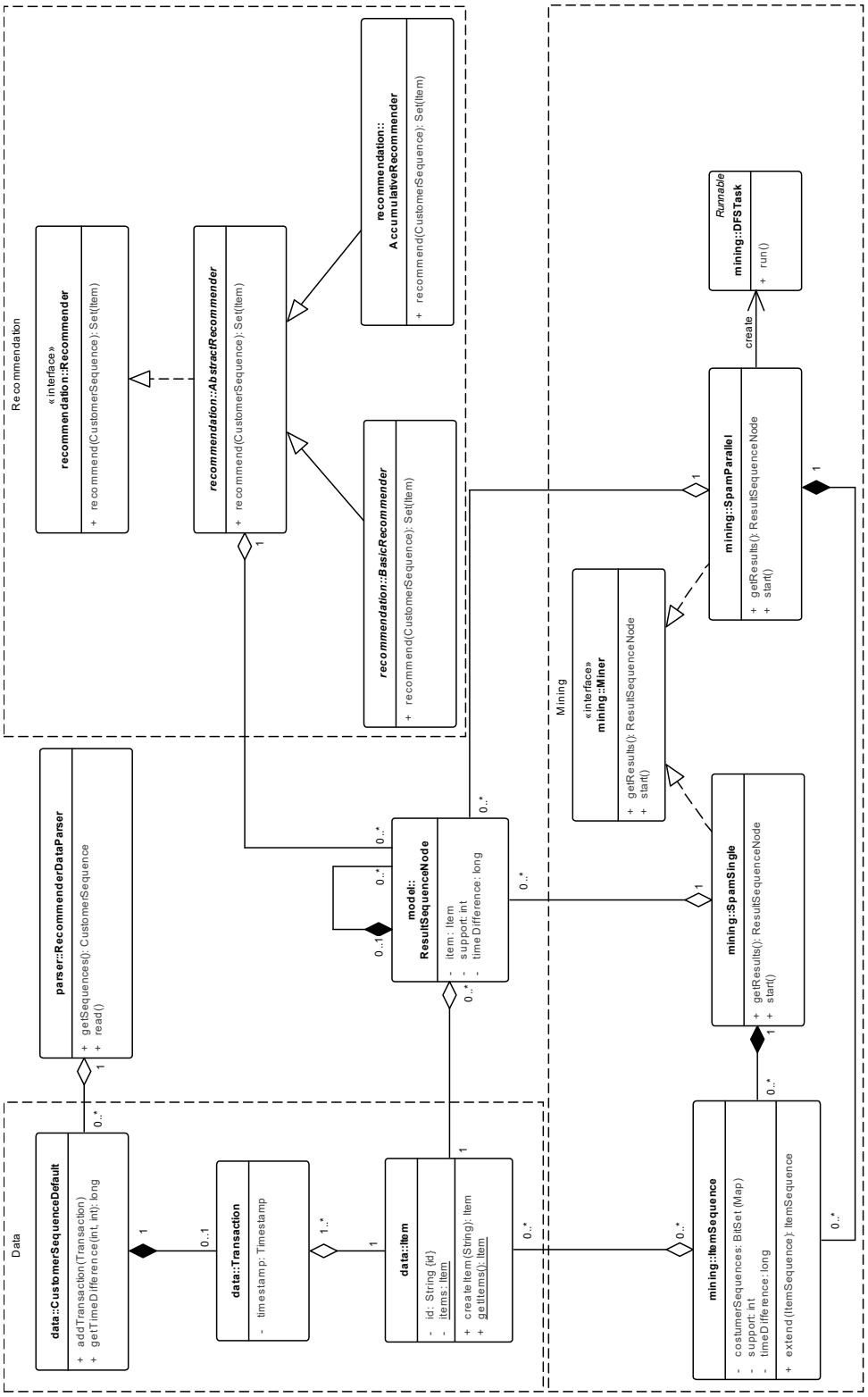
Základní struktura frameworku odpovídá problematikám popsaným v dřívějších kapitolách. Tato sekce je rozdělena přibližně podle jednotlivých balíčků, které odpovídají jednotlivým oblastem, které je nutné implementovat. Pro přehlednost a snazší vyjádření vztahů jsou hlavní třídy zakresleny v jednom diagramu 3.3. Naopak detaily implementace a pomocné třídy jsou především popsány a případné diagramy jsou k dohledání v příloze (od strany 47).

3.4.1 Načtení a příprava dat

Načítání dat zařizují třídy implementující rozhraní Parser. Jejich účelem je převést data z vnějšího zdroje do vnitřní reprezentace. V diagramu 3.3 je zakreslena třída RecommendationDataParser, která slouží k načítaní dat z databáze.



Obrázek 3.2: Diagram aktivit popisující průběh vyhodnocování ve frameworku



Obrázek 3.3: Diagram s nejdůležitějšími třídami

3.4.1.1 Třída CustomerSequence

Tato třída slouží k uložení zákaznické sekvence. Obsahuje list transakcí, které daný zákazník provedl a případné dodatečné informace. Zároveň umožňuje základní operace nad sekvencí, jako např. zjištění doby, která uplynula mezi dvěma transakcemi.

3.4.1.2 Třída Transaction

Ukládá informace o proběhlé transakci, především referenci na produkt, který zákazník zakoupil a čas nákupu.

3.4.1.3 Třída Item

Třída reprezentuje produkt, který se vyskytuje v alespoň jedné načtené transakci. Ukládá především identifikátor produktu a případné další informace. Instance této třídy jsou vytvářeny statickou metodou, což umožňuje zamezit duplikaci produktů během načítání.

3.4.2 Hledání sekvencí

Hledání častých sekvencí je realizováno třídami implementující rozhraní Miner. V praktické části je implementován algoritmus SPAM ve dvou variantách.

3.4.2.1 Třída SpamSingle

Jedná se o jednovláknovou implementaci algoritmu SPAM. Tato implementace generuje kandidáty pomocí rekurzivní funkce. Výstupem je strom častých sekvencí reprezentovaný objekty třídy ResultSequenceNode.

3.4.2.2 Třída SpamParrallel

Třída implementuje paralelní verzi algoritmu. Obsahuje pool vláken, do kterého se předávají objekty třídy DFSTask. Průběh jednotlivých kroků algoritmu je velice podobný rekurzivní verzi, ale díky tomu, že během výpočtu není nutná synchronizace, tato implementace je výrazně rychlejší.

3.4.2.3 Třída ItemSequence

Jedná se o vnitřní reprezentaci načtených dat určenou pro zpracování algoritmem SPAM. Třída obsahuje bitmapy a logiku nutnou k jejich využití pro reprezentaci sekvencí. Podrobnosti o této reprezentaci jsou vysvětleny v popisu algoritmu v sekci 2.3.1.

3.4.3 Doporučování

Doporučovací část frameworku je reprezentována rozhraním Recommender. Souvislosti s ostatními části jsou naznačeny v diagramu 3.3. Celá struktura tříd implementujících doporučování je zakreslena v diagramu 3.4.

3.4.3.1 Třída ResultSequenceNode

Třída slouží k vytvoření stromu častých sekvencí, který tvoří model, na jehož základě doporučování probíhá. Obsahuje referenci na produkt, dále podporu sekvence končící v daném uzlu a průměrný čas mezi výskyty tohoto a předchozího produktu v uživatelských sekvencích.

3.4.3.2 Třída AbstractRecommender

Abstraktní třída, která implementuje obecnou logiku a pomocné metody určené ke sčítání hlasů, nalezení produktů s nejvyšším počtem hlasů atd.

3.4.3.3 Třída BasicRecommender

Abstraktní třída implementující rekurzivní funkci na procházení stromu a sbírání hlasů pro nalezené produkty. Její podtřídy implementují různé metody určení počtu hlasů pro daný produkt.

Název třídy	Počet hlasů pro produkt
DepthBasedRecommender	Hloubka uzlu ve stromu
SupportBasedRecommender	Podpora sekvence končící v daném uzlu
ConfidenceBasedRecommender	Důvěryhodnost posledního kroku v sekvenci
CombinationBasedRecommender	Součin podpory a důvěryhodnosti

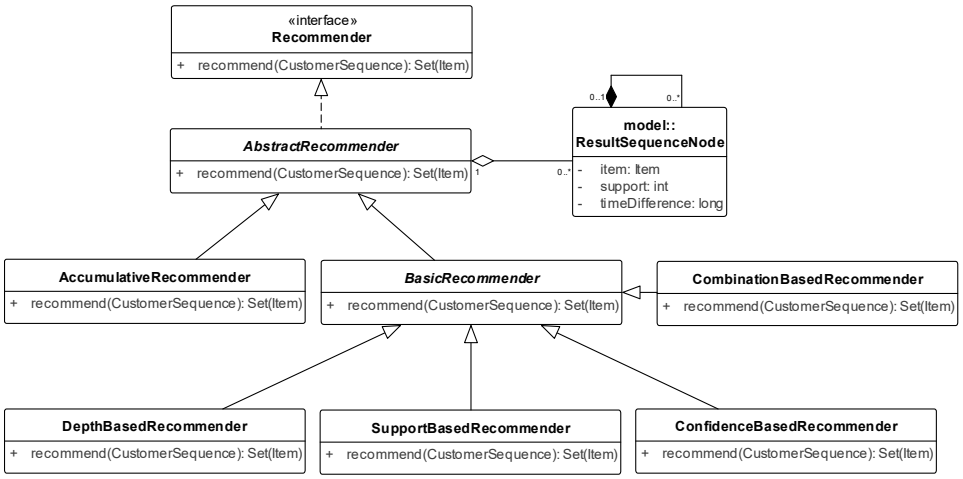
Tabulka 3.1: Výpočet počtu hlasů v různých podtřídách

3.4.3.4 Třída AccumulativeRecommender

Třída implementující složitější variantu rekurzivního procházení stromu, kde v průběhu procházení dochází ke sčítání důvěryhodnosti jednotlivých uzlů a počtem hlasů pro produkt v daném uzlu se stává tento součet.

3.4.4 Vyhodnocení úspěšnosti

Proces vyhodnocování je realizován pomocí několika tříd, které rozdělují data na trénovací a testovací a realizují sběr dat o doporučení a vyhodnocení výsledků. Vzhledem k tomu, že se jedná o pomocné třídy, diagram A.1 je umístěn v příloze na straně 47.



Obrázek 3.4: Diagram tříd implementující doporučování

3.4.4.1 Třída RecommendationTester

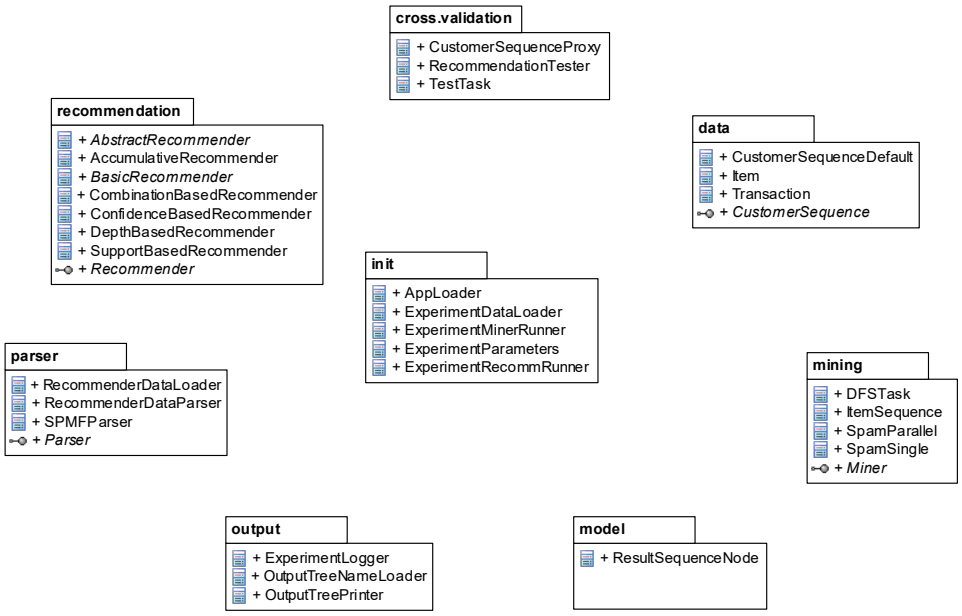
Třída slouží k provedení doporučení na testovací množině a sběru klíčových metrik. Doporučování samotné je paralelní. Pro každou sekvenci vznikne nový objekt TestTask, který je předán poolu vláken v této třídě. Z proběhnutých pokusů se nakonec vypočítají výsledné metriky.

3.4.4.2 Třída TestTask

Třída implementuje rozhraní Runnable a obsahuje vše potřebné pro vygenerování doporučení pro jednu uživatelskou sekvenci. Využívá se zde pomocná třída CustomerSequenceProxy, která obaluje jiný objekt implementující CustomerSequence a umožňuje získat odděleně sekvenci určenou pro doporučování a poslední položku sekvence, kterou je nutné z doporučování vyjmout a využít až při ověřování úspěšnosti.

3.4.5 Balíčky

Rozdělení frameworku do balíčků znázorňuje diagram 3.5. Tato sekce popisuje účel jednotlivých balíčků. Podrobnosti o důležitých třídách jsou rozepsány v předchozích sekcích.



Obrázek 3.5: Diagram balíčků a tříd frameworku

3.4.5.1 Balíček init

Obsahuje pomocné třídy, které slouží k inicializaci programu a třídy, které obalují jednotlivé části frameworku a umožňují snazší úpravy parametrů na nejvyšší úrovni. Diagram A.2 těchto pomocných tříd je v příloze na straně 48

3.4.5.2 Balíček cross.validation

Obsahuje třídy, které slouží k rozdělení načtených dat na trénovací a testovací část. Dále obsahuje pomocné třídy, které slouží k vyhodnocování a umožňují paralelní běh doporučování nad testovací množinou uživatelských sekvencí.

3.4.5.3 Balíček data

Obsahuje třídy reprezentující načtená data, tedy uživatelské sekvence, jejich jednotlivé položky a produkty. Tyto třídy pak tvoří vstupy velké části procesů v tomto frameworku.

3.4.5.4 Balíček mining

Obsahuje dvě implementace algoritmu na hledání častých sekvencí (standardní rekurzivní a paralelní). Kromě toho obsahuje pomocnou třídu, která obaluje vnitřní reprezentaci načtených dat použitou v algoritmu.

3.4.5.5 Balíček model

Obsahuje pouze jednu třídu, která se používá k vytvoření stromu častých sekvencí, který je výstupem procesu jejich hledání.

3.4.5.6 Balíček output

Obsahuje pomocné třídy, které umožňují ukládat nalezené sekvenci a výsledky vyhodnocování.

3.4.5.7 Balíček parser

Obsahuje třídy, které slouží k načítání dat z databáze nebo ze souboru a převodu načtených údají o sekvencích do vnitřní reprezentace.

3.4.5.8 Balíček recommendation

Obsahuje implementaci doporučovacího algoritmu v několika různých variantách, které se liší především metodou počítání hlasů.

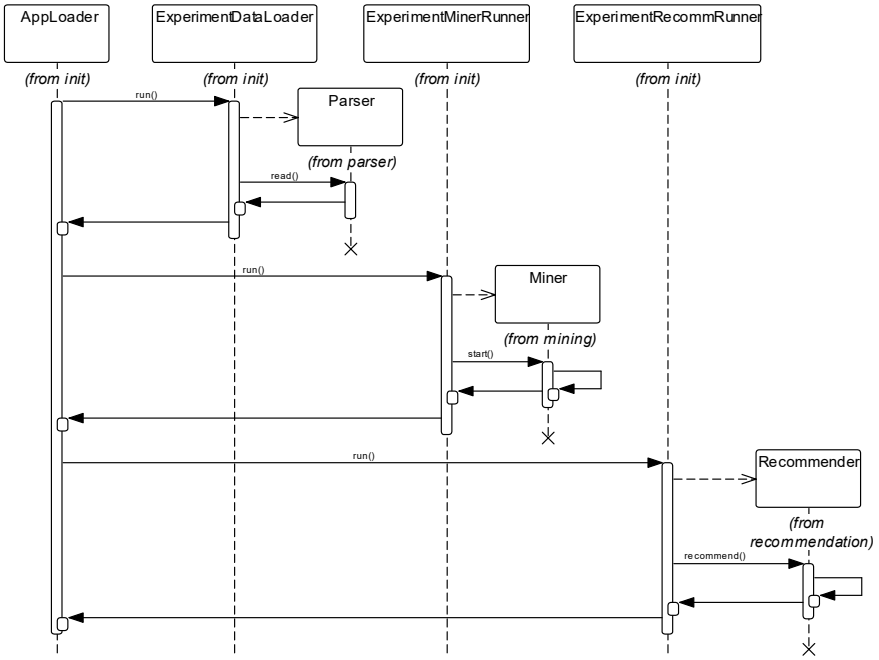
3.5 Běh frameworku

Pořadí provádění operací odpovídá logice popsané v předchozích sekcích. Framework se skládá ze třech hlavních částí, které se spouští postupně a zpracovávají výstupy předchozích částí. Průběh je znázorněn v diagramu 3.6. Každá z těchto částí je na nejvyšší úrovni obalena jednou třídou, což umožňuje snadné úpravy parametrů jednotlivých průběhů. Jednotlivé části jsou:

1. Načtení dat, které je na nejvyšší úrovni obaleno třídou ExperimentDataLoader. V této fázi dojde k načtení dat z vnějšího zdroje do vnitřních datových struktur.

2. Hledání sekvencí, kde dojde k vytvoření modelu ve formě stromu častých sekvencí. Na nejvyšší úrovni je tato část obaleno třídou ExperimentMinerRunner.

3. Vyhodnocení doporučovacího algoritmu, kde dojde ke spuštění doporučovacího algoritmu pro testovací množinu sekvencí a to opakovaně s různými parametry. Z těchto pokusů se poskládají a uloží výsledky. Tato část je obalena třídou ExperimentRecommRunner.



Obrázek 3.6: Sekvenční diagram popisující základní běh frameworku

Kapitola 4

Implementace

4.1 Implementace algoritmu pro hledání sekvencí

Pro použití v doporučování není nutné využívat plný algoritmus SPAM určený pro sekvence množin. Stačí pracovat pouze se sekvencemi jednotlivých položek. Díky tomu není nutné využívat prodlužování sekvencí přidáním do poslední množiny. Algoritmus se tím značně zjednoduší.

Původní algoritmus, tak jak byl popsán v originálním článku [9], předpokládá pro každý produkt bitmapu o délce rovné počtu všech transakcí v databázi. To je v případně skutečné databáze s miliony transakcemi, kde většina produktů se vyskytuje pouze v několika z nich, velmi neefektivní.

V této práci se využívá upravená struktura, která je tvořena tabulkou, kde klíčem je uživatel a hodnotou je bitmapa. Tato bitmapa je maximálně délky rovné počtu transakcí daného uživatele a bity v ní reprezentují, jestli se daný produkt nachází v dané transakci.

Položka v tabulce se vytvoří pouze, pokud daný uživatel koupil daný produkt. Bitmapu v praxi stačí omezit na délku rovnou indexu nejvyšší transakce, v které se produkt vyskytuje. Vzhledem k tomu, že většina produktů je provázána pouze s malou částí uživatelů a pouze malá část transakcí každého z těchto uživatelů obsahuje daný produkt, výsledkem je zásadní úspora paměti.

Výrazného vylepšení výkonu lze dosáhnout dalšími úpravami. Vzhledem k tomu, že doporučovací algoritmus je schopen využít pouze strom o omezené hloubce, je možné omezit hloubku rekurze a tím docílit výrazné úspory času i paměti, pokud vstupní data obsahují dlouhé sekvence.

Díky tomu, že rekurzivní funkce, která provádí prodloužení sekvence, nezávisí na výsledku hlubších vrstev rekurze, je možné algoritmus upravit do paralelní podoby.

4.2 Paralelizace algoritmů

Pro urychlení výpočtu probíhá hledání sekvencí i generování doporučení paralelně. Pro každou část se vytvoří pool vláken a proces je rozdělen na velký počet částí, které není nutné synchronizovat.

V případě hledání sekvencí je použit téměř stejný kód jako v původní rekurzivní funkci, pouze po nalezení prodloužené kandidátní sekvence dojde místo k opětovnému zavolání funkce k vytvoření nové objektu, který se zařadí do fronty pro zpracování. Díky tomu dochází k vytváření více větví výsledného stromu najednou.

Doporučování je paralelizováno na úrovní jednotlivých testovacích uživatelských sekvencí. Měřené údaje se sbírají do atomických proměnných a nakonec dojde k vyhodnocení a uložení.

4.3 Použité technologie

- Celý framework je implementován v jazyce Java. Vzhledem k současnému rozšíření je použita verze 8.
- K sestavení je použit Maven.
- Pro účely logování framework využívá knihovnu log4j.

4.4 Ukládání výsledků

4.4.0.1 Ukládání stromu

I když hlavním účelem frameworku je vyhodnotit úspěšnost algoritmů doporučování, první krokem tohoto procesu je nalézt časté sekvence. Jedná se poměrně dlouhý proces, který je náchylný na kvalitu vstupních dat i implementační chyby. Proto jsem během vývoje vytvořil pomocné třídy, které nalezený strom s častými sekvencemi uloží do souboru.

Pro uložení je využit formát GraphViz. Díky tomu je možné výstup vizualizovat v běžně dostupných nástrojích (např. Gephi, použit pro obrázek A.3 na straně 49) a ověřit smysluplnost nalezených sekvencí.

Jedná se o jednoduchý formát sloužící k uložení obecných grafů. Jak je vidět v 4.1, celý graf je uložený v jednom bloku typu digraph (orientovaný graf). Každý uzel grafu je nutné nejdříve deklarovat. Poté se již může libovolně využívat při deklaraci hran.

Každý uzel má vygenerované id a přiřazený popisek produktu z databáze. Každá hrana je zapsána pomocí jejích krajních uzlů a má přiřazenou váhu a vlastní popisek odpovídající její podpoře. V ukázce 4.1 se jedná o sekvence episod seriálů.

```
digraph tree {
        "1" [label="ROOT"]
        "444" [label="Unknown"]
        "1" -> "444" [label="770" weight="1.0"]
        "200" [label="Unknown"]
        "1" -> "200" [label="1315" weight="1.0"]
        "879" [label="The Relationship Diremption:20"]
        "1" -> "879" [label="1318" weight="1.0"]
        "13060" [label=""The Status Quo Combustion":24"]
        "879" -> "13060" [label="992" weight="13.047031887827568"]
        "13058" [label="The Proton Transmogrification:22"]
        "879" -> "13058" [label="1003" weight="12.195062253010121"]
        "13069" [label=""The Status Quo Combustion":24"]
        "13058" -> "13069" [label="811" weight="12.720224730484302"]
        "13068" [label="The Gorilla Dissolution:23"]
        "13058" -> "13068" [label="811" weight="11.821159883608734"]
}
```

Obrázek 4.1: Ukázka uložení stromu ve formátu GraphViz

4.4.0.2 Ukládání metrik

Framework je navržen tak, že v jednom běhu dojde k vyzkoušení různých algoritmů s různými parametry. Každá kombinace parametrů se vyhodnocuje. Použitý algoritmus, parametry a vypočtený recall a coverage se nakonec uloží jako jedna řádka do souboru ve formátu csv.

To umožní zpracování v prakticky libovolném tabulkovém editoru nebo softwaru pro práci s daty. Například pro vykreslování grafů a porovnávání různých variant konfigurace jsem v této práci používal RapidMiner Studio.

4.4.1 Prostředí pro vývoj a testování

Pokusy probíhaly na serveru s 12 jádrovým procesorem a 128 GB paměti. Velikost haldy JVM byla omezena na 90GB a jako garbage collector byl použit G1. Ostatní nastavení byla ponechána na výchozích hodnotách. Jako JVM byla použita 64-bitová verze Oracle Hotspot, konkrétně verze 1.8.0_121.

Kapitola 5

Testování

5.1 Pokusy na skutečných datech

Pro testování byly k dispozici dvě databáze pocházející ze skutečných webových služeb. V každé sekci se používají data nasbíraná během jednoho pokusu, při kterém byl vygenerován jeden strom častých sekvencí a následně na něm probíhalo doporučování s různými parametry.

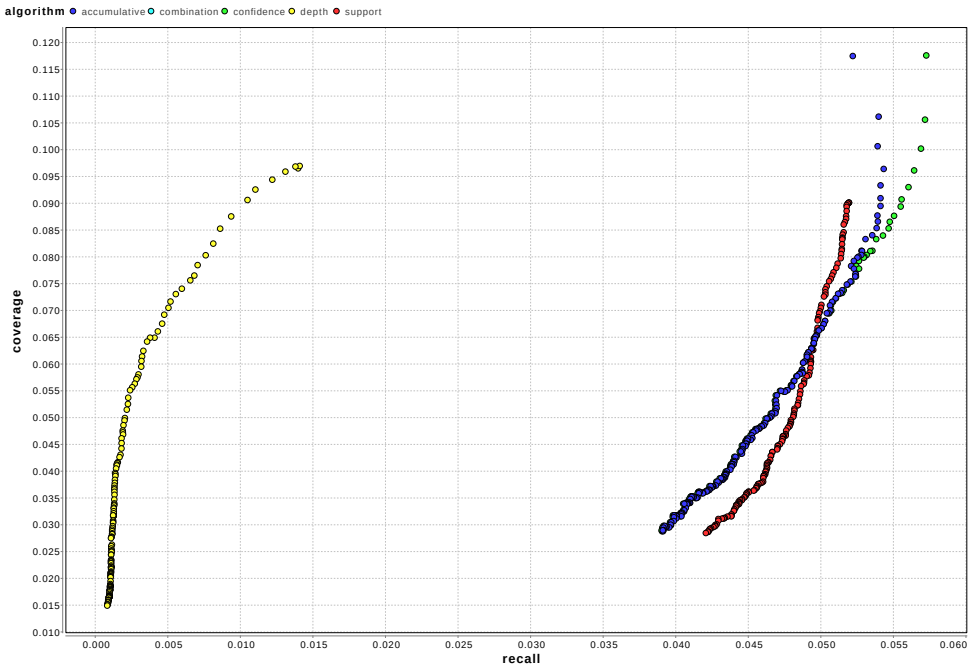
5.1.1 E-shop s dětským zbožím

První databáze pochází z menšího e-shopu, který se zaměřuje především na rodiče s malými dětmi. Databáze obsahuje velký počet uživatelů, kteří obvykle koupili pouze několik různých produktů (průměrně 2.9 produktů na jednoho uživatele). Díky tomu databáze obsahuje převážně krátké sekvence.

Velikost databáze umožňuje provést pokus s velmi malou podporou a pokusit se najít hranici, kde se algoritmus začne přeučovat. Zároveň je možné povolit i hlubší strom a zkoumat význam delších sekvencí na kvalitu doporučování.

Parametry pokusu:

- Minimální podpora: 0.00002 - 0.005
- Počet doporučených produktů: 12
- Maximální hloubka stromu: 2 - 4
- Počet transakcí: 445 732
- Počet sekvencí: 154 957
- Počet uzlů ve stromu: 20 miliónů



Obrázek 5.1: Hodnoty recall a coverage pro různé algoritmy. Je vidět malá úspěšnost algoritmu využívající pouze hloubku. Ostatní se chovají velmi podobně.

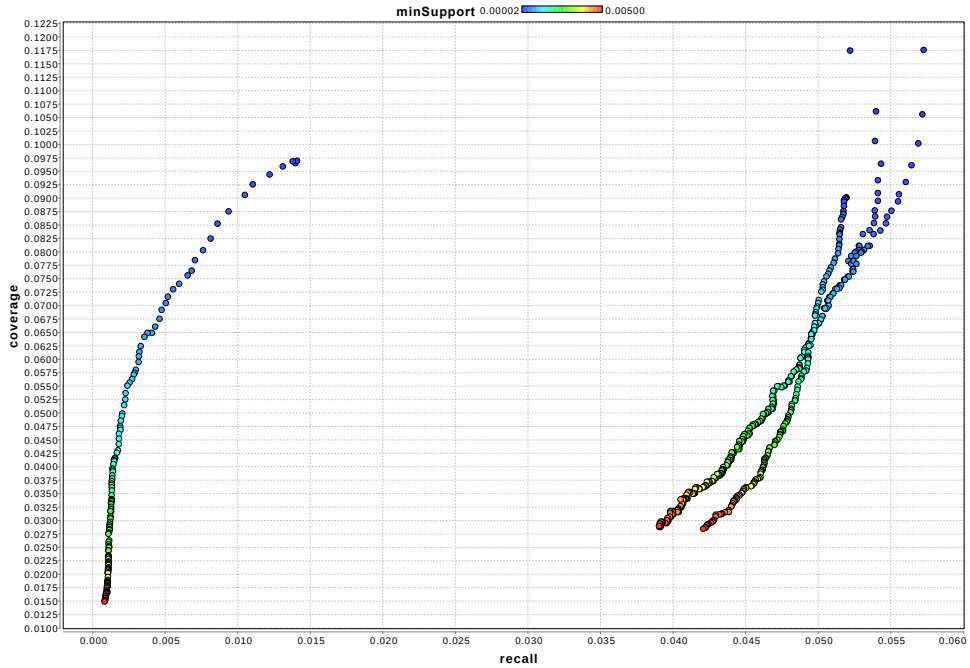
5.1.1.1 Srovnání algoritmů

Popis jednotlivých metod hlasování použitých v tomto pokusu lze nalézt v tabulce 3.1 na straně 23. V grafu 5.1 je vidět porovnání jednotlivých variant z hlediska coverage a recall metrik.

Algoritmus využívající pouze hloubku stromu (neboli délku časté sekvence) má na první pohled nejhorší výsledky. Je to dáno tím, že využívá pouze velmi malou část dostupných informací a také tím, že malá hloubka stromu (v tomto grafu je hloubka stromu 4) v praxi znamená, že mezi různými sekvencemi je malý rozdíl. Algoritmus je schopný doporučovat podobně různorodé produkty jako ostatní (má podobný coverage), ale recall je podstatně menší.

Ostatní algoritmy dosahují velmi podobných výsledků. Z grafů 5.1 a 5.2 je vidět, že algoritmy využívající důvěryhodnost a akumulovanou důvěryhodnost se chovají téměř stejně, pouze pro nejnižší podporu se od sebe mírně liší.

Algoritmus pokoušející se kombinovat důvěryhodnost a podporu na těchto datech nefungoval téměř vůbec. Zdá se, že tato jednoduchá kombinace dvou metrik nemá příliš dobré vlastnosti.



Obrázek 5.2: Hodnoty recall a coverage v závislosti na minimální podpoře. Je vidět vzrůstající tendence obou metrik s klesající minimální podporou. V pravém horním rohu došlo k mírnému snížení recallu naznačující přeučení.

5.1.1.2 Hranice přeučení

Přestože při tomto pokusu byla minimální podpora nastavena na co nejnižší možnou hodnotu (pro nižší hodnotu byl již výpočet příliš náročný), algoritmy vykazují pouze malé známky přeučení.

V případě algoritmů založených na hloubce a podpoře nedochází k přeučení vůbec. V případě algoritmů založených na důvěryhodnosti došlo k mírnému přeučení pro nejnižší hodnoty minimální podpory. V grafech 5.1 a 5.2 se to projevilo tak, že hodnota recallu mírně klesla.

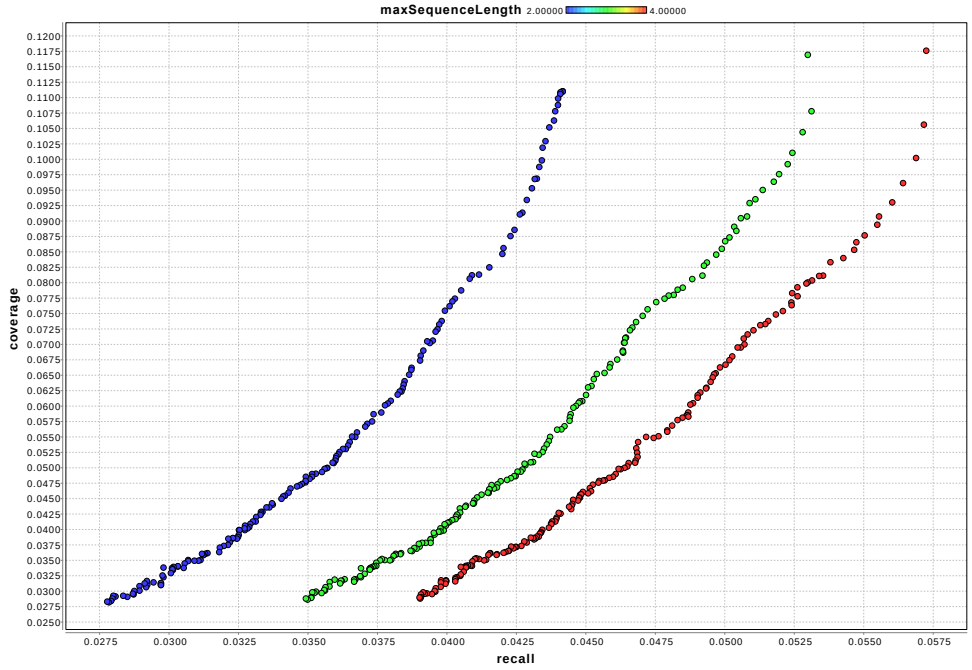
To odpovídá očekávání. Se snižující se podporou roste počet a různorodost natěžených sekvencí. Dostávají se do nich méně oblíbené produkty. Díky tomu, že sekvence obsahují více produktů, dojde i k doporučení více produktů a roste tedy coverage. Od určité hranice jsou již tyto produkty příliš nepopulární a šance, že je někdo koupil je malá. To nakonec způsobí snížení recallu.

Vzhledem k tomu, že k přeučení začalo docházet až na hodnotách, kterých algoritmus využívající podporu vůbec nedosáhl, výsledek se v tomto směru zdá být konzistentní.

5.1.1.3 Vliv měněných parametrů na výsledek

Graf 5.3 ukazuje porovnání výsledků algoritmu využívajícího důvěryhodnost pro různé hloubky stromu častých sekvencí. Výsledky naznačují, že i když původní data obecně neobsahují dlouhé sekvence, při prodloužení ze základní sekvence délky dva dochází ke značnému zlepšení.

Důvody jsou pravděpodobně dva. Doporučování s hlubším stromem zároveň využívá delší část uživatelské sekvence. Dále pravděpodobně dochází k tomu, že hlubší strom obsahuje relativně malý počet častých delších sekvencí, které se ovšem vyskytují častěji než většina a umožní zlepšit doporučení pro některé uživatele. Výsledky byly podobné i pro další algoritmy.



Obrázek 5.3: Hodnoty recall a coverage v závislosti na povolené hloubce stromu. Je vidět, že i přesto, že data neobsahují velký počet delších sekvencí, povolení delších sekvencí zlepšuje výsledky.

5.1.2 Online videotéka

Druhá databáze pochází z webové služby, která umožňuje předplatitelům sledovat filmy a seriály. Tato data obsahují výrazně delší sekvence než data z e-shopu v předchozí sekci.

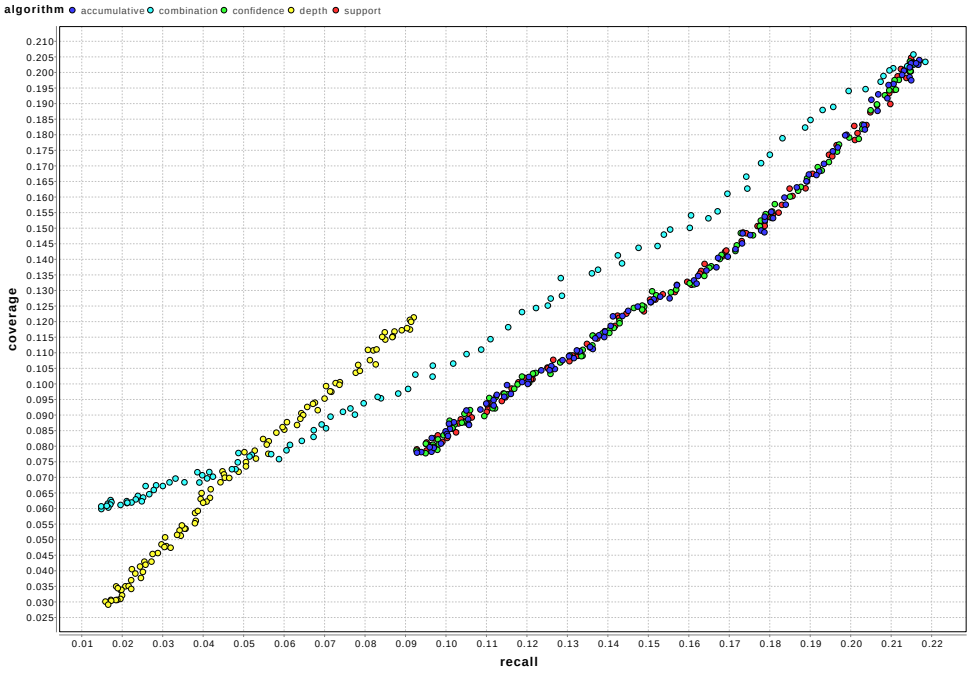
Vzhledem k velikosti databáze není prakticky možné vytvářet hluboké stromy při nízké podpoře, proto jsem pro účely toho pokusu omezil hloubku

stromu na 2. Časté sekvence tedy obsahují nejvíce tři produkty (Pro účely doporučování je první produkt v implementaci v hloubce 0).

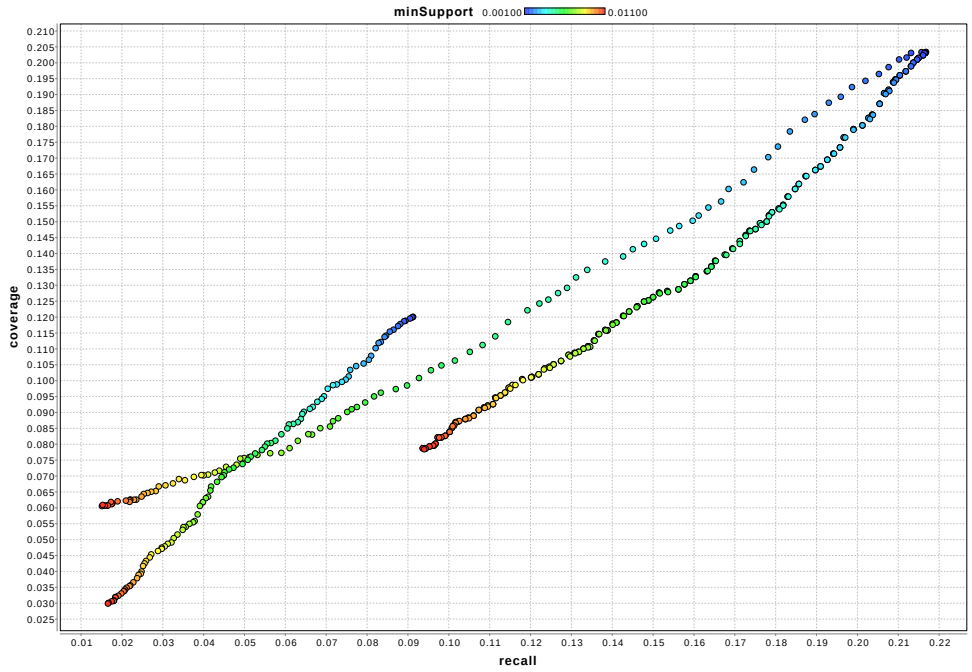
Parametry pokusu:

- Minimální podpora: 0.001 - 0.01
- Počet doporučených produktů: 2 - 15
- Maximální hloubka stromu: 2
- Počet transakcí: 5 000 000
- Počet sekvencí: 66 373
- Počet uzlů ve stromu: 283 miliónů

5.1.2.1 Srovnání algoritmů



Obrázek 5.4: Hodnoty recall a coverage pro různé algoritmy. Je vidět slabší výsledek algoritmu využívající hloubku. Kombinovaný algoritmus vykazuje poměrně velký rozptyl hodnot. Ostatní algoritmy se chovají stejně. Pro jejich rozlišení byl použit mírný jitter.



Obrázek 5.5: Hodnoty recall a coverage v závislosti na minimální podpoře. Je vidět vzrůstající tendence obou metrik s klesající minimální podporou pro všechny algoritmy.

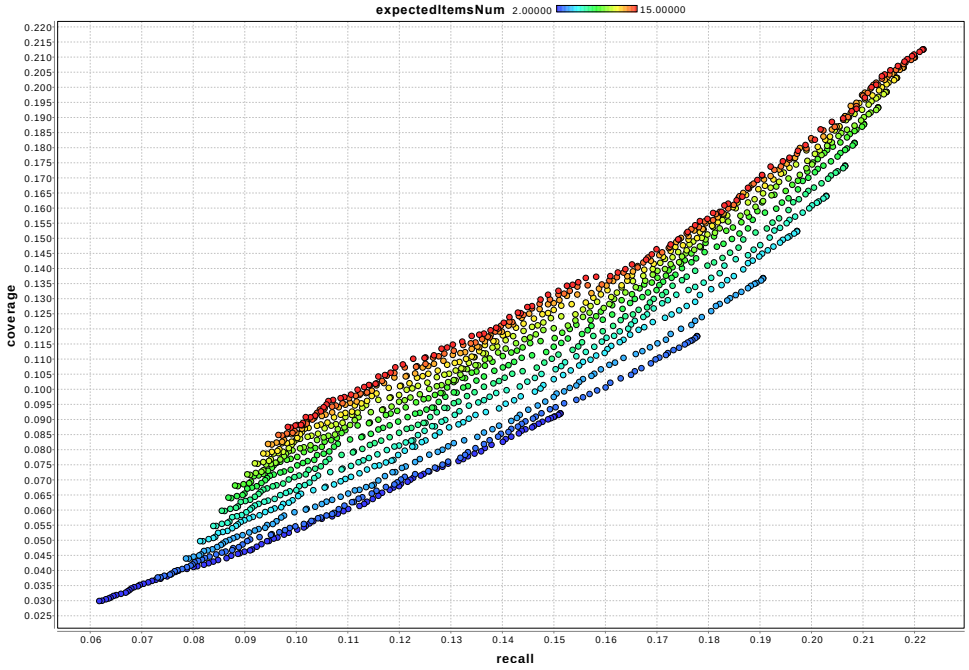
Jak je vidět v grafech 5.4 a 5.5, chování všech implementací doporučovacího algoritmu v tomto pokusu je podobné. Nejhorších výsledků dosáhl algoritmus založený pouze na hloubce stromu. To odpovídá očekávání, protože využívá nejméně informací uložených ve stromu. Navíc při maximální hloubce stromu dva bude velká část sekvencí stejně dlouhá.

Algoritmus kombinující podporu a důvěryhodnost se na těchto datech choval podobně jako ostatní algoritmy, proto je zahrnut i v grafech. Pro nejnižší minimální podporu jsou jeho výsledky mezi nejlepšími, se zvyšující se podporou ale rychle klesají.

Algoritmy využívající podporu, důvěryhodnost a akumulovanou důvěryhodnost se v tomto pokusu chovaly stejně a dosáhly nejlepších výsledků. Stejné chování přístupu založeném na důvěryhodnosti posledního kroku sekvence a přístupu založeném na akumulované důvěryhodnosti bylo očekávané. Důvodem je to, že při hloubce stromu použité v tomto pokusu se oba algoritmy chovají stejně.

Stejné chování algoritmu založeném na podpoře je pravděpodobně způsobeno tím, že v mělkém stromu se rozdíl mezi podporou samotnou a důvěryhodností stírá.

5.1.2.2 Vliv měněných parametrů na výsledek



Obrázek 5.6: Hodnoty recall a coverage v závislosti na počtu doporučených produktů. Je vidět posun křivky směrem k vyšším hodnotám s vzrůstajícím počtem doporučených produktů.

Jedním z parametrů doporučování je i počet očekávaných doporučených produktů. Tento parametr nemá přímo vliv na kvalitu doporučení, nicméně ovlivňuje hodnotu výsledných metrik.

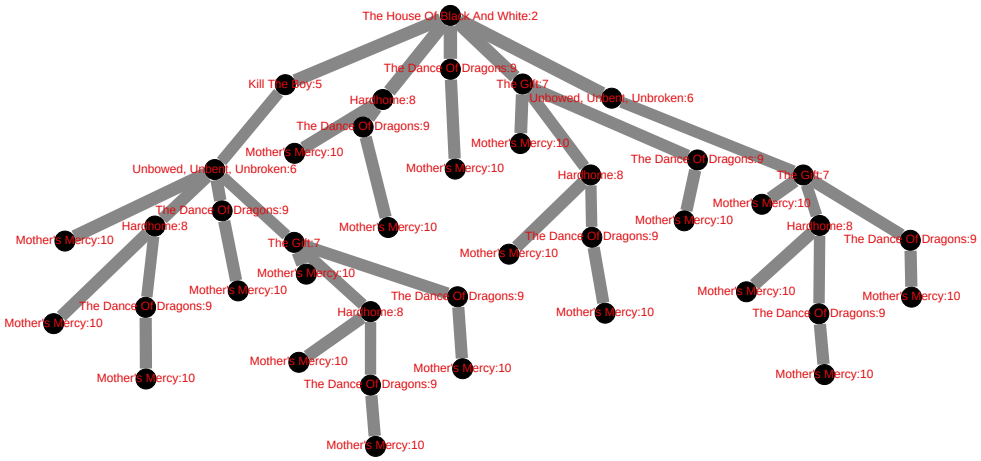
Graf 5.6 ukazuje, jak se liší hodnoty recallu a coverage pro různý počet doporučených produktů. Jak se dalo očekávat, vyšší počet produktů vede k vyšším hodnotám obou metrik.

I přesto, že počet produktů se mění zásadně (graf je v rozsahu od 2 do 15), hodnoty obou metrik se výrazně nemění. Výjimkou jsou značné rozdíly mezi 2 až 5 produkty, poté se přínos vyššího počtu produktů začíná rychle snižovat.

5.1.3 Příklad výsledných sekvencí

V obrázku 5.7 je vidět malá část stromu vzniklého při hledání častých sekvencí. Popisky uzlů obsahují název episody a pořadí této episody v rámci série. V tomto případě se jedná o epizody páté série seriálu Hra o Trůny.

V obrázku není zakreslen kořen stromu častých sekvencí, který je spíše implementačním detailem a umožňuje pracovat se všemi sekvencemi najednou.



Obrázek 5.7: Zjednodušený strom častých sekvencí pro jednu epizodu seriálu. Sekvence pokračují směrem dolů a je vidět, že za sebou následují epizody v sérii.

Nereprezentuje však žádný produkt. Bez něj by pro každý častý produkt vznikl samostatný strom. V tomto případě se jedná o zjednodušený strom pro epizodu The House Of Black And White.

V obrázku je vidět, že dle očekávání čísla epizod v sekvenci postupně rostou. Další efekt, který je důsledkem principu hledání sekvencí, je to, že stejné podstromy se vyskytují ve stromu vícekrát. Je to tak proto, že v každé sekvenci je možné vynechat libovolný produkt a sekvence bude stále častou.

Pro zajímavost je celý strom, z kterého tento příklad pochází, zakreslen v obrázku A.3 na straně 49 v příloze.

5.2 Porovnání výsledků

Z grafů 5.1 a 5.4 je vidět, že na databázi online videotéky byly hodnoty metrik výrazně vyšší než v případě e-shopu. Tento rozdíl je pravděpodobně způsoben tím, že data o sledování seriálů mají typicky sekvenční charakter, jelikož lidé obvykle sledují jednotlivé díly seriálu postupně.

Stejný efekt nejspíše způsobil i to, že všechny algoritmy se na datech z videotéky chovaly téměř stejně, zatímco na datech z e-shopu byly výraznější rozdíly v tom, jak se chování jednotlivých algoritmů liší s měnící se minimální podporou.

5.3 Škálovatelnost hledání sekvencí

Pro malou databázi dat z eshopu, v které se nevyskytuje příliš sekvencí, trvalo hledání samotné na použitém hardwaru cca. 10 minut. Databáze z online videotéky je výrazně větší a je možné na ní vyzkoušet závislost doby na množství vstupních dat.

Pokus popsaný v předchozí sekci byl proveden na 5 000 000 transakcí s nejnižší možnou minimální podporou (0,001) a hledání trvalo 1,5 hodiny. Při spuštění na 10 000 000 transakcí se srovnatelnou minimální podporou trvalo cca. 3,5 hodiny.

Bohužel je tyto čísla obtížné interpretovat, protože průběh hledání ovlivňuje více různých jevů. Zaprvé není možné pro různý počet transakcí použít stejnou minimální podporu. To je způsobené tím, že při načtení většího počtu transakcí značná část přidaných transakcí prodlouží již existující uživatelské sekvence.

Protože počet podsekvencí v sekvenci je exponenciálně závislý na délce sekvence, počet nalezených častých sekvencí se začíná rychle zvyšovat. Brzy dojde k tomu, že počet sekvencí již není zvládnutelný, proto je při zvýšení počtu transakcí nutné zvýšit i minimální podporu.

Dále do tohoto procesu vstupují další vlivy, které způsobují méně výrazné změny doby běhu tím, že ovlivňují jednotlivé kroku algoritmu. Jedná se například o počet sekvencí, které obsahují jeden produkt a které je tedy nutné prověřit při prodlužování sekvence tímto produktem. Podobný vliv má délka sekvencí.

Závěr

Výsledky v provedených testech ukazují, že hledání častých sekvencí má potenciál pro využití v doporučovacích systémech. Podle očekávání došlo k tomu, že data, která měla více sekvenční charakter, umožnila tomuto přístupu dosáhnout lepších výsledků.

Naopak pro data, v kterých se delší sekvence příliš nevyskytují, byl efekt omezený. Nicméně i v tomto případě došlo k určitému zlepšení při umožnění delších častých sekvencí. To naznačuje, že tento přístup by mohl být přínosný při zařazení do většího hybridního systému.

Provedené pokusy ukazují, že implementovaný framework je schopen vyhodnocovat doporučovací algoritmy. Díky možnosti spustit testy na výkonném serveru došlo i k otestování značné škálovatelnosti implementovaného řešení. Té bylo dosaženo především díky paralelizaci hledání sekvencí i doporučování.

Tuto práci by bylo možné rozšířit několika způsoby, které považuji za zajímavé. Testování by mohlo být robustnější, pokud by se při načítání z databáze načítala vždy celé sekvence uživatele a množství dat se regulovalo na úrovni počtu uživatelských sekvencí místo počtu načtených transakcí.

Další zajímavou oblastí je využití času mezi nákupem jednotlivých produktů v sekvenci. Algoritmus hledání lze jednoduše rozšířit o zpracování časových informací během hledání podpory pro častou sekvenci. Statistické zpracování těchto dat a využití při doporučování by podle mého názoru mohlo vést k znatelnému zlepšení výsledků.

Literatura

[1] Aggarwal, C.: *Recommender systems: the textbook*. Springer Science + Business Media, první vydání, 2016, ISBN 9783319296579.

[2] Vala, M.: *E-learning – doporučovací systémy [online]*. 2012, [cit. 2017-03-05]. Dostupné z: `http://is.muni.cz/th/359917/fi_b/bp_final_vala.pdf`

[3] Netflix Prize homepage. `http://www.netflixprize.com/index.html`, [cit. 2017-04-09].

[4] Töscher, A.; Jahrer, M.; Bell, R. M.: The bigchaos solution to the netflix grand prize. *Netflix prize documentation*, 2009: s. 1–52.

[5] Agrawal, R.; Srikant, R.: Mining sequential patterns. In *Proceedings of the Eleventh International Conference on Data Engineering*, Mar 1995, s. 3–14, doi:10.1109/ICDE.1995.380415.

[6] Choi, K.; Yoo, D.; Kim, G.; aj.: A Hybrid Online-product Recommendation System: Combining Implicit Rating-based Collaborative Filtering and Sequential Pattern Analysis. *Electron. Commer. Rec. Appl.*, ročník 11, č. 4, Červenec 2012: s. 309–317, ISSN 1567-4223, doi: 10.1016/j.elerap.2012.02.004. Dostupné z: `http://dx.doi.org/10.1016/j.elerap.2012.02.004`

[7] Mooney, C. H.; Roddick, J. F.: Sequential pattern mining–approaches and algorithms. *ACM Computing Surveys (CSUR)*, ročník 45, č. 2, 2013: str. 19.

[8] Srikant, R.; Agrawal, R.: Mining sequential patterns: Generalizations and performance improvements. In *International Conference on Extending Database Technology*, Springer, 1996, s. 1–17.

[9] Ayres, J.; Flannick, J.; Gehrke, J.; aj.: Sequential pattern mining using a bitmap representation. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 2002, s. 429–435.

[10] Zaki, M. J.: SPADE: An efficient algorithm for mining frequent sequences. *Machine learning*, ročník 42, č. 1-2, 2001: s. 31–60.

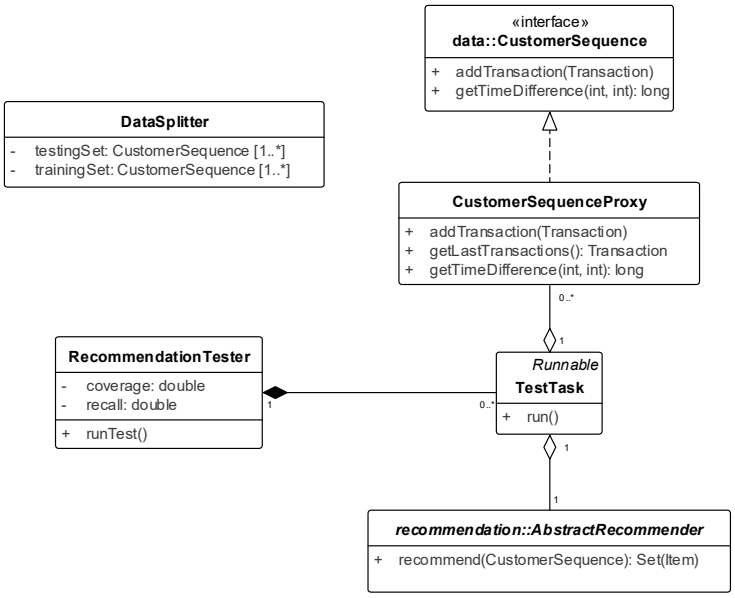
[11] Han, J.; Pei, J.; Mortazavi-Asl, B.; aj.: FreeSpan: frequent pattern-projected sequential pattern mining. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 2000, s. 355–359.

[12] Pei, J.; Han, J.; Mortazavi-Asl, B.; aj.: Mining sequential patterns by pattern-growth: The prefixspan approach. *IEEE Transactions on knowledge and data engineering*, ročník 16, č. 11, 2004: s. 1424–1440.

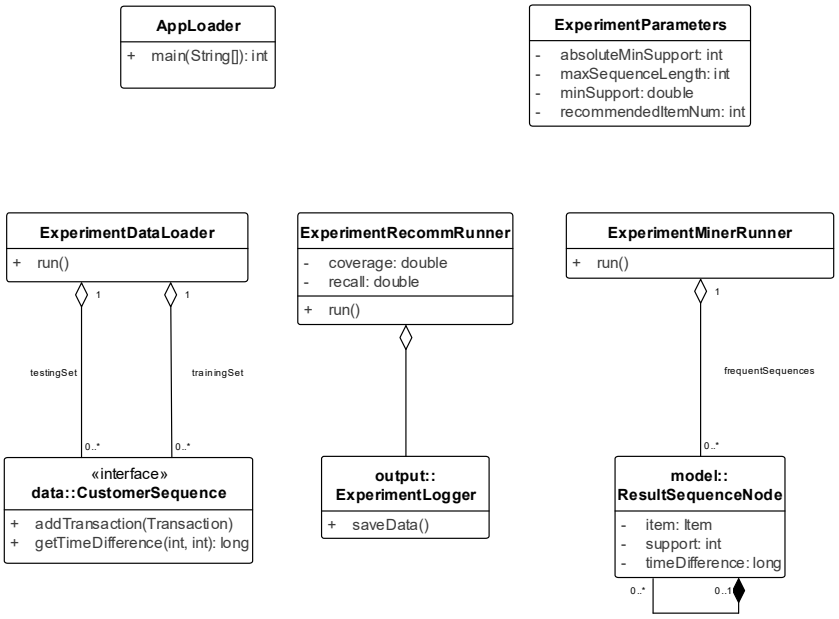
[13] Pandey, Y.; aj.: Performance Evaluation on State of the Art Sequential Pattern Mining Algorithms. *International Journal of Computer Applications*, ročník 65, č. 14, 2013.

Příloha **A**

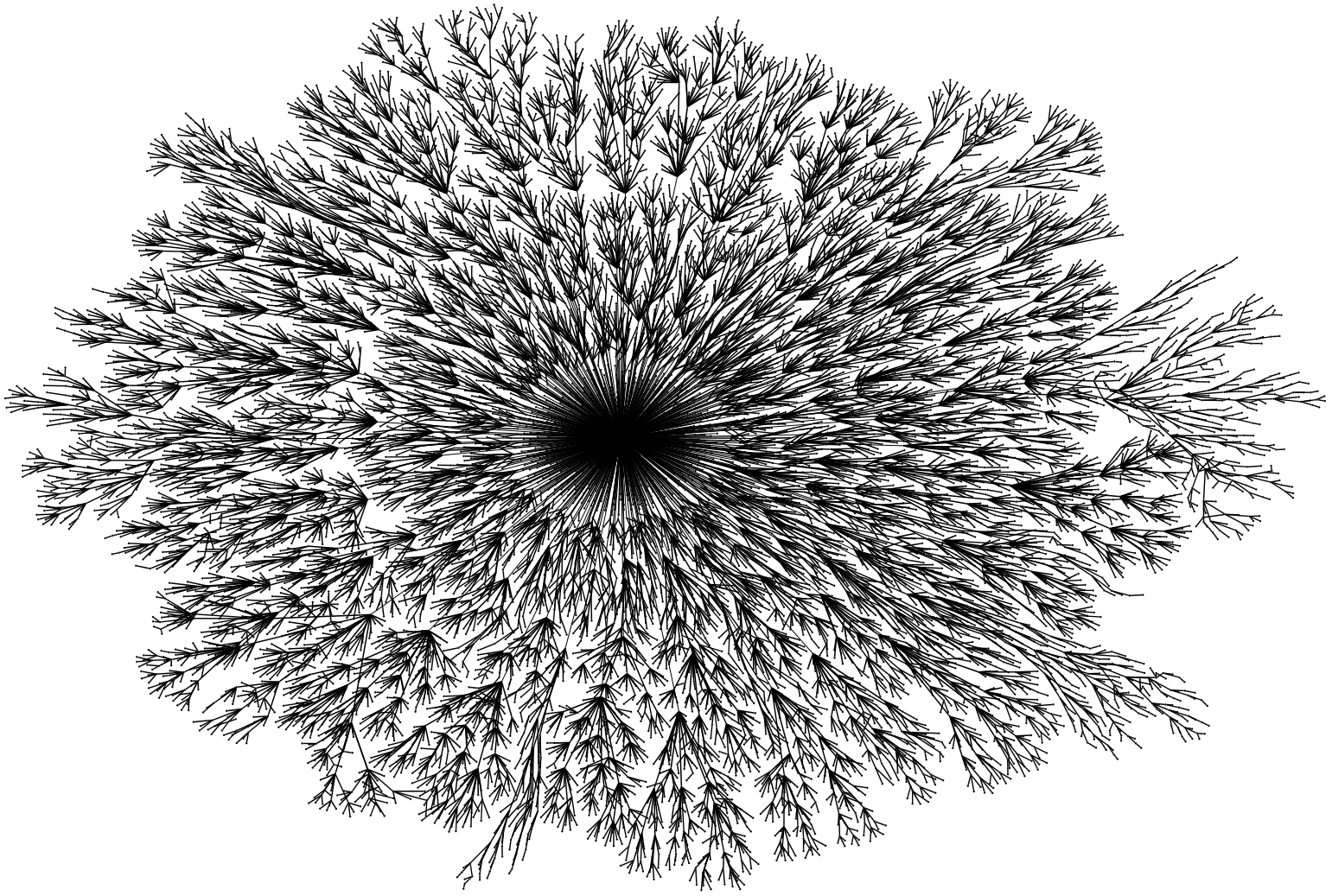
Diagramy



Obrázek A.1: Třídy v balíčku cross-validation



Obrázek A.2: Pomocné třídy v balíčku init



Obrázek A.3: Ukázka menšího stromu častých sekvencí (obsahuje 14 128 uzlů)

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `sources` ..... zdrojové kódy
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
    - `BP_Bajer_Michal_2017.tex`
    - `images`
    - `documents`
- `BP_Bajer_Michal_2017.pdf` ..... text práce ve formátu PDF
- `data` ..... data nasbíraná během testování