Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Experimentální zpracování velkých dat

Plzeň, 2016 Michal Kasal

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 8. května 2016

Michal Kasal

Poděkování

Na tomto místě bych rád poděkoval vedoucímu práce Ing. Romanu Moučkovi, Ph. D. za jeho odborné rady.

Abstract

This master thesis focuses on Big Data analytics in biology. After introducing this term, it provides an overview of important biological databases and describes a representation of stored data and approaches that can be used to obtain them. The thesis also contains the brief description of used methods for analysis of Big Data in comparative genomics. The thesis introduces current paradigms of processing Big Data, describes tools used for analytics and compares them. Some of these tools are then used for the classification variants of human genomes by ethnicity.

The classification was done using KNIME as a standalone tool and in combination with Apache Spark. The experiments showed that distributed processing of big data in framework Apache Spark was optimal. Results can be applied on existing projects. This thesis also proposes a solution for analysis of the big data stored in the EEG/ERP Portal.

Keywords

Big Data, biological databases, KNIME, Apache Spark

Abstrakt

Diplomová práce se zabývá problematikou velkých dat v biologii. Představuje termín velkých dat a nabízí přehled významných biologických databází. U těchto databází popisuje reprezentaci uložených dat a přístupy, kterými lze data získat. Dále práce obsahuje informace o některých metodách, které lze využít k analýze velkých dat v oblasti komparativní genomiky. Práce představuje současná paradigmata zpracování velkých dat a popisuje jednotlivé nástroje, které lze ke zpracování dat využívat. Tyto nástroje jsou v práci srovnány a některé z nich následně používány pro provádění experimentů týkajících se klasifikace variant lidského genomu podle etnicity.

Klasifikace byla provedena nástrojem KNIME samostatně a také v kombinaci s Apache Sparkem. Provedené experimenty ukázaly, že optimálního zpracování velkých biologických dat lze dosáhnout distribuovaným zpracováním ve frameworku Apache Spark. Výsledky práce lze aplikovat na existující projekty. Bylo navrženo řešení pro analýzu velkých elektrofyziologických dat uložených v EEG/ERP Portálu.

Klíčová slova

velká data, biologické databáze, KNIME, Apache Spark,

Obsah

1 Úvod

Nástup moderních technologií, webu, mobilních zařízení, internetu věcí, vývoj nových senzorů, a v oblasti vědy také snižující se cena prováděných měření pomocí nových technologií, vede k nárůstu dat, která tyto nové prostředky generují. Tento růst je mnohdy exponenciální. Rostoucí velikost a rozdílnost zdrojů, odkud data přicházejí, vede k potřebě implementovat nové postupy a používat nové nástroje pro jejich uchovávání, předzpracování a analýzu. Také se ukazuje, že díky velkému množství dat, je možné hledat v datech skryté vzory, přistupovat k analýze dat jiným než tradičním způsobem a klást si tak nové otázky, které mohou vést k objevení nových informací a znalostí, které se v datech skrývají.

Tato rozměrná a komplexní data lze označit termínem velká data. Jeho vymezení však nespočívá jen v posouzení dat z hlediska objemu, ale zahrnuje více aspektů, které se dat týkají, a není tak jednoduché tato data definovat. Tomuto problému se věnuje první kapitola této diplomové práce, jejímž cílem je seznámení s tímto termínem a právě různými aspekty, které se týkají dat, hovoříme-li o nich jako o velkých datech. Zkoumání velkých dat v biologii se věnuje také bioinformatika. Tato věda je v této kapitole také popsána.

V další kapitole se práce věnuje biologickým datům jako takovým. Cílem této kapitoly je poskytnout přehled o databázích, které skladují velká biologická data. Zároveň je zde zmíněno, jak jsou data v těchto databázích uchovávána, v jakém se vyskytují formátu a zda je v této biologické oblasti uplatňována nějaká standardizace. Dále se tato kapitola věnuje tomu, jak lze data z těchto veřejně přístupných databází získat.

Čtvrtá kapitola této diplomové práce se věnuje některým metodám, které se ve zvolené doméně dají používat. Cílem této kapitoly je přiblížit proces objevování znalostí v datech a metody dolování dat. V této kapitole práce také popisuje, jakým způsobem lze provádět evaluaci klasifikačních metod a tím ověřit, zda byly použity vhodné klasifikační metody a zda byly správně nastaveny jejich parametry.

V další části diplomové práce se věnuji paradigmatům zpracování velkých dat. Cílem této části je ukázat základní koncepty zpracování dat centralizovaným způsobem a distribuovaným způsobem. Ukázat základní postupy, které se v tomto oboru používají a popsat výhody a nevýhody jednotlivých metod. Na tuto část navazuje přehled nástrojů, které jsou rozděleny s ohledem na paradigmata popsaná v předcházející kapitole. U těchto nástrojů je popsaná jejich funkčnost a je uvedeno jejich srovnání a výběr kombinace nástrojů pro následné experimenty.

Cílem této diplomové práce je také provedení experimentů s vybranými biologickými daty a kritické zhodnocení výsledků. Práce obsahuje tři experimenty, z nichž dva jsou vlastní a jeden převzatý. Na těchto experimentech je ukázáno, jak lze použít nástroje různých generací a různé analytické metody při řešení konkrétního problému na konkrétní množině dat.

Díky výsledkům těchto experimentů lze získat lepší náhled na technologie a techniky zmíněné v předešlých kapitolách. A díky tomuto náhledu pak lze vytvořit některá doporučení pro projekty, které chtějí analýzu velkých dat implementovat. Jedním z takových projektů je EEG/ERP Portál. Doporučení, jak řešit problém uchovávání a analýzy dat, která jsou v Portálu ukládána, je zmíněno v poslední části této práce.

2 Koncept velkých dat

Definice velkých dat není jednoznačná [1]. Většina autorů se však kloní k názoru, že data, aby byla vnímána termínem velká data (big data), musí splňovat několik vlastností nebo alespoň podstatnou část z nich. Podle obecně uznávané definice jsou termínem velká data označována data s velkým objemem, s velkou rychlostí generování a velikou různorodostí. Vedle těchto vlastností se někdy uvádí v souvislosti s velkými daty ještě několik dalších vlastností. Jedná se o věrohodnost dat, variabilitu dat, komplexnost a viskozitu [2].

Někdy je kromě této definice používána definice velkých dat z pohledu zpracování dat, kdy jsou za velká data považována taková data, která nelze ukládat, spravovat a analyzovat běžnými softwarovými postupy a prostředky v rozumném čase [3]. Jednotlivé vlastnosti velkých dat jsou dále rozebírány níže.

2.1 Vlastnosti velkých dat

2.1.1 Objem (Volume)

Jedná se o velký objem dat. Velká data jsou uvažována v řádech terabytů až petabytů. Podle jedné studie považovalo 1144 respondentů za velká data taková data, která mají velikost alespoň 1 terabyte. Velikost dat však záleží z vysoké míry také na samotném charakteru dat. Data charakteru klíč-hodnota mají menší velikost než například data snímků. Často se také liší velikostí datasetů [1].

2.1.2 Rychlost generování a zpracování dat (Velocity)

U velkých dat je typický jejich velký růst. S časem rostou velká data exponenciálně. Je to především kvůli rozšíření mobilních zařízení, sociálních sítí, dostupností senzorů a nárůstem využívání informačních technologií obecně (například nakupováním přes internetové obchody, používání zákaznických karet), v biologii je pak nárůst dat často spojený se snižováním finančních nákladů spojených se získáváním dat (například pokles ceny sekvencování DNA).

2.1.3 Různorodost (Variety)

Různorodost dat souvisí s povahou dat a s množinou zaznamenávaných atributů dat (a metadat). Data jsou nesourodná, mají odlišný obsah. Taková data je obvykle obtížné zaznamenávat ve strukturované podobě (např. v tabulkách relačních databází). Velká data bývají nestrukturovaná a často vyžadují odlišné metody pro zpracování a analýzu oproti strukturovaným datům. Někdy je různorodost dat chápána také v souvislosti se zdroji, odkud data pocházejí (např. data generovaná z různých senzorů). Uvádí se, že jen asi 5 % veškerých dat je strukturovaných a jejich růst je mnohem menší než růst nestrukturovaných dat [4].

2.1.4 Věrohodnost dat (Veracity)

Věrohodnost dat je posuzována z hlediska jejich úplnosti, jednoznačnosti a konzistence, čili věrohodnost je ovlivněna kvalitou dat. Dalším aspektem definujícím věrohodnost dat je míra srozumitelnosti dat. Věrohodnost je dána povahou dat a tím, z jakých zdrojů data pocházejí, případně tím, zda jejich hodnoty nějakým způsobem závisejí na lidském úsudku. Metody zpracovávání velkých dat musejí počítat s proměnlivou mírou věrohodnosti dat [1].

2.1.5 Další vlastnosti velkých dat

U velkých dat se někdy uvádějí také další vlastnosti, které obvykle nebývají zahrnovány do obecné definice velkých dat.

- **Variabilita dat (Variability)**

  Někdy je na data pohlíženo z hlediska datového toku. Variabilitou dat je pak myšlena proměnlivost přítoku nových dat. Ačkoliv se principiálně jedná o změny druhé výše zmíněné vlastnosti velkých dat, někdy je touto vlastností myšlena změna jakékoliv z uvedených vlastností velkých dat [5].

- **Hodnota dat (Value)**

  Aby mělo uchovávání dat a jejich zpracování smysl, musejí mít nějakou hodnotu. Hodnotou dat je v tomto kontextu míněn potenciální přínos nalezených informací v datech. U velkých dat platí, že jejich hodnota narůstá s objemem dat.

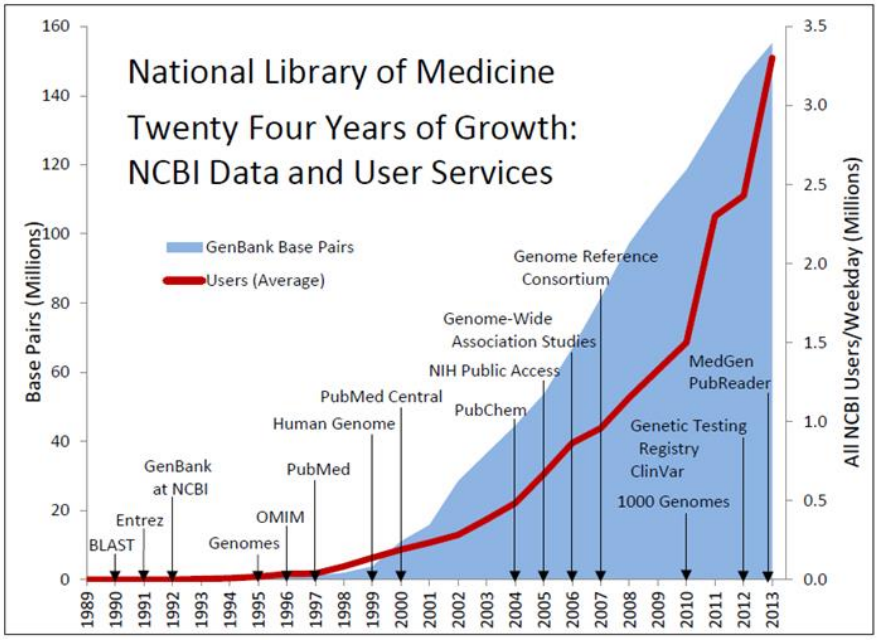
- **Viskozita dat (Viscosity)**

  Viskozita dat je určena na základě času od zachycení dat do doby jejich zpracování [2]. Viskozita dat se liší v závislosti na doméně dat [4]. V případě analýzy velkých dat je někdy nutné počítat s tím, že data přicházejí ve vlnách.

2.2 Bioinformatika

Termín bioinformatika byl dříve používán k popisu vědy zabývající se studiem informačních procesů v biologických systémech. Dnes jím označujeme spíše vědu, která se zabývá aplikací počítačových technologií a metod na ukládání a extrakci biologických dat, jejich zpracování a analýzu a na interpretování a vizualizaci výsledků analýz [6]. Předmětem zájmu bioinformatiky jsou především data týkající se genetických sekvencí, struktur proteinů a jejich vzájemných interakcí.

S klesající cenou sekvencování genů a dostupností technologií narůstají exponenciálně data, která jsou dostupná z veřejných databází. Nárůst dat je typický pro všechny domény, ale v této doméně je nejmarkantnější. Obrázek níže ukazuje nárůst dat uložených v databázi GenBank, počet uživatelů a jednotlivé projekty uveřejněné Národním centrem pro biotechnologické informace v USA [7].



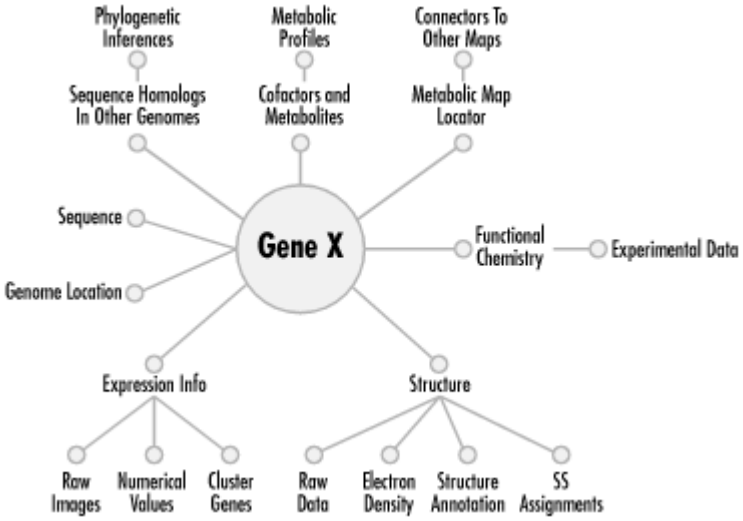
Obrázek 1: Graf velikosti databáze GenBank a počtu uživatelů od roku 1989 do roku 2013 s označením významných projektů NCBI [7]

3 Biologické databáze

Velká data v biologii lze rozdělit do několika skupin s odlišnými charakteristikami, které závisejí především na doméně dat. V této kapitole jsou uvedeny některé významné skupiny biologických dat a jejich úložišť, jejich charakteristiky, nástroje a metody používané k jejich analýze a zpracování a možnosti přístupu k vybraným úložištím.

3.1 Databáze genů, genomů a proteinů

Tato doména bioinformatiky obsahuje největší počet biologických databází. Tyto databáze se dají dále rozdělit v závislosti na tom, která data o genech obsahují, na databáze sekvencí DNA a RNA, databáze proteinů, databáze struktur proteinů a databáze interakcí genů a fyzikálních vlastností. Některé informace, které souvisejí s konkrétním genem, zobrazuje obrázek níže.



Obrázek 2: Část informací spojených s genem [8]

3.1.1 Databáze sekvencí DNA, RNA a proteinů

Tato kategorie biologických databází obsahuje uložené informace o sekvencích DNA, RNA a genomu. Tyto sekvence jsou definovány jako posloupnosti po sobě jdoucích bází, kde jedna báze je reprezentována jedním znakem z abecedy G, A, T, C v případě DNA nebo G, A, U, C u RNA. Posloupnost je dále tvořena dalšími znaky, které nahrazují nejednoznačně určené báze, například N reprezentuje možnost výskytu všech čtyř bází na daném místě sekvence [9].

Jednotlivé sekvence jsou uloženy spolu s metadaty v prostých souborech. V této doméně existuje několik standardních výměnných formátů, ve kterých jsou soubory ukládány [10, 11]. Často používaným formáty jsou GenBank Format, FASTA Format, EMBL Format, GCG Format, IG Format a NBRF Format. Všechny formáty jsou uloženy jako prostý text. Tyto formáty se liší svojí strukturou a reprezentací metadat.

Ve formátu FASTA jsou metadata uložena v jednom řádku. Záznam začíná znakem „>“ a následuje kód, jméno a původ sekvence, případně další metadata obsahující například délku sekvence. Na dalších řádkách je sekvence, která je ukončena znakem „*“ [11, 12]. Formáty IG a NBRF jsou pak obdobné s několika rozdíly.

NBRF začíná shodným znakem jako FASTA, obsahuje znak identifikující, zda je sekvence kompletní a číslici označující její typ. Středníkem je oddělen identifikátor sekvence a na další řádce jsou popsána další metadata sekvence jako je celý název sekvence a pomlčkou oddělený název druhu organizmu, ke kterému se sekvence vztahuje.

IG Formát obsahuje metadata na prvních dvou řádkách obdobně jako NBRF formát. Komentář je označen středníkem. První řádka obsahuje zdrojový organismus a popis sekvence, na druhé řádce je identifikátor sekvence. Mezi tyto *úsporné* formáty lze zařadit formát GCG, který obsahuje metadata na několika řádcích ukončené dvěma tečkami. GCG obsahuje v metadatech informaci o celkovém počtu bází [10].

EMBL a GenBank Format obsahují oproti předchozím formátům více metadat. Tato metadata jsou definována značkami. V případě EMBL jde o dvouznakové zkratky, oddělovačem klíče od hodnoty je tabulátor. Data sekvence začínají na další řádce po značce SQ. V GenBank formátu je použitý podobný princip, jen místo dvouznakových značek se použijí celá slova s definovaným významem. Zmíněné nejpoužívanější formáty jsou uvedeny v následující tabulce.

FASTA Format	NBRF Format	IG Format
>AB000263 |acc=AB000263|descr=Homo sapiens mRNA, complete cds.|len=368 ACAAGATGCCATTG…	>P1;ILEC lexA repressor – Escherichia coli MKALTARQQEVFDLIRD**…**	; comment ; comment AB000263 ACAAGATGCCA…
GCG Format	**EMBL Format**	**GenBank Format**
BASE COUNT 215 A 224 C 263 G 250 T ORIGIN Filename, Length of sequence, Date, Check-sum, .. 1 GAATTCGATA AATCTCTGGT…	ID id code for sequence in DB AC access number DT dates of entry and modification KW key cross-reference or source OS source organismu RN literature reference (RP, RX, RA, RT) SQ count of A, C, G, T, … gaattcgata aatctctggt …	LOCUS name of locus, type of sequence DEFINITION description of entry ACCESSION accession numbers SOURCE source organism REFERENCE COMMENT FEATURES information about base pos. BASE COUNT count of A, C, G, T, … ORIGIN text indicating start of seq. 1 gaattcgata aatctctggt …

Tabulka 1: Přehled vybraných formátů sekvencí DNA [10, 11]

Od FASTA formátu byl odvozen FASTQ formát, který mimo metadat zahrnutých ve FASTA formátu, obsahuje ještě informace o kvalitě bází. Skóre kvalit bází je stejně dlouhé jako sekvence a určuje, s jakou pravděpodobností je báze správně určena [12].

Někdy jsou data z této domény ukládána také do relačních databází. Používaná databázová schémata často neobsahují relace mezi tabulkami. Příkladem využívání relačních databází jsou projekty Ensembl, UCSC a GO [13].

Volba metody přístupu k datům v uložených databázích závisí na tom, jaká máme vstupní data a jaký experiment chceme provádět. K většině databází je možné připojit se a přenést data uložená v prostých souborech prostřednictvím protokolu FTP. Některé databanky umožňují stahovat také dumpy relačních databází, do kterých ukládají data.

Nad některými databázemi z této domény je vystavěn indexovací a dotazovací systém SRS (Sequence Retrieval System). [14] Jedná se o homogenní rozhraní [15], které umožňuje dotazování a získávání dat nad více než 141 biologickými databázemi z této domény [16]. SRS má objektově orientovaný design. Pomocí metadat jsou nadefinovány třídy pro jednotlivé záznamy a také pravidla pro parsování textu prostých souborů a určení vlastností záznamu. SRS umožňuje indexovaná obousměrná propojení různých úložišť (například pomocí vlastnosti data reference nebo pomocí identifikátoru záznamu). Tímto způsobem je možné propojovat množiny záznamů a kombinovat je s logickými operátory AND, OR a NOT (někdy BUTNOT), a tím vyhledávat nad různými heterogenními databázemi z této domény [17]. Příkladem tohoto systému je Entrez, který umožňuje vyhledávání v několika desítkách databází prostřednictvím webového rozhraní [18].

Projekt Ensembl, což je významný zdroj dat anotovaných sekvencí DNA, RNA a proteinů z více databází, umožňuje přístup k datům také prostřednictvím REST API implementovaného v Perlu. Prostřednictvím API lze získávat data jednotlivých sekvencí, metadata, ale také provádět operace s těmito daty a metadaty. Výstupní odpovědi jsou poskytovány zpravidla ve formátu JSON, JSONP a XML [19].

Databáze sekvencí proteinů a souvisejících dat UniProt obsahuje data popsaná pomocí RDF a poskytuje rozhraní pro zadávání dotazů v jazyce SPARQL. Data v UniProtu jsou také klasifikována do hierarchické stromové struktury organizmů. Pomocí dotazů jazyka SPARQL lze provádět dotazy a získávat tak informace o taxonomii organizmů a také data sekvencí proteinů [20].

```
PREFIX up:<http://purl.uniprot.org/core/>
PREFIX taxon:<http://purl.uniprot.org/taxonomy/>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
SELECT ?protein ?aa
WHERE
{
    ?protein a up:Protein .
    ?protein up:organism ?organism .
    {
        ?protein up:organism taxon:83333 .
    } UNION {
        ?protein up:organism ?organism .
        ?organism rdfs:subClassOf+ taxon:83333 .
    }
    ?protein up:sequence ?s .
    ?s rdf:value ?aa
}
```

Kód 1: Příklad dotazu vracejícího sekvence aminokyselin bakterie e-coli K12 v jazyce SPARQL nad databází UniProt [20]

Často používaným způsobem získávání dat z databází sekvencí DNA a RNA je vyhledávání na základě podobnosti sekvencí. Jednou z možností vyhledávání podobnosti sekvencí je algoritmus FASTA.

FASTA (fast alignment) je heuristický algoritmus sloužící k vyhledávání sekvencí na základě jejich podobnosti pomocí lokálního zarovnávání využívající dynamického programování. Algoritmus FASTA se skládá z několika kroků. Nejprve se najdou společná slova mezi zadanou sekvencí a sekvencemi z databáze, což jsou podmnožiny posloupností o velikosti *k*. Vytvoří se matice, ve které se označí společné báze a ohodnotí se diagonály. Vyberou se diagonály s nejvyšším hodnocením a přehodnotí se podle oceňovací matice, naleznou se podobné oblasti, kratší než *k*. Konce oblastí, které se neshodují, jsou vypuštěny. Diagonály, s hodnocením nad určitou mezí, jsou spojovány s tím, že mezery penalizují výsledné hodnocení kombinace diagonál. V posledním kroku se pomocí dynamického programování nalezne kombinace s nejvyšším hodnocením [21].

Dalším možným algoritmem, který lze použít k vyhledání podobných sekvencí, je rychlejší algoritmus BLAST (Basic Local Alignment Search Tool). V prvním kroku algoritmu se postupně pro každou pozici v zadané sekvenci vytvoří seznam slov zadané délky dané sekvence, jejichž skóre je větší než zadaná hodnota. V dalším kroku se prohledá databáze a naleznou se sekvence, které obsahují slovo z přechozího vytvořeného seznamu. Části sekvencí v konkrétní sekvenci se obousměrně prodlouží, dokud jejich ohodnocení roste. Všechny sekvence, které mají ohodnocení vyšší než zadanou mez, jsou nalezené jako podobné [22].

3.1.2 Databáze genomů

Genom organizmu je jeho kompletní sada molekul DNA, která zahrnuje všechny jeho geny [23]. Databáze genomů obsahují zpravidla genomy jednoho druhu organizmu (například lidské, myší). Databáze obsahují soubory obsahující zarovnané kusy sekvencí (SAM anebo BAM formát), které se dál zpracovávají do výsledných souborů, které obsahují varianty genomů. Tyto výsledné soubory jsou uložené ve formátu VCF, který je chápán jako standard pro ukládání variant genomů.

VCF soubory jsou textové soubory, které obsahují varianty v porovnání s referenčním genomem. Začátek souboru je tvořen komentáři s prefixem ##. V nich je uveden formát souboru (tj. verze použitého VCF formátu), datum, referenční sekvence, zdroj dat apod. Verze formátu je povinná. Část metadat je uložena ve strukturované podobě. Pro označení strukturovaných metadat se použijí značky ##INFO, ##FORMAT, ##FILTER a další.

Níže je uveden příklad zápisu těchto strukturovaných metadat [24].

```
##INFO=<ID=ID,Number=number,Type=type,Description="description",Source="description",Version="128">
```

Kód 2: Zápis strukturovaných metadat ve formátu VCF [24]

Po sekci komentářů následuje hlavička, která je povinná a obsahuje 8 povinných sloupců:

- CHROM – obsahující číslo chromozomu
- POS – referenční pozice báze, první báze je na pozici jedna a pozice jsou uspořádány vzestupně
- ID – unikátní identifikátor varianty
- REF – referenční báze
- ALT – alternativní báze, pokud je uvedeno více alternativních bází, jsou odděleny čárkou
- QUAL – skóre kvality, které určuje, s jakou pravděpodobností byla alternativní báze zvolena správně
- FILTER – status filtrování
- INFO – další informace, uložené ve formátu <klíč>=<hodnota>

Následují sloupce, které obsahují identifikátor subjektu.

```
CHROM POS ID REF ALT QUAL FILTER INFO FORMAT NA00001
20 14370 rs6054257 G A 29 PASS NS=3;DP=14;AF=0.5;DB;H2 GT:GQ:DP:HQ 0|0:48:1:51
```

Kód 3: Ukázka hlavičky a záznamu jedné varianty jednoho zkoumaného genomu [24]

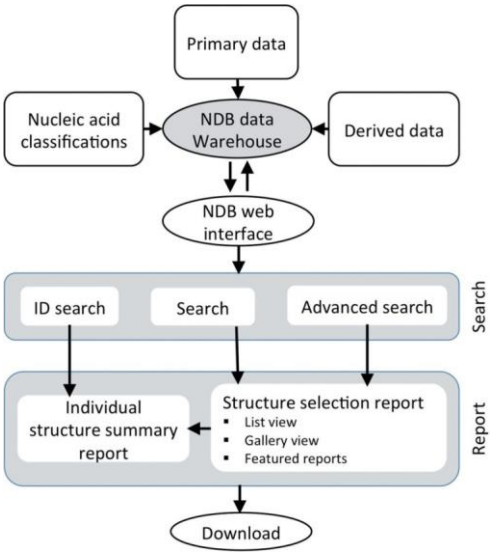
Data variant genomů a dodatečné soubory obsahující zarovnání sekvencí jsou nabízené zpravidla jako komprimované soubory a lze je stáhnout prostřednictvím FTP. Některé z databází umožňují i jednoduché vyhledávání podle konkrétního genu nebo podle pozice v genomu [25, 26].

Genové ontologie

Genové ontologie slouží k popisu atributů genů a jejich produktů a souvisejících procesů. Často používanou ontologií je Gene Ontology (GO). GO je genová ontologie vyvíjená GO konsorciem. GO popisuje oblasti buněčných komponent, molekulárních funkcí a biologických procesů. Každý genový produkt má alespoň jednu molekulární funkci a je použitý alespoň v jednom biologickém procesu v alespoň jedné buněčné komponentě. Strukturou této ontologie je acyklický orientovaný graf. Vrcholy grafu tvoří axiomy ontologie. Hrany představují relace mezi axiomy. Díky tomu lze vytvořit hierarchii pojmů, z nichž rodič vrcholu obsahuje vždy obecnější axiom než jeho potomek, čili pokud je nějaký genový produkt popisován vrcholem, pak je popisován i jeho rodičem [27]. GO je využívaná při vyhledávání v biologických databázích například podle názvu genu, kdy umožňuje vyhledávání pomocí synonym názvu [28].

3.1.3 Strukturální databáze nukleových kyselin a proteinů

Strukturální databáze proteinů uchovávají informace o 3D struktuře makromolekul proteinů. Data jsou uložena v relačních databázích, ze kterých se při aktualizaci databází generují textové soubory ve formátu PDB a mmCIF [29, 30]. K datům struktur nukleových kyselin lze přistupovat prostřednictvím webového rozhraní. Schématická reprezentace toku dat mezi databází a webovým rozhraním, hledání a reporting dat je zobrazena na obrázku níže.



Obrázek 3: Reprezentace toku dat mezi databází a webovým rozhraním databáze struktur nukleových kyselin [29]

K strukturám proteinů v databázi PDB je možné přistupovat více způsoby. Jednou z možností je využití webového rozhraní. Přes webové rozhraní lze data stahovat také v dávkách. Po odeslání vyhledávacího dotazu je stažen Download Tool, který používá Java Web Start protokol a který stáhne požadovaná data [31]. K datům v této databázi lze přistupovat také přes FTP a HTTP. Další možností přístupu k datům struktur proteinů je použití RESTful webové služby, která umožňuje vyhledávání a přenos dat anotací, popisů a sekvencí proteinů. Je možné také vyhledávat sekvence na základě podobností pomocí algoritmu BLAST. Výstupem je pak sekvence ve FASTA formátu [32].

Struktury proteinů a nukleových kyselin jsou uloženy ve standardních formátech PDB, PDBML nebo mmCIF. PDB a mmCIF formát jsou uložené jako prostý text, PDBML je pak XML reprezentace PDB. PDB obsahuje značky a příslušná data oddělená mezerami. Každá další značka začíná na novém řádku. PDB je rozdělen do několika částí. Nejprve je obsažena hlavička, která obsahuje metadata. Zde je uveden název vzorku, datum, ID struktury, informace o autorech, kteří definovali strukturu, a další metadata. Následuje část komentářů a dalších specifikací označená značkami REMARK 0-999. Poté následuje část obsahující data o struktuře proteinu, jeho složení a pozice jednotlivých atomů určeného souřadnicemi *x*, *y* a *z* [33].

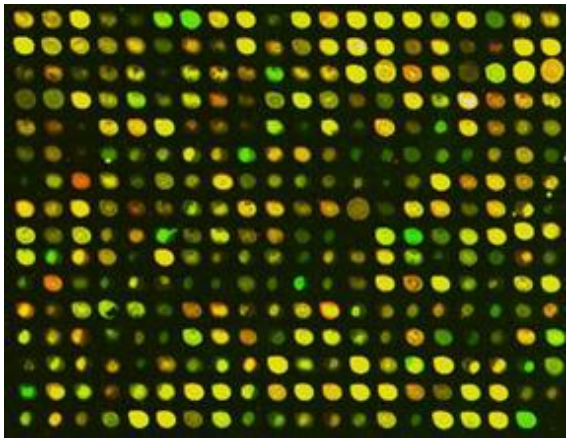
Formát mmCIF je novějším formátem používaným k popisu struktury proteinu. Data v tomto formátu jsou reprezentována jako dvojice klíč a hodnota. Jednotlivé klíče začínají podtržítkem. První značkou je _entity_poly.entity_id, obsahující identifikátor struktury. Formát má hierarchickou strukturu. Jednotlivé kategorie obsahují další podkategorie [34].

3.1.4 Databáze čipů exprese genů

Tyto databáze uchovávají data týkající se exprese genů, která je pozorovaná na čipech (microarrays). Genová exprese je složena z transkripce (převodu DNA na RNA) a translace (převodu RNA na aminokyseliny) [35].

Microarray data jsou ukládána v relačních databázích nebo jako soubory v souborovém systému. Jedná se o data oddělená tabulátory [36]. Získat data lze několika způsoby, například ArrayExpress databáze umožňuje přístup k datům prostřednictvím webového rozhraní, programově pomocí balíčku v jazyce R, přímo prostřednictvím FTP nebo s použitím REST webové služby [37].

Data ukládaná v databázích jsou nabízena v původní formě (tak jak byla vygenerována čtečkami čipů) nebo ve zpracované formě (např. normalizovaná data). Jsou přijímána jen data, která vyhovují MIAME (Minimum Information About a Microarray Experiment) [38] nebo MINSEQE (Minimum Information about a high-throughput SEQuencing Experiment) standardu. Tím je zajištěn dostatečný popis poskytovaných dat. Avšak samotná data z čtecích zařízení nemají jednotný formát, což je zapříčiněno různými výrobci čteček [39]. Často používanými formáty jsou GPR (GenePix Results) a soubory Affymetrix CEL.



Obrázek 4: Naskenovaný microarray [40]

3.2 Databáze medicínských snímků

Jedná se o databáze dat snímků rentgenu, počítačové tomografie, magnetické rezonance, mamografu, retinopatie, virtuální kolonoskopie a dalších. Tato data jsou ukládána podle standardu DICOM, který je standardním formátem pro ukládání medicínských snímků. Soubory v sobě obsahují kromě snímků také metadata popisující pacienta, vyšetření, datum a další informace. Důležitými metadaty v hlavičce souboru je typ souboru, který určuje, o který snímek se jedná (např. snímek počítačové tomografie). Tato metadata jsou neoddělitelná od snímků [41].

Z povahy dat je přístup k těmto databázím zpravidla vázaný na existenci uživatelského účtu a odsouhlasení podmínek použití dat. Po přihlášení bývá možné přistupovat k datům prostřednictvím FTP nebo přes webové rozhraní, které umožňuje prohledávání dat podle definovaných dotazů. Některé databáze snímků, například databáze zaměřená na snímky rakoviny Cancer Imaging Archive, také vystavují REST API, které lze použít ke stahování metadat ve formátech JSON, XML nebo CSV, ale také ke stahování snímků v archívech [42].

3.3 Databáze signálů

Významnou databankou zpřístupňující záznamy medicínských signálů z ostatních databází je PhysioBank. PhysioBank umožňuje prostřednictvím webového rozhraní PhysioBank ATM vyhledat a zobrazit libovolný záznam z kterékoliv zapojené databáze. Stažení většího množství dat je možné dávkově pomocí programů wget a rsync [43].

Data jsou poskytována v binární formě. PhysioBank poskytuje utility, které umožňují převod binárních souborů do textové reprezentace. Příklad textové reprezentace je uveden v tabulce níže. Každý záznam obsahuje soubor měřených dat (např. EKG signál), soubor obsahující anotace, ve kterém jsou popsány události, které během měření nastaly (například údery srdce) a poznámky, které obsahují údaje o měřené osobě (věk, pohlaví, diagnóza, medikace) anebo údaje o měření. Rozdíl mezi metadaty anotací a poznámek spočívá v tom, že anotace obsahují přechodné jevy, u kterých je zaznamenán čas vztažený k průběhu měření, zatímco poznámky obsahují charakteristiky měření jako celku [44].

Textová reprezentace	Příklad
Signál	time MLII V1 (sec) (mV) (mV) 300.000 -0.095 -0.140 300.003 -0.110 -0.140 300.006 -0.110 -0.120
Anotace	Time Sample # Type Sub Chan Num Aux 5:13.155 112736 N 0 0 0 5:13.480 112853 + 0 0 0 (B
Poznámky	a01 2 128 7680 17:42:40 a01.dat 16 170.94/mV 0 0 -14 29711 0 ECG

Tabulka 2: Textová reprezentace záznamu EKG poskytovaného databází PhysioBank [44]

EEG data jsou ukládána v různých formátech. Například se jedná o formáty EDF, EDF+ a MIT Format [43]. EDF obsahuje hlavičku (viz níže), ve které je uvedena verze použitého EDF formátu, identifikátor pacienta, identifikátor měření, datum a čas začátku měření, počet záznamů, délka trvání záznamu v sekundách, počet signálů a další metadata. EDF+ je pak rozšířenou verzí EDF formátu. Oproti EDF může například obsahovat nespojité záznamy, anotace a také záznamy o nastalých událostech [45].

```
0 MCH-0234567 F 16-SEP-1987 Haagse_Harry Startdate 16-SEP-1987 PSG-1234/1987 NN Telemetry03 16.09.8720.35.00768
Reserved field of 44 characters 2880 30 2 EEG Fpz-Cz Temp rectal AgAgCl cup electrodes Rectal thermistor uV
degC -440 34.4 510 40.2 -2048 -2048 2047 2047 HP:0.1Hz LP:75Hz N:50Hz LP:0.1Hz (first order)
15000 3 Reserved for EEG signal Reserved for Body temperature
```

Kód 4: Příklad hlavičky EDF souboru

Formát MIT obsahuje binární soubory s uloženými digitalizovanými signály (*MIT Signal files*). Tyto soubory nezahrnují jako svojí součást hlavičku, která je uložena zvlášť v textových souborech *MIT Header files* (viz příklad níže). V nich je obsažen například identifikátor signálu, počet signálů, formát v jakém jsou signály reprezentovány, název datového souboru a další metadata. Anotace jsou obsaženy v souborech *MIT Annotation files*. [43].

```
m008 4 5000 15000000
m008.dat 16 6621.4983(9589)/mV 0 0 0 23850 0 I
m008.dat 16 23704.8031(-17713)/mV 0 0 0 -16565 0 II
m008.dat 16 102404.2972(-2547)/mV 0 0 0 31690 0 RESP
m008.dat 16 401.6118(-1881)/mV 0 0 0 -432 0 SCG
#Long measurement for subject 8
```

Kód 5: Příklad hlavičky uložené v MIT Header souboru

3.4 Přehled charakteristik databází podle domén

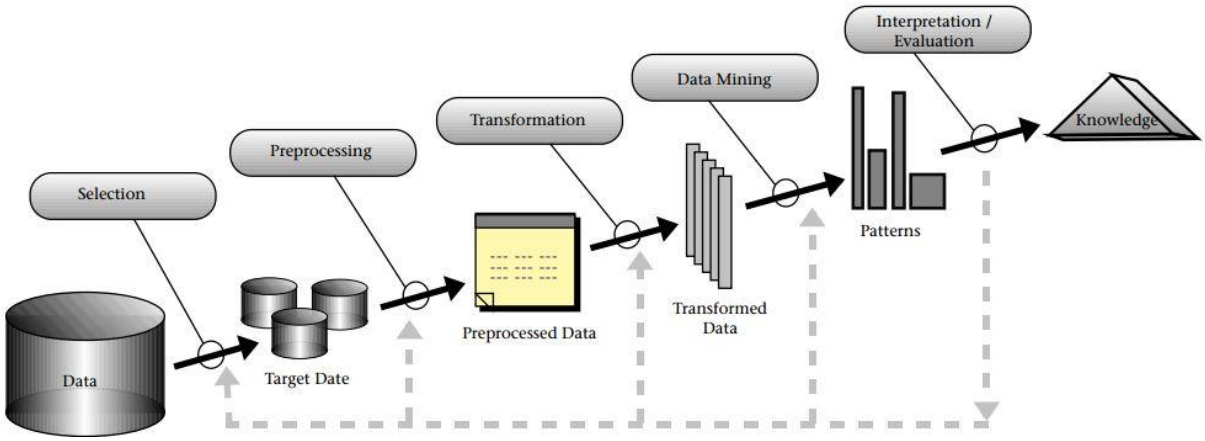
V tabulce níže jsou uvedeny základní charakteristiky databází rozdělených podle typu.

	Sekvence DNA, RNAa	**Databáze variant genomů**	**Strukturální databáze**	**Databáze medicínských snímků**	**Databáze medicínských signálů**
Charakter dat	Posloupnosti textu s omezenou abecedou	Data oddělená tabulátory	Klíč a hodnoty, pozice molekul	Binární soubory obsahující ne/komprimované snímky	Binární nebo textové reprezentace časových řad
Standardy reprezentace dat	Často se používá FASTA	VCF pro ukládání variant	PDB soubory a nástupce mmCIF	Standard DICOM	Není
Přístup přes webové rozhraní	ANO	ANO	ANO	ANO	ANO
FTP přístup	ANO	ANO	ANO	ANO	NE
API	REST API	NE	REST API	REST API	NE
Vyhledávání dat	Web, BLAST	Web	Web, BLAST, SPARQL	Web	Web
Stahování dávek	Ano	Ne	Pomocí Java Download Tool	Přes API	Ano wget, rsync

Tabulka 3: Základní charakteristiky biologických databází

4 Techniky analýzy velkých biologických dat

K analytickému zpracování velkých biologických dat se používají různé metody, které lze označit jako metody data miningu (dolování dat). Dolování dat je analytická metodologie získávání skrytých, netriviálních a potenciálně užitečných informací z dat. Jedná se o jeden dílčí krok celého procesu získávání znalostí analýzou velkých dat (někdy označovaného jako knowledge discovery nebo knowledge discovery in databases (KDD)) [46]. Jednotlivé kroky tohoto procesu jsou znázorněny na následujícím obrázku.



Obrázek 5: Proces získávání znalostí [46]

Kroky získávání znalostí z dat jsou [46]:

1. Porozumění dané doméně a pochopení již známé relevantní znalosti pro určení cíle procesu získávání znalostí.
2. Výběr datasetu na základě předchozího kroku.
3. Předzpracování dat (například doplnění chybějících hodnot, normalizace a odstranění šumu)
4. Redukce dimenzionality dat, výběr relevantních atributů.
5. Výběr metody dolování dat (např. klasifikační metody, shlukovací metody)
6. Výběr konkrétních modelů dané skupiny (například SVM, naivní Bayesův klasifikátor)
7. Dolování dat (sestavení modelů)
8. Interpretace získaných vzorů v datech, návrat k předchozím bodům pro zpřesnění procesu nebo pro alternativní postup, pokud se ukázal zvolený model chybný
9. Interpretace znalosti, dosazování objevené znalosti do kontextu jiných apod.

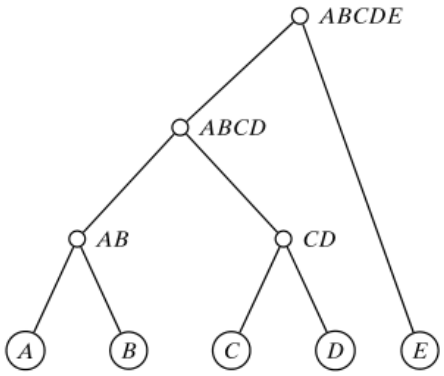
Experimenty prováděné ve vybrané doméně (analýza variant genomů) využívají obecné nástroje dolování dat. Jedná se o klasifikaci dat a shlukování. Některé metody shlukování a klasifikace dat, které jsou využitelné v této doméně, jsou uvedeny níže. Více podrobností ke zmíněným metodám, další používané metody a jejich implementace lze nalézt v odborné literatuře. Předchozí kroky procesu získávání znalostí jsou uvedené u příslušných experimentů.

4.1 Shluková analýza

Cílem shlukové analýzy je nalézt v datech nějakou strukturu nebo organizaci dat. V celé množině instancí objektů se hledají podmnožiny tak, aby platilo, že objekty patřící do stejné množiny jsou si co nejvíce podobné a co nejvíce rozdílné oproti objektům jiných skupin. Tyto podmnožiny (třídy) nejsou předem známé [47].

4.1.1 Hierarchické shlukování

Hierarchické shlukování hledá v datech hierarchickou stromovou strukturu. Tuto stromovou strukturu můžeme graficky znázornit dendrogramem (viz obr. 6). Každý shluk může obsahovat podshluky nižšího řádu a on sám může být součástí shluku vyššího řádu. Při konstruování hierarchického shlukování hledáme posloupnost rozkladů $S^{(k)}, k = 1, 2, \ldots, n$, kde n je počet instancí objektů od jediného shluku $S^{(1)}$až po $S^{(n)}$, kde $S^{(n)}$představuje shluky tvořené jednotlivými objekty [48]. Hierarchické shlukování lze využít například u hledání nových superpopulací při analýze variant genomu.



Obrázek 6: Dendrogram [49]

Divizivní hierarchické shlukování

Divizivní hierarchické shlukování postupuje od shora dolů. Postupně rozděluje množinu do shluků, dokud není počet shluků roven počtu objektů. Postupuje tedy obráceně než aglomerativní hierarchické shlukování. Divizivní shlukování je často používáno v biologii pro sestavování tzv. fylogenetických stromů.

Aglomerativní hierarchické shlukování

Opačným přístupem je aglomerativní shlukování, které postupuje od spodu nahoru. Začíná tím, že jednotlivé objekty reprezentují samostatné shluky. Určí se podobnost mezi shluky pomocí výpočtu distanční matice, najdou se shluky, které jsou nejbližší (tj. jejich podobnost je maximální), ty se spojí v jeden, a tím se vytvoří vyšší úroveň shluků. Algoritmus skončí, když se spojily všechny shluky do jediného. Aglomerativní shlukování není příliš vhodné pro velké datasety [48, 49]. K určování podobnosti lze použít několik rozdílných způsobů a několik rozdílných metrik určování vzdálenosti δ [48].

Metoda nejbližšího souseda

Vzdálenost mezi shluky se určí jako minimální vzdálenost bodu prvního shluku a bodu druhého shluku. Formálně:

$$\delta(C_i, C_j) = \min\{\delta(x, y) \mid x \in C_i, y \in C_j\} \quad (1)$$

Nevýhodou je, že se i vzdálené shluky mohou spojit v jeden, pokud se mezi nimi vytvoří zřetězení jiných shluků.

Metoda nejvzdálenějšího souseda

Vzdálenost mezi shluky se určí jako maximální vzdálenost jejich prvků od sebe. Formálně:

$$\delta(C_i, C_j) = \max\{\delta(x, y) \mid x \in C_i, y \in C_j\} \quad (2)$$

Nevýhodou této metody je tvorba malých kompaktních shluků.

Metoda průměrné vazby

Vzdálenost mezi shluky se určí jako průměrná hodnota vzdáleností jejich prvků. Tato metoda vede v častých případech ke stejnému problému jako předcházející metoda.

Centroidová metoda

Určí se centroid každého existujícího shluku a podobnost se určí jako vzdálenost centroidů jednotlivých skupin. Nevýhodou je spojení shluků velmi rozdílné velikosti, pak se může centroid nového shluku nacházet velmi blízko většího shluku, ze kterého vznikl.

Wardova metoda

Wardova metoda operuje se změnou sumy čtverců chyb. Obvykle tvoří shluky podobné velikosti.

4.1.2 Reprezentativní nehierarchické shlukování

Metody tohoto shlukování vycházejí z existence tzv. reprezentativního bodu, který charakterizuje konkrétní shluk. Oproti hierarchickým metodám se hodí pro shlukování tam, kde nepředpokládáme existenci stromové organizace dat [48].

Metoda k-průměrů

Metoda k-průměrů využívá centroidy jako reprezentativní body. Jednotlivé clustery jsou tvořeny minimalizací celkové chyby čtverců vzdáleností uvnitř skupin. Vstupem algoritmu je informace o počtu shluků, které chceme najít. Postup je následující [48]:

1. Zvolí se počáteční umístění centroidů (nejčastěji náhodně)
2. Pozice všech objektů se zhodnotí a objekt se přesune do shluku s nejmenší vzdáleností k jeho centroidu.
3. Přepočítají se centroidy každého shluku.
4. Opakují se kroky 2 a 3, dokud existuje přesun, který zmenšuje celkovou chybu čtverců uvnitř skupiny

Nevýhoda této metody je problém citlivosti vůči odlehlým hodnotám a také nutnost zvolit počet shluků (například metodou lokte nebo metodou vyhovění účelu). Modifikací algoritmu k-průměrů je algoritmus x-průměrů, kdy se místo stanoveného počtu shluků zadá rozmezí počtu hledaných shluků. V takovém případě je aplikován algoritmus k-průměrů s počtem shluků určených spodní hranicí rozsahu. Po dokončení algoritmu k-průměru se některé sluky rozloží na dva a určí se výskyt nových centroidů. Další modifikace algoritmu je metoda k-medoidů, kdy se využívá jako reprezentativní objekt medoid. Tato metoda je méně náchylná na přítomnost odlehlých hodnot [48].

4.1.3 Validace výsledků shlukové analýzy

Validace výsledků shlukové analýzy slouží k posouzení, zda byl nalezen optimální počet shluků. Zároveň ji lze použít pro měření kvality různých shlukovacích algoritmů, případně měřit kvalitu použitých různých proměnných u stejného algoritmu. Obvykle se validace provádí výpočtem validačních indexů, kterých existuje celá řada. Jeden z nich je zmíněný níže.

Daviesův – Bouldinův validační index

Určuje podíl součtu rozložení uvnitř shluku a mezi shluky. Nízké hodnoty tohoto indexu indikují správně zvolený počet a dobře oddělené kompaktní shluky daleko od sebe. Index můžeme určit ze vztahu:

$$DB = \frac{1}{n}\sum_{i=1}^{n} max\left\{\frac{S_n(Q_i) + S_n(Q_j)}{S_n(Q_i, Q_j)}\right\} \tag{3}$$

kde $i \neq j$, n je počet shluků, $S_n(Q_i)$ je průměrná vzdálenost objektů od středu shluku a $S_n(Q_i, Q_j)$ je vzdálenost od středu shluků reprezentovaných buď centroidy nebo medoidy [48].

4.2 Klasifikace

Klasifikační úloha je úlohou zařazení objektu do jedné z předem známých tříd. Klasifikaci můžeme rozdělovat například podle principu klasifikačního algoritmu na klasifikaci pomocí vzdáleností od etalonů (reprezentativních obrazů klasifikačních tříd), pomocí diskriminačních funkcí určujících příslušnost k určité třídě nebo pomocí hranic, které oddělují jednotlivé klasifikační třídy. Klasifikací můžeme například na základě variant genomu rozhodnout o příslušnosti daného člověka k dané etnicitě nebo rozhodovat o typech karcinomu a podobně. Využití klasifikačních algoritmů je v biologii a medicíně celá řada [48].

4.2.1 Metoda *k*-tého nejbližšího souseda

Metoda *k*-tého nejbližšího souseda je zobecněná metoda nejbližšího souseda. Stejně jako tato metoda pracuje na principu určování vzdálenosti objektu od nejbližších *k*-prvků. Minimální vzdálenost můžeme zapsat jako:

$$D_{NN}(x, C) = \min D(x, x_q), \qquad x_q \epsilon C \tag{4}$$

Parametr *k* se volí zpravidla jako liché číslo, protože v případě sudého čísla by mohly nastat nerozhodné situace, kdy by nejbližší sousedi byli rovnoměrně rozloženi mezi třídami, a nebylo by možné určit příslušnost prvku k dané třídě. Tato metoda je citlivá na pozorování odlehlých hodnot a obvykle dává špatné výsledky v případě, kdy jednotlivé třídy nejsou podobně velké. Pokud by byly tyto třídy od sebe vzdálené, pak má klasifikátor k-nejbližšího souseda tendenci klasifikovat prvky do třídy s největším počtem prvků. Na druhou stranu nemá tato metoda žádné požadavky na rozložení vstupních dat a je poměrně rychlá [48].

4.2.2 Metoda podpůrných vektorů (SVM)

Metoda podpůrných vektorů (support vector machine) hledá v prostoru atributů trénovacích dat optimální rozdělující nadrovinu tak, aby body projekce trénovacích dat ležely na jejích opačných stranách poloprostorů, které odděluje a zároveň byl po obou stranách této nadroviny co nejširší pás bez bodů, čili aby bylo minimum vzdáleností bodů od této nadroviny co největší. SVM slouží k binární klasifikaci.

Pokud nejsou data lineárně oddělitelná, volí se modifikace metody podpůrných vektorů s použitím jádra. Obvykle platí, že pro data s velkým množstvím příznaků a malou množinou objektů trénovacích dat lze použít metodu podpůrných vektorů bez jádra (také označované jako SVM s lineárním jádrem), u dat s malým množstvím atributů se použije SVM s Gaussovským jádrem. To je vhodné použít u obtížně oddělitelných dat. Dalšími možnostmi je využití polynomiálního jádra a dalších, které jsou složitější než Gaussovské jádro a obvykle poskytují horší výsledky [50].

Jak už bylo zmíněno, SVM je binární klasifikátor, nicméně existují implementace, které umí provádět klasifikaci do více tříd. Výhodou metody podpůrných vektorů je, že nepředpokládá normální rozdělení dat. Nevýhodou je nutnost volby parametrů jádra a regularizačního parametru a sledování dopadů na přesnost klasifikace [48].

4.2.3 Model náhodného lesa

Model náhodného lesa využívá ke klasifikaci množinu rozhodovacích stromů. Rozhodovací stromy jsou stromy, ve kterých je každý uzel, který není listem, rozhodovacím pravidlem a větve, které z tohoto uzlu vedou, představují možné výsledky aplikace rozhodovacího pravidla uzlu. Každý list stromu je ohodnocen třídou.

Pro aplikaci klasifikace pomocí náhodného lesa je nejprve sestaven model náhodného lesa z jednotlivých stromů. Vybere se vzorek z trénovacích dat (bootstrap sampling) a na tomto vzorku se naučí strom, který není prořezáván. Během tvorby stromu neuvažujeme všechny atributy trénovacích dat, čímž se snižuje možná podobnost jednotlivých vytvořených stromů. Samotnou klasifikaci pak provedeme klasifikací dat pomocí jednotlivých stromů, přičemž zvolena je ta třída, jejíž počet byl určen jednotlivými stromy nejčastěji (tzv. hlasování). Je zde snaha minimalizovat chybu modelu jako celku, nikoliv jednotlivých stromů. Výhodou klasifikace náhodným lesem je obecně vysoká přesnost a rychlost klasifikace [51, 52].

4.2.4 Rozklad dat na trénovací a testovací data

Důležitým krokem k posouzení, zda byl klasifikační model zvolen správně, je evaluace klasifikace, tedy způsob, jakým ohodnotit a porovnat úspěšnost klasifikace. Aby mohla být evaluace klasifikátoru provedena, je potřeba definovat testovací data.

Výběr testovacích dat lze provést několika způsoby s různými dopady na výsledek hodnocení úspěšnosti klasifikace.

Resubstituce

Všechna data se použijí na natrénování klasifikátoru a z těchto samých dat se vyberou data pro testování. Takový přístup vede k problému přeučení (overfitting) a nalezené znalosti spíše vystihnou náhodné charakteristiky trénovacích dat [48].

Křížová validace (cross-fold validation)

Množina dat se rozdělí na *k* dílů. Pro trénování dat se použije *k-1* dílů a 1 díl se užije jako testovací data. Celý proces se zopakuje *k*-krát a výsledek ohodnocení se zprůměruje. Výhodou je celkem přesný výsledek, nevýhodou je vyšší časová náročnost [48].

Náhodný výběr s opakováním

Metoda je podobná metodě křížové validace s rozdílem, že se objekty vybrané pro natrénování klasifikátoru mohou opakovat. *N*-krát náhodně s opakování zvolíme *n* objektů pro učení. Na velkých datech se vybere přibližně 63% trénovacích objektů. Zbytek jsou testovací data. Výhodou této metody je, že trénovací data jsou stejně velká jako původní množina dat a oproti křížové validaci je aplikování této metody rychlejší. Nevýhodou je opakování dat [48].

Predikce externí validací

Při tomto často používaném způsobu rozdělení dat na trénovací a testovací podmnožiny, se vybere podmnožina dat (obvykle kolem jedné třetiny dat), která slouží k testování. Zbylá data jsou určena k natrénování klasifikátoru. Tento postup je možní opakovat a výsledky průměrovat. V případě opakování této metody je výhodou velká přesnost, ovšem za cenu vyšší časové náročnosti [48].

4.2.5 Přesnost klasifikace

Jedním ze způsobů, jak hodnotit úspěšnost klasifikace, je sestavení konfúzní matice (také označovanou jako matice záměn). V této matici je uvedena skutečná třída, do které objekt opravdu náleží, a výsledek klasifikace. Můžeme tak definovat objekty [48]:

- Skutečně pozitivní – objekty, které byly správně klasifikovány do příslušné třídy
- Skutečně negativní – objekty, které nenáleží do dané třídy, ale byly tak klasifikovány
- Falešně negativní – objekty, které náleží do dané třídy, ale byly klasifikovány do jiné
- Falešně pozitivní – objekty, které nenáleží do dané třídy, ale byly klasifikovány jako by do ní náležely

Díky konfúzní matici můžeme určit celkovou správnost jako podíl skutečně pozitivních a skutečně negativních objektů z celkového počtu objektů. Tato hodnota je označována jako přesnost klasifikace.

Vhodnost užitého klasifikačního modelu srovnáváme s klasifikací náhodným klasifikátorem. Správnost náhodné klasifikace určíme jako podíl nejpočetnější třídy objektů vůči celkovému počtu objektů. Výběr použitého klasifikačního modelu pak testujeme jednovýběrovým binomickým testem [48], jehož testovací statistiku lze zapsat ve tvaru:

$$z = \frac{P_A - P_R}{\sqrt{(P_R(1 - P_R))/N}} \tag{5}$$

kde P_A je přesnost zvoleného klasifikátoru a P_R je úspěšnost klasifikace náhodným klasifikátorem vyjádřená vztahem:

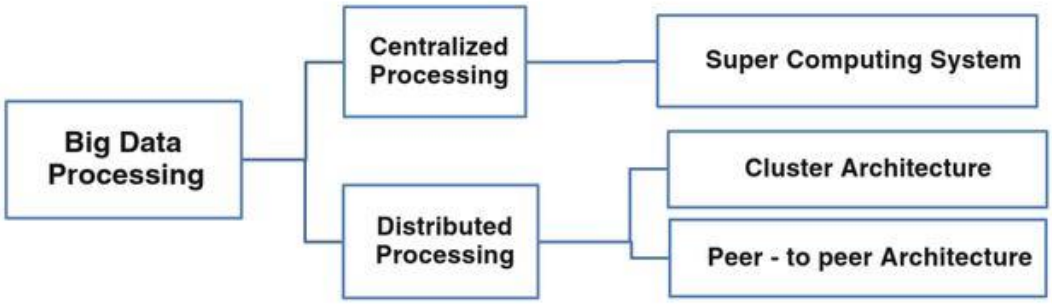
$$P_R = \frac{N_i}{N} \tag{6}$$

kde N_i je počet instancí největší třídy a N je počet všech instancí.

Nulovou hypotézu o shodnosti úspěšnosti klasifikace použitým klasifikátorem a úspěšnosti náhodného klasifikátoru zamítáme pokud $|z| > 1{,}96$.

5 Paradigmata zpracování velkých dat

V dnešní době existuje několik základních paradigmat zpracování velkých dat. V zásadě jdou rozdělit do dvou skupin. První skupinou je centralizované zpracování velkých dat, tou druhou distribuované zpracování [53, 54]. Tyto přístupy jsou rozebrány v této kapitole.



Obrázek 7: Přístupy ke zpracování velkých dat [54]

5.1 Centralizované zpracování velkých dat

Centralizované zpracování velkých dat je starším přístupem. Jedná se o způsob zpracování dat, ve kterém jsou data uložena na jednom místě a na jednom místě jsou následně zpracovávána výkonným počítačovým systémem. Centralizované zpracování velkých dat je zpravidla neškálovatelné, omezené výkonem a kapacitou daného systému. Na druhou stranu může být centralizované zpracování velkých dat přínosné na místech, kde vznikají nová data a zároveň jsou zde analyzována [53].

Pro zpracování většiny stávajících biologických datasetů je centralizovaný způsob zpracování velkých dat použitelný, neboť velikost jednotlivých datasetů není zatím příliš velká. Navíc jsou vyvíjeny způsoby, kterými by se dala analýza velkých biologických dat na jednom superpočítači zefektivnit. Například se jedná o zpracování velkých dat pomocí grafických jednotek (GPU) [55, 56].

5.2 Distribuované zpracování velkých dat

S rostoucím objemem dat přestává být centralizované zpracování dat efektivní a je třeba zvolit jiný přístup. Novějším přístupem k analýze velkých dat je distribuované zpracování dat. Základní myšlenkou distribuovaného zpracování velkých dat je rozdělení dat na části, které mohou být samostatně zpracovávány na odlišných uzlech v clusteru. V současné době existuje několik modelů, jak distribuované zpracování dat provádět. Distribuované zpracování dat využívá distribuované souborové systém. Typickým zástupcem je HDFS.

HDFS

Hadoop Distributed Files System je distribuovaný souborový systém, který je odolný vůči selhání, určený k ukládání velkých souborů. Počet souborů uložených na HDFS může být v řádu desítek miliónů. Podporuje ukládání souborů do hierarchické struktury adresářů. Data jsou organizována do bloků, typicky o velikosti 64 megabajtů. HDFS cluster má master-slave architekturu. Obsahuje uzly typu NameNode a jim podřízené uzly DataNode. Typicky se uvažuje architektura s jedním uzlem typu NameNode a více uzly typu DataNode. Pokud je uzlů typu NameNode více, pak hovoříme o HDFS federaci. NameNode obsahuje informace o uložených souborech a jejich replikách a řídí přístupy k těmto souborům. Uzly DataNode obsahují jednotlivé bloky souborů. Bloky jednoho souboru mohou být uloženy i na více uzlech [89].

5.2.1 MapReduce model

MapReduce je označení pro framework určený ke zpracování velkých datasetů. Základní myšlenkou tohoto frameworku je použití funkcí *map* a *reduce*. Open source implementace MapReduce frameworku je Apache Hadoop. Proces zpracování dat se skládá ještě z dalších kroků.

Základní kroky procesu zpracování dat pomocí MapReduce jsou [57]:

1. *Split* – vstupní data se rozdělí na části, pro každý díl je vytvořena mapovací úloha
2. *Map* – vezme vstupní díl *(klíč 1, hodnota 1)* a převede jej na dvojici *list (klíč 2, hodnota 2)*
3. *Shuffle&Sort* – jednotlivé dvojice klíčů a hodnot se seřadí a odešlou na základě hodnot klíčů na dané uzly (seskupí se podle klíče), který vykoná operaci *reduce* nad danou podmnožinou

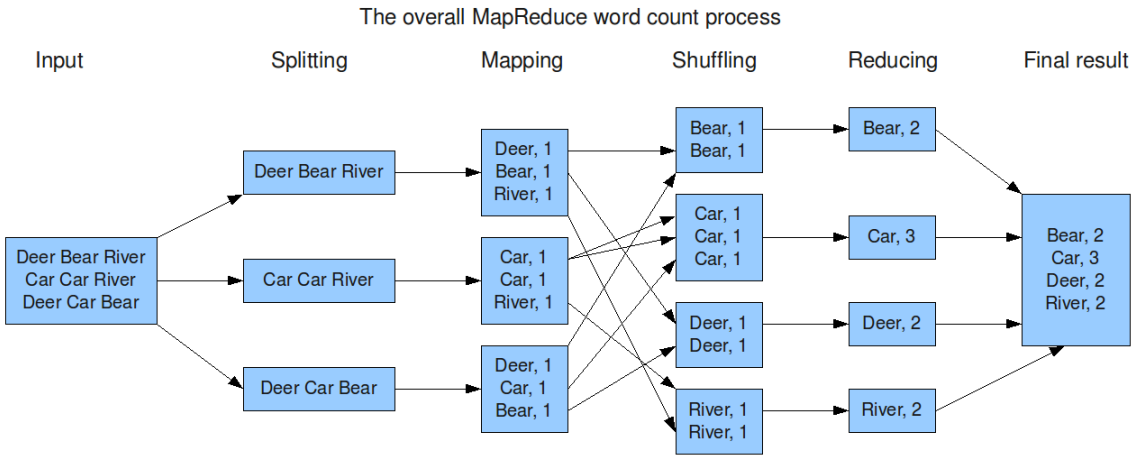
4. *Reduce* – převede seskupené dvojice (definované jako *(klíč 2, list (hodnoty2))* obecně na dvojici *(klíč 3, hodnota 3)* podle nadefinované funkce (obvykle je použita agregační funkce) [58].

Formální zápis těchto operací je uveden v tabulce níže s vysvětlivkou vztaženou k příkladu četnosti slov, který následuje v této kapitole.

Operace	Formální zápis	Vysvětlivka vztažená k příkladu počtu slov
Map	$(K_1, V_1) \rightarrow list(K_2, V_2)$	K_1 – název souboru, V_1 – obsah souboru; K_2 – slovo, V_2 – číslo „1“
Reduce	$(K_2, list\ (V_2)) \rightarrow list(K_3, V_3)$	K_3 – slovo, V_3 – četnost slova

Tabulka 4: Formální zápis operací map a reduce [59]

Posloupnost těchto kroků je znázorněna na příkladu určení četnosti slov v dokumentu (viz obrázek 8).



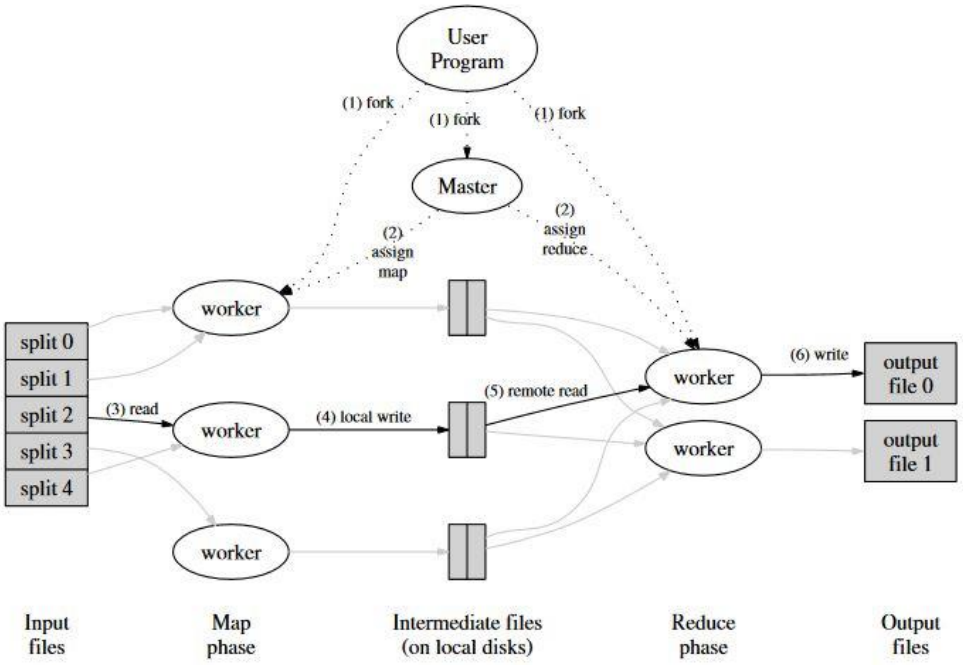
Obrázek 8: Příklad hledání četnosti jednotlivých slov pomocí frameworku MapReduce [60]

Realizace MapReduce procesu

Proces MapReduce začíná rozdělením vstupu na M dílů. Dalším krokem je nakopírování uživatelského programu v rámci clusteru. Jedna kopie programu je označena jako master, zbývající kopie jsou označeny jako workers, které jsou přiřazeny master uzlem. Master uzel je řídícím uzlem, který přiřazuje úlohy ostatním podřízeným uzlům. Master přiřadí M uzlům mapovací úlohu a R uzlům úlohu redukce. Podřízený uzel, kterému byla přiřazena mapovací úloha, načte obsah, který odpovídá přidělenému dílu dat [59].

Poté tento podřízený uzel parsuje nadefinované dvojice (klíč, hodnota) a provádí na nich operace nadefinované v mapovací funkci. Výsledky jsou udržovány v operační paměti, přičemž se periodicky zapisují na *R* umístění daných počtem uzlů, které budou provádět redukci. Umístění těchto výsledků je zasláno podřízenými uzly zpět řídícímu uzlu, který tato umístění přeposílá podřízeným uzlům, kterým byla přiřazena úloha redukce. Jakmile dojde k zaslání umístění uzlu, kterému byla přiřazena úloha redukce, začne vzdáleně načítat data, která jsou uložena na lokálních umístěních mapovacích uzlů.

Jakmile tento uzel dokončí načítání dat, seřadí hodnoty podle jejich klíčů a seskupí je. Následně iteruje seřazenými daty a pro každý klíč provede operaci redukce nad všemi jeho hodnotami a výstup zapisuje do výsledných R souborů. Řídící uzel vrátí data zpět do uživatelského programu a pokračuje v jeho dalším vykonávání ihned poté, co uzly provádějící operaci mapování a všechny uzly provádějící redukci skončily [59].



Obrázek 9: Realizace MapReduce [61]

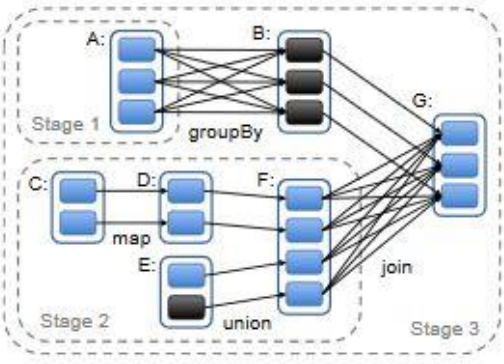
Tento způsob řízení implementovaný v Hadoopu 1 obsahuje však několik významných problémů. Jedním z problémů je nemožnost sdílet prostředky s aplikacemi, které nepoužívají MapReduce Framework. Dalším problémem je omezená škálovatelnost (maximálně zhruba 4000 uzlů). Také je možné každému uzlu přiřadit buď mapovací úlohu, nebo úlohu redukce [62].

Hadoop ve verzi 2 umožňuje jiný způsob řízení zdrojů například pomocí Apache YARN (Yet Another Resource Negotiator), což umožňuje běh aplikací, které nevyužívají MapReduce na stejném clusteru a zároveň dochází k rozumnější alokaci paměti a jader procesorů jednotlivých uzlů v clusteru [62].

5.2.2 Model orientovaného acyklického grafu

Tok dat je definován jako acyklický orientovaný graf, ve kterém vrcholy grafu představují operace, které mají být provedeny s daty [63]. Jednou z implementací využívající tento model je Apache Spark, který zčásti využívá také model MapReduce.

Apache Spark využívá dvou klíčových metod. Jednou z nich je již zmíněný model acyklického grafu, který je používán plánovačem pro sestavení postupu prováděných operací (stages). Tvorba acyklického orientovaného grafu je znázorněna na následujícím obrázku.



Obrázek 10: Tvorba DAG Apache Sparkem. Obdélníky reprezentují RDD, černé jsou již obsaženy v paměti, stage 1 se proto znovu nevykoná, čili dojde ke spuštění stage 2 a následně stage 3 [64]

Druhá klíčová součást Sparku je použití tzv. *Resilient Distributed Datasets* (RDD). RDD představují rozdělená data na části, formálně určená jen pro čtení. Nad těmito datasety je možné provádět transformace a akce.

Spark definuje například transformace [65]:

- *map* – vrátí nový dataset transformovaný podle funkce, která je parametrem této transformace (obdoba operace map Hadoopu)
- *filter* – podle funkce, jejíž návratová hodota je typu Boolean vrací záznamy, u kterých funkce vrátila true.
- *union*, intersection – spojuje dva datasety v jediný (spojením nebo v druhém případě průnikem)
- *groupByKey* – seskupí data podle klíče a vrátí dataset složený z dvojic (klíč, hodnota) vytvořený z datasetu (klíč, list (hodnoty)

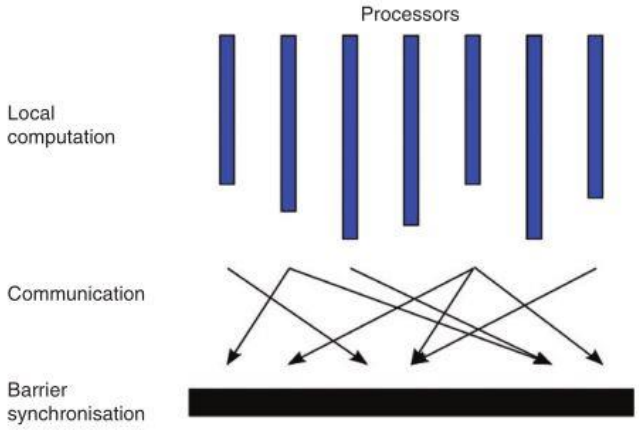
Všechny tyto transformace jsou typu lazy. K jejich provedení dochází, až když je na výsledný dataset zavolána akce [64].

Některé základní akce, které jsou ve Sparku definovány [65]:

- *reduce* – redukuje obsah datasetu použitím nadefinované funkce (např. agregační)
- *collect* – vrátí prvky datasetu jako pole
- *saveAsTextFile* – uloží dataset (RDD) jako soubor v lokálním souborovém systému, HDFS nebo v jiném souborovém systému podporovaném Hadoopem

5.2.3 BSP model

The Bulk Synchronous Parallel (BSP) model je paralelní model pro zpracování iterativních grafových algoritmů. Tento model definuje supersteps, což jsou jednotlivé dílčí úseky běhu aplikace, ve kterých dochází k současnému paralelnímu zpracování dat různými uzly clusteru. Výpočty probíhají na lokálně uložených datech každého uzlu. Mezi jednotlivými uzly může probíhat komunikace. Každý superstep je ukončen synchronizační bariérou. Workflow tohoto modelu je zobrazeno na obrázku níže [63]. Příkladem implementace nástroje pro analýzu velkých dat BSP modelem je Apache Hama [87].



Obrázek 11: BSP model [63]

Ze zmíněných modelů se dřívější implementace modelu frameworku MapReduce využívá u dávkového zpracování velkých dat. Implementace využívající orientovaného acyklického grafu jsou zaměřeny na iterativní algoritmy, zatímco implementace využívající BSP model jsou primárně určeny na analýzu grafových strukur. Zde je však nutné podotknout, že uvedené modely lze využít i pro jiné aplikace, než pro které byly primárně určeny. Například pro Apache Spark existuje rozšíření GraphX, které umožňuje efektivní zpracování grafových struktur.

6 Vybrané nástroje pro analýzu dat

K analýze dat dnes již existuje velké množství nástrojů různé kvality disponujícími různými funkcemi. Tyto nástroje lze dělit do tří generací, přičemž nejmarkantnější rozdíl je mezi první a ostatními generacemi. Nejprve jsem tedy srovnal různé generace dostupných nástrojů, přičemž v každé generaci jsem vybral některé z nich a ty mezi sebou porovnal. Mezi hlavní kritéria výběru vhodného nástroje patřily především možnosti vstupních dat, čili zda je možné data načíst ze souborů, databází, jakou můžou mít maximální velikost a jaké jsou jejich přípustné formáty. Dalším bodem srovnání byly analytické funkce nástroje, tj. kolik je jich v nástroji obsaženo a jaké mají možnosti konfigurace. Důležitým kritériem pro volbu vhodného nástroje byly i možnosti reprezentace výstupů, vizualizace dat, trendů v datech a možnost evaluace použitých analytických funkcí. Dále jsem se zaměřil na možnost rozšiřování nástroje o vlastní skripty a moduly, či možnosti přizpůsobování stávajících modulů. Zaměřil jsem se také na ovládání nástrojů, licenci a historii projektu.

PMML

Standard, kterým lze popsat model používaný pro data mining, je PMML. PMML je strukturovaný XML dokument, který obsahuje popis používaných datových typů a poté část, která je specifická pro každý druh modelu. V této části je obsažena informace, podle které je možné v aplikacích využívající PMML sestavit model (například pro klasifikaci dat).

Následuje ukázka modelu zapsaného pomocí PMML. Tento model byl vyexportován z nástroje KNIME s použitím datasetu variant lidského genomu, kterému se věnují experimenty popsané v této práci.

```
</PMML>
<!-- definice modelu -->
<TreeModel modelName="DecisionTree" functionName="classification" splitCharacteristic="binarySplit" missingValueStrategy="lastPrediction" noTrueChildStrategy="returnNullPrediction">

   <!—definuje sloupce používané v modelu-->
   <MiningSchema>
    <MiningField name="Col0" invalidValueTreatment="asIs"/>
   …
   </MiningSchema>

    <!--  reprezentuje uzly stromu a pravidla klasifikace -->
   <Node id="0" score="GBR" recordCount="165.0">
    <True/>
       <SimplePredicate field="Col83327" operator="lessOrEqual" value="2.5"/>
       <ScoreDistribution value="GBR" recordCount="1.0"/>
       <ScoreDistribution value="CHB" recordCount="0.0"/>
       <ScoreDistribution value="ASW" recordCount="2.0"/>
…
   </Node>
  </TreeModel>
</PMML>
```

Kód 6: Zápis vytvořeného rozhodovacího stromu pomocí PMML

6.1 Nástroje první generace

Část této kapitoly vychází z mého dříve vypracovaného oborového projektu.

První generace nástrojů pro zpracování velkých dat se vyznačuje především velkou kolekcí analytických algoritmů, které jsou v nich implementovány. Tyto nástroje lze používat prakticky již po jejich instalaci a nejsou u nich předpokládané další prerekvizity jako například požadavky na prostředí, ve kterém je možné je používat. Zpravidla je v nich umožněna nejenom samotná analýza dat, ale i jejich předzpracování a veškerá funkcionalita funguje *out of the box*, tedy bez nutnosti další složité konfigurace. Právě jednoduchost používání nástrojů označovaných jako nástroje první generace je jejich hlavní výhoda. Proces zpracování dat lze v těchto nástrojích zpravidla jednoduše znázornit pomocí diagramů označovaných jako scientific workflows.

Problém využívání těchto nástrojů však spočívá především v jejich škálovatelnosti, která je pouze vertikální, nikoliv horizontální [63]. Z tohoto důvodu jsou tyto nástroje vhodné pro zpracování velkých dat dle centrálního paradigmatu. Tyto nástroje nejsou také odolné vůči selhání. Jakmile během analýzy selže nějaká komponenta nástroje, selže i zbytek procesu (single point of failure).

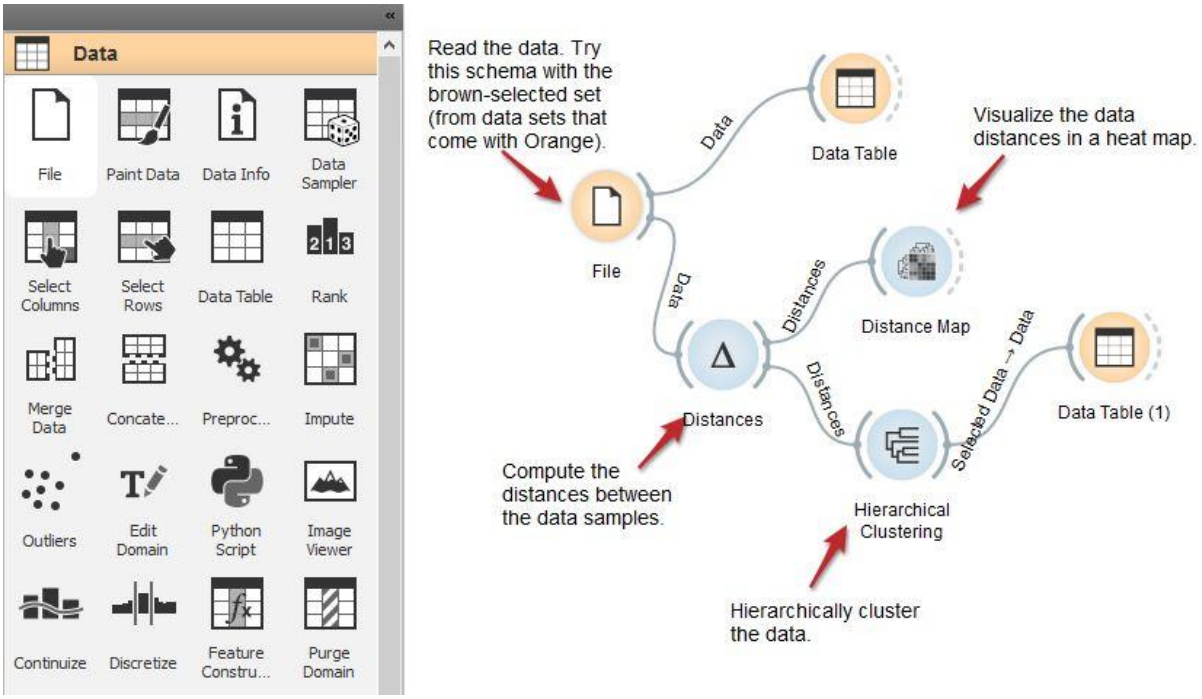
Nemožnost již zmíněného horizontálního škálování se snaží tvůrci těchto nástrojů řešit integrací nástrojů druhé anebo třetí generace. V praxi se jedná o možnost používat komponenty nástrojů třetí generace a kombinovat je s komponentami nástroje první generace.

Scientific workflows

Jednotlivé kroky analýzy dat a část životního cyklu dat lze modelovat pomocí procesů (scientific workflows). Tento přístup umožňuje vytvářet přehledné modely, které lze používat opakovaně. Analýzu dat lze rozdělit na jednotlivé kroky, snadno ji modifikovat, a vytvářet tak nové modely. Modelování procesů také usnadňuje vzájemnou spolupráci více lidí na vytváření nového modelu, umožňuje debugování jednotlivých částí procesu a změnu dílčích kroků, aniž by bylo třeba spouštět celý proces od začátku. Díky těmto výhodám jsem možnost grafického modelování začlenil mezi hodnocené kritérium vybraných programů.

6.1.1 Orange

Orange je open source nástroj, který umožňuje předzpracování, analýzu a vizualizaci dat. Jedná se o nástroj vyvíjený Bioinformatickou laboratoří univerzity v Ljubljani, uvolněný pod licencí GNU GPL 3. Samotná aplikace je napsána jako komponentová aplikace v Pythonu, díky tomu je možné v aplikaci používat vlastní Python skripty a také do ní přidávat vlastní komponenty zvané widgety. [66] V současnosti má aplikace třetí verzi a její vývoj dále pokračuje, nicméně některá rozšíření vyvíjená třetími stranami nejsou k dispozici a odkazy na ně jsou nefunkční.



Obrázek 12: Workflow v Orange

Vstupní data

Orange umožňuje načítat soubory prostřednictvím widgetu File. Soubory obsahují buď data oddělená tabulátory nebo čárkami nebo soubory ve fromátu ARFF, což je formát souborů používaný Wekou (viz dále). Orange oficiálně neumožňuje přímé načítání dat z databází.

Orange umožnuje uživateli rozlišit, zda se ve vstupních datech nacházejí chybějící hodnoty, které nezná nebo které jej v rámci analýzy nebudou zajímat. Orange je definuje jako „don't know" a „don't care" hodnoty. Pokud jsou data načítána z více různých souborů, je možné rozhodnout, jak se tyto atributy načtou (například se přidají jako nové, použijí se stávající, atd.) [66]

Předzpracování vstupních dat

Orange umožňuje předzpracovávat vstupní data. Umožňuje spojování dat, rozdělování dat, filtrování podle hodnot atributů, vybrat pouze atributy, se kterými chceme dál pracovat. Dále je možné převádět spojitá data na diskrétní a diskrétní na spojitá a provádět další modifikace. [66]

Analytické algoritmy

V základní sadě widgetů se nacházejí algoritmy pro tvorbu klasifikačních stromů, hledání nejbližších sousedů, shlukování podle průměrů, naivní Bayesův klasifikátor, hierarchické clusterování a několik dalších algoritmů. Aplikace obsahuje také widgety pro zjišťování trendu a pro předpovídání hodnot. Počet analytických widgetů je však menší ve srovnání s dalšími nástroji popsanými dále (viz tabulka 5). [66]

Vizualizace dat a výstup dat

Aplikace umožňuje uložit výstupy jako hodnoty oddělené čárkou nebo tabulátorem nebo ve formátu Python Pickle. Graficky je možné data vizualizovat pomocí různých grafů, například pomocí bodového diagramu, heat mapy, Vennových diagramů a pomocí dalších způsobů. Grafy je možné ukládat jako obrázky PNG nebo SVG. [66]

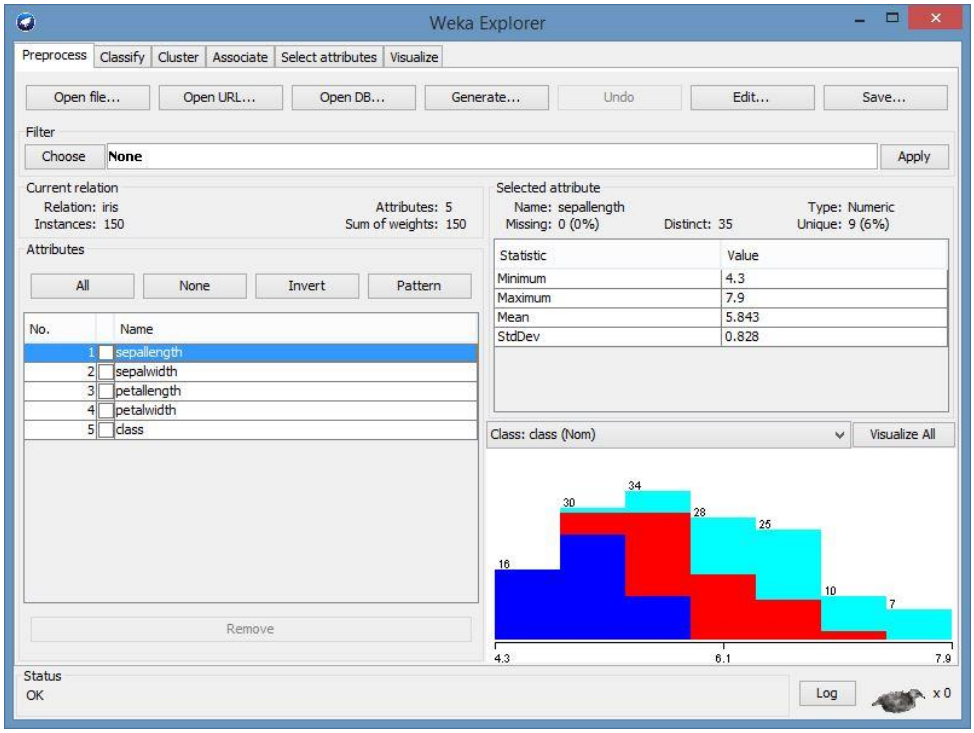
6.1.2 Weka

Weka je open source nástroj pro data mining implementovaný v Javě univerzitou Waikato. Aplikace umožňuje pracovat s daty prostřednictvím grafického klienta, příkazového řádku, klienta pro tvorbu workflows a voláním z externích programů přes API.

Jedná se o open source aplikaci uvolněnou pod licencí GNU GPL. Současná verze Weky je 3. 7. 13 ze září 2015. [67]

Ovládání aplikace

Aplikace nabízí po svém spuštění GUI Chooser, který slouží pro otvírání jednotlivých částí programu. V režimu Explorer je možné provádět úpravy a analýzu dat krok po kroku a zaznamenávat si mezivýsledky. Pro komplexnější analýzu je vhodnější použít KnowledgeFlow, což je nástroj pro tvorbu workflows, který umožňuje jednoduše metodou drag&drop vkládat objekty na plochu a propojovat je. Nástroj Experimenter umožňuje provádět experimenty, které spočívají ve statistickém benchmarku více analytických algoritmů proti různým sadám dat. Simple CLI (Command Line Interface) pak spouští příkazovou řádku. [67]



Obrázek 13: Weka Explorer

Vstupní data

Aplikace umožňuje načítání dat ze souborů ve vlastním formátu ARFF, případně v jeho XML verzi XRFF. ARFF formát obsahuje hlavičku, která definuje atributy dat a jejich datové typy a pak samotná data, XRFF je potom reprezentace ARFF pomocí XML. Weka umožňuje pracovat se vzdálenými daty prostřednictvím protokolu HTTP/S.

Weka dále umožňuje připojení k databázím prostřednictvím JDBC. Podporovány jsou všechny běžné typy relačních databází. [67]

Předzpracování vstupních dat

Předzpracování vstupních dat je ve Wece na vysoké úrovni. K dispozici jsou desítky různých filtrů a modulů pro filtrování a předzpracování dat. Weka umožňuje čištění dat, určování jejich relevance, generalizaci a normalizaci dat. [67]

Analytické algoritmy

Weka obsahuje velké množství analytických algoritmů. Jsou v ní zahrnuty algoritmy rozhodovacích stromů (např.: C4.5, Hoeffedingův strom, M5), statistické algoritmy (např.: Bayesův naivní klasifikátor, Bayesovské sítě), shlukovací algoritmy využívající vzdáleností (např.: k-průměry) a další kombinace zmíněných algoritmů.

Aplikace umožňuje také zjišťování trendů v datech pomocí regrese. Některé z algoritmů lze spouštět v inkrementálním módu a tím zpracovávat i velké objemy dat. Weka umožňuje provádět evaluaci vybraných analytických metod.

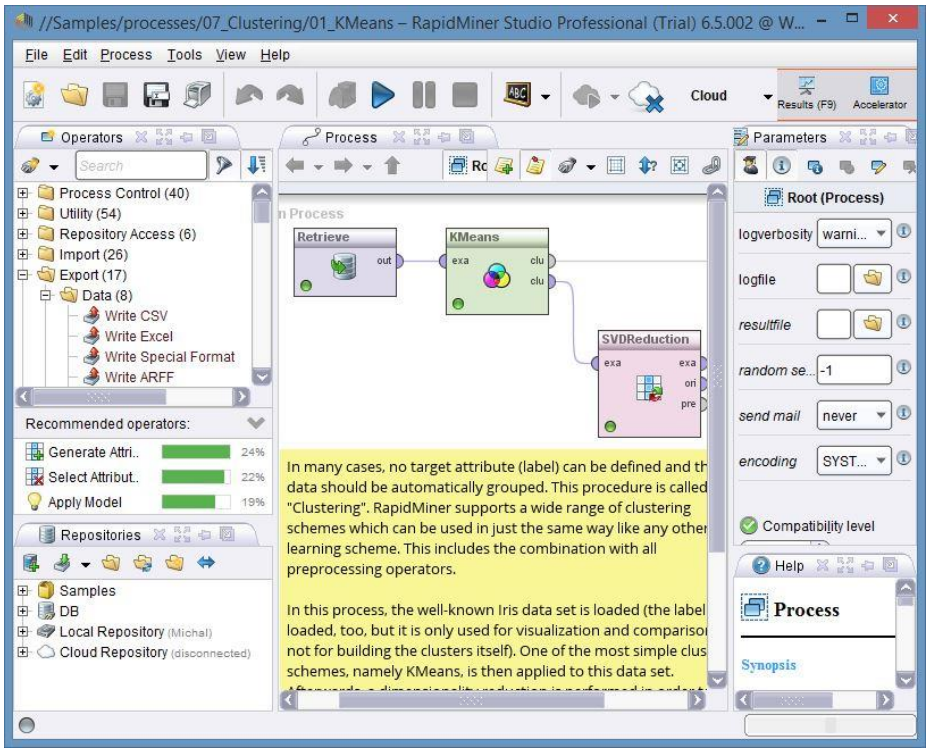
Funkčnost aplikace lze rozšířit instalací nových balíčků, čímž lze aplikaci doplnit například o chybějící algoritmy využívající neuronové sítě. Některé z balíčků vyžadují instalaci dalších nástrojů třetích stran, které mohou mít odlišné licenční podmínky použití. Do aplikace lze přidávat i vlastní implementované balíčky. [67]

Vizualizace dat a výstup dat

Data lze v aplikaci vizualizovat pomocí různých grafů, například rozptylovou mapou. Výsledky lze ukládat do souborů ARFF, XRFF, CSV, JSON a případně do databázových tabulek. Weka umožňuje do souboru uložit i například model C4.5 stromu. [67]

6.1.3 RapidMiner Studio

RapidMiner Studio je nástroj pro analýzu dat, který je k dispozici zdarma pod licencí AGPL nebo pod komerční licencí, která obsahuje další funkce. Komerční verze podporuje více způsobů získávání vstupních dat (komerční databáze, některé další vstupní formáty dat, cloudové služby, NoSQL, Hadoop), možnost z dat odebrat osobní údaje, možnost provádět výpočty v cloudu a další. K bezplatné verzi je možné se zdarma registrovat, a získat tak možnost rozšiřovat funkcionalitu aplikace prostřednictvím dalších doplňků z marketplace. Tyto doplňky mají různá licenční omezení. Po registraci je také k dispozici našeptávací funkce, která na základě doporučení komunity navrhuje který nástroj použít. [68]



Obrázek 14: RapidMiner

Vstupní data

V bezplatné verzi je možné jako vstupní data používat soubory ve formátech XML, XRFF, CSV, XLS, načítat data z opensource relačních databází a to i inkrementálně. Data lze načíst i ze vzdáleného umístění. [68]

Předzpracování dat

RapidMiner nabízí velké množství nástrojů pro předzpracovávání dat. Umožňuje transformace dat, převody datových formátů, diskretizaci dat, výběr atributů, generování hodnot atributů, výběr dat, nahrazení chybějících hodnot, samplování dat, rozdělování dat, množinové operace, řazení dat a agregaci dat. [68]

Analytické algoritmy

RapidMiner umožňuje klasifikaci dat pomocí systémů klasifikačních pravidel a rozhodovacích stromů. Dále obsahuje shlukovací algoritmy využívající vzdálenosti v datech, statistické algoritmy jako například Bayesův naivní klasifikátor, neuronové sítě, algoritmy pro zjišťování trendů v datech, SVM a další.

Z marketplace je možné stahovat analytické nástroje programu Weka a integrovat je do aplikace. K dispozici jsou také nástroje pro zpracovávání textu, analýzu signálů, zahrnující některé transformace (Wavelet, Fourier), či dolování dat z webu (například z kanálu RSS). Do aplikace lze také přidávat vlastní skripty psané ve skriptovacím jazyce R. Aplikace umožňuje validaci modelu a evaluaci provedených analýz. [68]

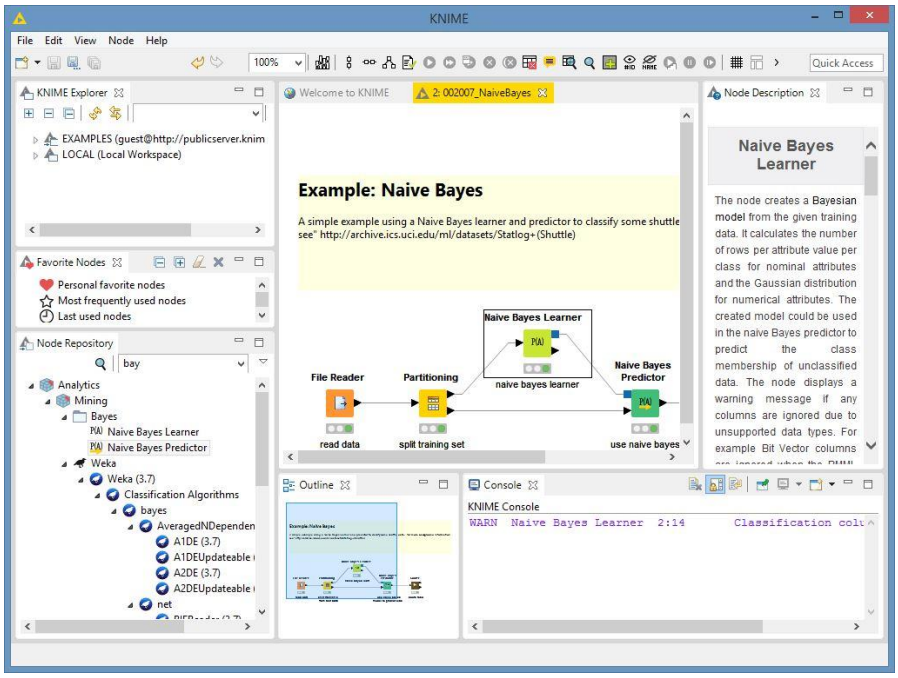
Vizualizace dat a výstup dat

Výstupy aplikace lze ukládat jako soubory CSV, XLS, XRFF, ARFF, či ve vlastním definovaném formátu. Dále lze výstupy ukládat do databáze. Některé výstupy lze ukládat jako grafy k zobrazení v aplikaci GNUPlot. Výsledky analýz lze také zobrazit v grafech přímo v aplikaci.[68]

6.1.4 KNIME

KNIME je open source data mining aplikace vystavěná na Eclipse. Aplikace je uvolněná pod licencí GNU GPL. Funkčnost aplikace je možné rozšiřovat pomocí dalších pluginů, ale k některým je potřeba zakoupit licenci. Aplikace je stále vyvíjena, aktuálně se nachází ve verzi 3.0.0 vydané v říjnu 2015.

Ovládání KNIME je intuitivní, jednotlivé nástroje jsou rozděleny do kategorií, lze mezi nimi vyhledávat. Každý modul obsahuje i stručné informace o konfiguraci. Aplikace si pamatuje naposledy používané moduly a případně lze moduly přidávat do oblíbených a přistupovat k nim rychleji. Pro rychlejší seznámení s aplikací je k dispozici nápověda v podobě článků a demonstrativních ukázek. [69]



Obrázek 15: KNIME

Vstupní data

Aplikace umožňuje číst vstupní data buď ze souborů anebo z databáze. Podporované formáty souborů jsou ARFF, CSV, XLS, XML, JSON a případně lze nadefinovat formát vlastní. Soubory lze číst i ze vzdáleného umístění. Data lze také načítat z relačních databází, podporovány jsou všechny běžné relační databáze. Podporována je také nerelační dokumentová databáze MongoDB. Přistupovat lze také ke službě Google Analytics a příspěvkům na Twitteru. [69]

Předzpracování dat

Z načtených dat lze filtrovat atributy a jednotlivé záznamy podle různých kritérií. Aplikace umožňuje agregaci dat podle atributů, rozdělování a spojování dat a množinové operace. Data lze normalizovat a denormalizovat. Chybějící hodnoty atributů lze doplňovat podle několika různých strategií (průměr, klouzavý průměr, minimální nebo maximální hodnota, lineární interpolací, …). Data lze řadit a vzorkovat. [69]

Analytické algoritmy

KNIME v sobě integruje analytické algoritmy Weky. K dispozici je také velké množství algoritmů vlastních. Ve vlastní implementaci jsou zahrnuty algoritmy rozhodovacích stomů a lesů, neuronových sítí, shlukovacích algoritmů (podle průměrů, hierarchické shlukování, …), hledání asociačních pravidel a další.

Zahrnuty jsou také algoritmy pro analýzu textu, testování hypotéz, analytické algoritmy používané v chemii a algoritmy pro analýzu časových řad. KNIME umožňuje rozšiřování funkcionality stahováním dodatečných pluginů nebo pomocí vlastních skriptů psaných v Javě, Pythonu a v jazyce R. [69]

Vizualizace dat a výstup dat

V aplikaci lze data zobrazit v různých typech grafů. K dispozici je knihovna JFreeChart, případně vlastní implementace grafů, z nichž některé jsou interaktivní (histogram, koláčový graf). Data lze zobrazovat také v grafech, které používají javascriptový kód, případně použít pro zobrazení vlastní JS skript. Aplikace umožňuje ukládat výstupní data jako XLS, PDF, CSV, ARF, HTML soubory, zapisovat výsledky do relační databáze nebo je ukládat jako JSON objekty, případně ukládat změněné předtím načtené dokumenty do MongoDB. [69]

6.1.5 Srovnání nástrojů

Následující tabulka shrnuje podstatné funkce vybraných aplikací a srovnává je. Funkčnost, která je v aplikaci zahrnuta je, označena „+“, omezená funkčnost je značena „O“ s vysvětlivkou v závorce nebo bez vysvětlivky, pokud je třeba aplikovat vlastní skript, aby bylo možné použít vybranou funkčnost, „P“ je označena funkčnost, která vyžaduje instalaci pluginů třetích stran, „W“ je označena funkčnost, kterou v sobě nástroj integruje pomocí knihoven Weky a „-“ je označena funkce, která v nástroji úplně chybí a nelze ji získat ani instalací dalších rozšíření.

Kategorie	Funkce	RapidMiner	Weka	Orange	KNIME
	Licence	AGPL	GPL	GPL	GPL
	Čtení databází	O (MySQL a PostgreSQL)	+	O (prototyp)	+ (včetně MongoDB)
	XLS	+	-	-	+
Předzpracování dat	Filtrování	+	+	+	+
	Diskretizace	+	+	+	+
	Normalizace	+	+	+	+
	PCA	+	+	+	+
	ICA	+	-	-	-
	MDS	-	+	+	+
	SVD	+	+	P	+
Rozhodovací stromy	ID3	W	+	+	W
	C4.5	W	+	+	W
	CART	W	+	+	W
Bayeské sítě	Naivní B.	+	+	+	+
Trénování podle instancí	k-NN	+	+	+	+
	LWL	W	+	-	W
Funkční analýza	Regresní analýza	+	+	+	+
	ANN	+	+	+	+
	SVM	+	+	+	+
	Bayesiánské stromy	W	+	-	W

Kombinované hybridní algoritmy	Funkční stromy	W	+	-	W
Učení	AdaBoost	+	+	+	W
	Náhodný les	+	+	+	+
	Bagging	+	+	+	W
	Stacking	+	+	-	W
Shlukování	k-průměry	+	+	+	+
	x-průměry	+	+	-	W
	Fuzzy c-průměry	-	-	P	+
	EM shlukování	+	+	-	W
	DBSCAN	+	+	-	W
Asociační pravidla	GSP	+	+	-	W
	Apriori	W	+	+	W
Další funkce	Analýza časových řad	P	O	-	+
	Datové streamy	+	P	-	+
	Analýza textu	P	O	P	P
	Paralelizace	+	O	-	+
	Integrace nástrojů další generace	2. a 3. gen.	2. gen.	-	2. a 3. gen

Tabulka 5: Srovnání funkčnosti vybraných data mining nástrojů [70]

6.2 Nástroje druhé generace

Nástroje druhé generace pro analýzu velkých dat vznikly z důvodu omezené škálovatelnosti nástrojů generace první. Jak již bylo zmíněno v kapitole o paradigmatech ve zpracování velkých dat, přestává být centralizované zpracování obecně efektivní (vyjímaje například některých metod využívajících grafické čipy), proto tyto *nástroje* (ve své podstatě se jedná spíše o knihovny) implementují distribuované zpracovávání velkých dat prostřednictvím Hadoopu s využitím frameworku MapReduce. To s sebou přináší i další výhodu v podobě odolnosti vůči selhání. [63]

Tyto nástroje stejně jako původní verze Hadoopu nedokáží příliš efektivně zpracovávat iterační úlohy [63], avšak s tím, jak se vyvíjel Hadoop, se vyvíjely i tyto nástroje a postupně přešly od používání Hadoopu verze 1 k novější variantě, což vlastně znamená, že tato generace nástrojů postupně přechází do generace třetí.

Používání těchto knihoven již není tak jednoduché jako v případě nástrojů první generace. K jejich používání je potřeba psát skripty, které volají funkce obsažené v těchto knihovnách, k tomu je také nutné transformovat data dle definovaných vstupních parametrů těchto funkcí (v případě knihovny Apache Mahout) nebo lze využít model definovaný pomocí PMML (v případě použití Cascading Patterns). V obou případech je potřeba načítat vstupní data z distribuovaného souborového systému a načtená data případně posléze transformovat.

Jednou z dalších velkých nevýhod oproti nástrojům první generace je malé množství již existujících implementovaných algoritmů a nutnost vlastní implementace většiny algoritmů použitých pro zpracování dat. [63]

6.2.1 Apache Mahout

Apache Mahout je open source knihovna (vydaná pod licencí Apache Licence 2) implementující některé algoritmy strojového učení patřící do druhé generace nástrojů pro zpracování velkých dat. V současné době není její používání vázáno jen na použití Hadoopu, ale je možné ji využít například v kombinaci se Sparkem. Množství implementovaných algoritmů se ale mezi jednotlivými kombinacemi liší (viz tabulka 6). Apache Mahout umožňuje horizontální škálování.[71]

Algoritmus	Hadoop	Spark
Klasifikace		
Naivní Bayesův klasifikátor	X	X
Náhodný les	X	-
Shlukování		
K-průměry	X	X
Fuzzy k-průměry	X	X
Modifikované k-průměry pro velké datasety (streaming k-means)	X	X
Spektrální shlukování	-	X
Redukce dimenzionality dat		
PCA	X	X
SVD	X	X
QR dekompozice	X	X

Tabulka 6: Algoritmy implementované v knihovně Mahout [71]

Používání Apache Mahout je možné pomocí příkazů z terminálu nebo je možné volat funkce implementované v Mahoutu z kódu psaného v programovacím jazyce Java.

6.2.2 Cascading

Dalším nástrojem druhé generace je Cascading s implementací algoritmů strojového učení v rozšíření Cascading Pattern. Open source projekt Cascading je stejně jako Cascading Pattern uvolněn pod licencí Apache Licence 2 [72, 73]. Cascading Pattern umožňuje horizontálně škálovatelné zpracování dat v clusteru. [74]

Cascade Pattern v současné době podporuje následující typy PMML modelů [74]:

- Hierarchické shlukování
- Shlukování k-průměry
- Lineární regrese
- Logistická regrese
- Model náhodného lesa

Použití PMML modelů je užitečné tam, kde existuje již stávající model, který je používaný jiným nástrojem (například nástrojem z první generace).

6.3 Nástroje třetí generace

Tyto *nástroje* (opět přesněji knihovny) jsou podobné nástrojům druhé generace. Stejně jako předešlé nástroje umožňují horizontální škálování a tedy distribuované zpracování velkých dat. A stejně tak potřebují využívat distribuovaný souborový systém (například HDFS). Oproti předešlé skupině však poskytují často vyšší výkon v iteračních úlohách a větší počet implementovaných algoritmů. Navíc jsou navrženy tak, aby je bylo snazší používat.

6.3.1 Apache Spark MLlib

Apache Spark je vydáván pod licencí Apache Licence 2, stejně tak je licencováno jeho rozšíření MLlib. MLlib je knihovna implementující algoritmy strojového učení a některé statistické algoritmy využívající Apache Spark, čímž je zajištěna jejich horizontální škálovatelnost. [75]

MLlib je navržen tak, aby implementované algoritmy bylo možné volat po sobě bez nutnosti provádět složité transformace dat mezi jednotlivými voláními. Tím je zajištěn způsob, kterým lze budovat workflow pro komplexní analýzu dat. Základními koncepty jsou používání datového typu DataFrame a algoritmů typu Transformer a Estimator.

DataFrame reprezentuje dataset (množinu instancí a jejich atributů ve sloupcích). Transformers jsou pak algoritmy, které převedou jeden DataFrame na jiný implementací metody transform. Typicky se jedná o převody atributů a modely algoritmů strojového učení. Estimator je pak abstrakce algoritmů, které se používají pro sestavení modelu z trénovací množiny dat. Posloupnost volání tříd typu Estimator a Transformer pak tvoří workflow označovaný jako Pipeline. Pipeline je vyjádřen jako skript napsaný v některém z podporovaných jazyků. Spark v současnosti podporuje programovací jazyky Scala, Python a Java. [75]

Oblast použití	Příklady implementovaných algoritmů
Transformace dat	Tokenizer, n-gram, binarizer, PCA, normalizer, DCT, indexace řetězce,
Binární klasifikace	Lineární SVM, logistická regrese, rozhodovací stromy, model náhodného lesa, naivní Bayesův klasifikátor, BT
Klasifikace mezi více třídami	Logistická regrese, rozhodovací stromy, model náhodného lesa, naivní Bayesův klasifikátor
Regrese	Metoda nejmenších čtverců, rozhodovací stromy, model náhodného lesa, isotonická regrese, hřebenová regrese
Shlukování	k-průměry (různé varianty), GMM, PIC, LDA
Redukce dimenzionality	SVD, PCA

Tabulka 7: Příklady implementovaných algoritmů v knihovně MLlib pro Apache Spark [75]

MLlib nabízí implementace několika populárních algoritmů pro transformace dat, klasifikaci a shlukování. Apache Spark je ale širší platforma, na kterou je možno implementovat algoritmy vlastní nebo používat již hotové balíčky třetích stran. Jedním z takových užitečných balíků je SparkNet. SparkNet je implementace deep learningových algoritmů, implementovaných ve frameworku Caffe. Tato implementace umožňuje používat některé algoritmy frameworku Caffe paralelně na více uzlech clusteru. [76]

6.4 Srovnání jednotlivých generací

V předešlé části jsou prezentovány některé významné nástroje užívané k analýze velkých dat. Jednotlivé generace nástrojů vycházejí z odlišných paradigmat zmíněných v předešlé kapitole. Zatímco nástroje první generace jsou určeny především pro centralizované zpracování velkých dat, nástroje druhé a třetí generace se zaměřují na distribuovaný způsob zpracování velkých dat (ačkoliv je možné je méně efektivně využít i pro centralizované zpracování velkých dat).

Ve všech generacích nástrojů se nicméně objevují trendy, které mění využívání těchto nástrojů. Druhá generace nástrojů přechází v generaci třetí a místo zpracování velkých dat Hadoopem verze 1 (s tradičním pojetím frameworku MapReduce) přechází na používání například Apache Sparku. Protože je však nutné jejich původní implementace přepisovat, postrádají nástroje této generace jak celou řadu funkcí (oproti třetí generaci), tak i výkon. Ukazuje se, že vývoj těchto nástrojů se postupně utlumuje. Naopak nástroje třetí generace se vyvíjejí rychle a jejich funkcionalita se postupně rozšiřuje.

Nástroje třetí generace však není snadné používat a toho se snaží využít nástroje první generace, které se v současné době snaží o integraci s nástroji třetí generace, a tím pádem nabídnout i možnost horizontálního škálování ve snaze zachovat jednoduchost používání, kterou tyto nástroje nabízí.

	1. generace	**2. generace**	**3. generace**
Významní zástupci	KNIME, Weka, RapidMiner	Mahout, Cascading s Cascading Pattern	Apache Spark s MLlib a dalšími rozšířeními
Paradigma zpracování dat	Primárně centralizované	Primárně distribuované	Primárně distribuované
Škálování	Vertikální	Horizontální pomocí Hadoopu (původně)	Horizontální s/bez Hadoopu
Tendence	Integrace s nástroji 3. generace, vlastní distribuované zpracování	Splynutí s 3. generací	Rozšiřování funkčnosti, hlubší analýza dat
Odolnost vůči selhání	Jediný bod selhání	Odolné vůči selhání	Odolné vůči selhání

Tabulka 8: Srovnání nástrojů podle jednotlivých generací [63]

6.5 Volba nástroje pro provádění experimentů

Zvolená databáze variant lidského genomu obsahuje data uložená jako VCF soubory. Tato databáze nenabízí žádné online nástroje pro jejich analytické zpracování. Je potřeba je nejprve stáhnout (například přes HTTP). Vzhledem k tomu, že stažená data budou uložena na jednom místě a zároveň je jejich celková velikost maximálně v řádu stovek GB, jeví se jako možné použít nástroje první generace, které nabízí možnost centralizovaného zpracování velkých dat, což je způsob, který by bylo možná vhodné použít, pokud se budou analýzy provádět pro jednotlivé chromozomy (data jednoho chromozomu jsou velká zhruba 20 GB). Pro analýzy prováděné nad celou množinou chromozomů by bylo pak možné využít integraci těchto nástrojů s nástroji generace třetí a provádět analýzy pomocí nich.

Vzhledem k velkému množství analytických nástrojů, integraci WEKY, podpory mnoha vstupních formátů soborů, relačních databází a nerelační databáze a díky integraci možnosti psaní vlastních pluginů v R, Pythonu nebo Javě a vlastní vizualizaci pomocí JavaScriptu, se jako vhodný kandidát jeví KNIME. Experimenty v něm lze navíc vyjádřit graficky v přehledných workflows.

KNIME také nabízí integraci Apache Sparku a tvrdí, že jej lze horizontálně škálovat a pak také nabízí přímo připojení k Hadoopu a jeho distribuovanému souborovému systému. Lze tak kombinovat jednotlivé přístupy a provádět některé experimenty nebo jejich dílčí kroky distribuovaně a zároveň využít bohatou funkcionalitu, kterou KNIME ve svém jádru nabízí.

7 Analýza variant lidského genomu

Vzhledem ke klesající ceně sekvencování lidského genomu vzrůstá počet celých nasekvencovaných genomů. Tyto sekvence jsou dále zpracovávány například tak, že se hledají v daném genomu variace oproti referenčnímu genomu. Výsledkem zpracování je nalezení tzv. variant calls, což jsou diference nukleotidu na daných pozicích. [77] Zkoumání variací genomů je užitečné v několika oblastech biologického a medicínského výzkumu. Lze jej využít například na [77, 78]:

- Hledání genetických asociací souvisejících s chorobami.
- Hledání rakovinných mutací.
- Vysvětlení diverzity lidské populace a jejího rozložení.
- Hledání chybějících variant genomů.

V následujících experimentech se věnuji klasifikaci variací části lidského genomu (konkrétně variacím části dvacátého chromozomu) podle etnicity. Tento případ užití byl vybrán na základě dostupných metadat a také na základě jednoduché struktury dat oproti datům jiných domén. Experiment lze například využít pro vysvětlení rozložení lidské populace na základě klasifikace neznámých jedinců do známých populací.

7.1 Centralizovaný způsob klasifikace variant genomů

V tomto experimentu provedu klasifikaci různě velkých vzorků několika klasifikačními algoritmy a provedu jejich srovnání. Cílem experimentu je ověřit, zda lze pro danou klasifikační úlohu použít některé algoritmy prezentované v kapitole 4, a zjistit přesnost klasifikace pomocí těchto použitých modelů. Další cíl tohoto experimentu je objektivní posouzení, zda je možné provádět experimenty tohoto typu pomocí centralizovaného paradigmatu s využitím nástroje KNIME a jeho *out of box* funkcí.

7.1.1 Vstupní data

Vstupní data použitá v tomto experimentu tvoří variace genomu§ které jsou uloženy podle formátu VCF. Tato data jsou dostupná z webu projektu 1000 Genomes [79]. Soubor je sestaven z variant 2504 lidí z 26 identifikovaných etnicit a obsahuje zhruba 1,8 milionu variací pro dvacátý chromozom. Projekt obsahuje data zabalená v archivech, která je nutno dekomprimovat na disk. Formát VCF obsahuje hlavičku s komentáři popisující soubor, ty jsou jednořádkové a označené znakem #. Soubor obsahuje data oddělená tabulátorem.

#CHROM	POS	ID	REF	ALT	QUAL	FILTER	INFO	FORMAT	HG00096	HG00097	HG00099
20	60343	rs527639301	G	A	100	PASS	AC=1;AF=0.000199681;AN=5008;…;AA=.|||;VT=SNP	GT	0|0	0|0	0|0

Data jsou uspořádána tak, že každý řádek obsahuje danou variantu a sloupce obsahují informace o variantě a její výskyt u měřeného jedince. Samotné varianty jsou popsány řetězcem ve formátu 0|0, 0|n, n|0, n|n, kde n je libovolné přirozené číslo označující alternativní alelu a 0 označuje referenční alelu.

7.1.2 Načtení vstupních dat a jejich transformace

Vstupní data jsou načtena pomocí uzlu *File*. V dialogu stačí nadefinovat znak komentářů a KNIME provede autodetekci, kterou zjistí, jakým způsobem jsou data oddělena, a přečte hlavičky sloupců. Načtená data obsahují atributy, které v analýze nejsou potřeba, a můžeme je z analýzy vypustit. Z důvodu zrychlení následujících operací je tak vhodné učinit na začátku workflow. Odstranění sloupců provede uzel *Filter Columns* s nastavením sloupců, které chceme do dat zahrnout. Vzhledem k tomu, že se data nachází ve formátu VCF, který pro tento experiment obsahuje zbytečná data v několika prvních sloupcích, lze je vyjmout již během procesu načítání dat v pokročilém nastavení uzlu *File*. To vede k vyšší rychlosti načítání dat a samozřejmě k menšímu objemu dat uložených v operační paměti.

Data obsahují chybějící hodnoty, které je nutné ošetřit. Protože máme dostatek variant, můžeme varianty, které jsou neúplné, vyfiltrovat. Za tímto účelem využijeme uzel *Missing Value* s nastavením odstranění řádků s chybějícími daty.

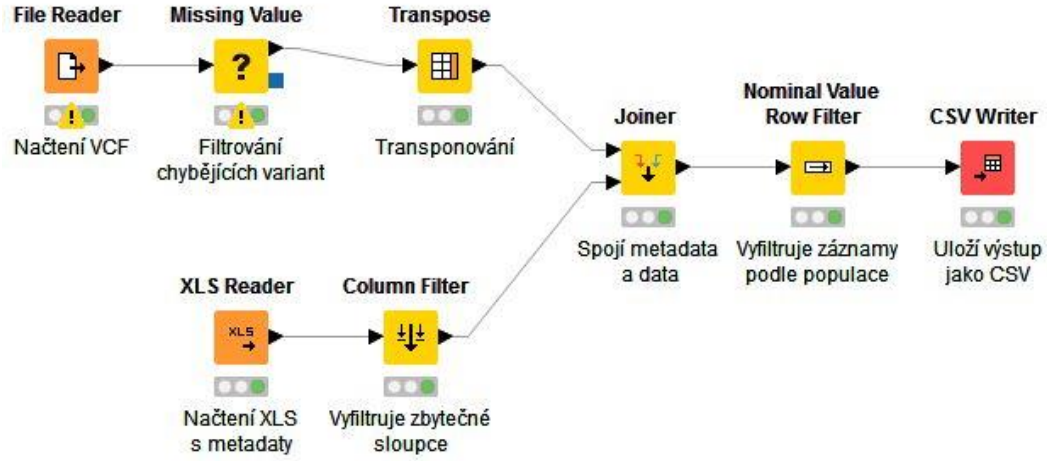
Protože v datech chybí informace o příslušnosti k etnicitě dané osoby, je potřeba tyto údaje spojit s těmito metadaty. Metadata lze stáhnout ze stejného umístění jako data. Jedná se XLS soubor obsahující identifikátor subjektu a data o něm. Identifikátor se nachází v prvním sloupci a data o etnicitě ve druhém.

Načtení dat provede uzel *XLS Reader*, ve kterém zvolíme pouze, že chceme načítat zmíněný rozsah dat a určíme ID řádku jako první sloupec dokumentu a hlavičku sloupců jako jeho první řádku.

Před spojením metadat s daty je potřeba transponovat data, protože data obsahují záznamy o variantách v řádcích a subjekty ve sloupcích, čili varianty jako instance a subjekty jako atributy. Cílem experimentu je ale klasifikovat osoby do etnických tříd podle variant částí genomů (tj. varianty alel na různých pozicích jsou v tomto experimentu chápány jako atributy a osoby jako instance). Transponování dat provede uzel *Transpose* a samotné spojení dat uzel *Join*, který nastavíme na spojení přes identifikátory záznamu a jako druh spojení zvolíme vnitřní spojení.

V experimentu [80], který zpracovává VCF soubory, jsou zpracovávána dat ze tří etnicit z různých superpopulací. Po jeho vzoru se v tomto experimentu omezíme také na zmíněné etnicity. Z tohoto důvodu provedeme filtrování daných instancí podle etnicity. Zvolené populace jsou ASW (Americans of African Ancestry in SW USA), CHB (Han Chinese in Bejing, China) a GBR (British in England and Scotland). Filtrování provede uzel *Nominal Value Row Filter*, ve kterém zvolíme filtrování podle sloupce Population, který byl k tabulce připojen z metadat, a který pak vybere výše zmíněné nominální hodnoty.

Před samotnou klasifikací je ještě nutné transformovat zápisy 0|0, n|0, 0|n a n|n na příslušné odpovídající číselné reprezentace (0, 1, …, n). Za tímto účelem je možné použít několik variant. Přepis můžeme převést pomocí uzlu *Category To Number*, který však vyžaduje velké množství paměti a jeho vykonávání trvá nepřiměřeně dlouho, ačkoliv nahrazování provádí odděleně pro jednotlivé sloupce. Dále můžeme použít iteraci přes sloupce a nahrazování hodnot pomocí uzlu *Cell Replacer*, to však také není optimální především kvůli nárokům na výpočetní čas. Vhodným alternativním řešením se jeví uložení mezivýsledků do čárkami odděleného souboru (pomocí uzlu *CSV Writer*) a následné nahrazení textu pomocí utility SED (stream editor) a opětovné načtení uzlem *CSV Reader*. Zvolené operace jsou uspořádány do workflow (viz obr. 16).



Obrázek 16: Proces prvotního načtení a transformace dat

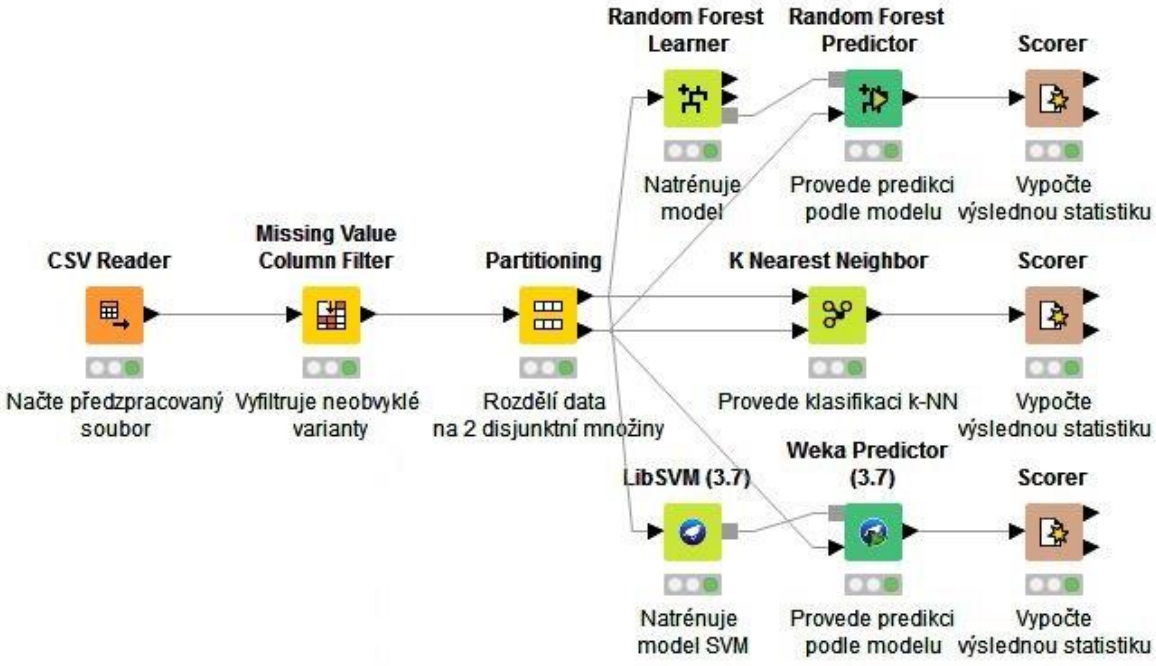
U znovu načtených dat je automaticky rozpoznán datový typ *Integer*. Neobvyklé varianty, které nebyly nahrazeny utilitou SED, nejsou načteny a je s nimi zacházeno, jako by se jednalo o chybějící hodnoty. Tyto chybějící hodnoty je nutné znovu vyfiltrovat. Protože jsou načtená data již transponována (atributy jsou nyní ve sloupcích a instance v řádcích), je nutné použít pro filtrování chybějících hodnot odlišný uzel, konkrétně *Missing Value Column Filter*, u kterého je potřeba nastavit limit chybějících hodnot ve sloupci na hodnotu blízkou nule.

7.1.3 Klasifikace dat

Ke klasifikaci dat jsem zvolil několik různých algoritmů. Jedním z nich je algoritmus k-nejbližších sousedů (k-NN), jehož popis je uveden ve čtvrté kapitole. Implementace tohoto algoritmu je obsažena v uzlu *K Nearest Neighbor*. Vstupními parametry jsou počet tříd a sloupec obsahující kategorie. V našem případě se jedná o atribut *Population*. Počet tříd je určen metodou vyhovění účelu (*downstream purpose*), neboť známe počet kategorií (ASW, CHB, GRB). Výstupem tohoto uzlu jsou vstupní data a nový atribut, který určuje skupinu, do které byla instance přiřazena. K měření vzdálenosti je používaná euklidovská metrika. Vstupem klasifikátoru je trénovací a testovací množina. Výstupem pak testovací množina doplněná o predikované třídy.

Dalším použitým klasifikátorem je model náhodného lesa. Jako parametry lze zvolit hloubku jednotlivých stromů, rozdělovací kritérium a počet stromů. Posledním zvoleným klasifikátorem je klasifikace pomocí SVM, implementovaná knihovnou LibSVM, a poskytnutá jako balíček aplikace Weka.

Výsledek klasifikace je evaluován pomocí uzlu *Scorer*. Hodnocení klasifikátoru je provedeno predikčním testováním externí validací (*hold-out* metoda) rozdělením dat do dvou disjunktních množin (trénovacích a testovacích dat). Pro získání přesnějších výsledků byla klasifikace pro každý počet instancí a každý klasifikátor opakována třikrát a výsledky byly zprůměrovány prostým aritmetickým průměrem. Workflow klasifikace a vyhodnocení úspěšnosti klasifikátorů je zobrazeno na následujícím obrázku.



Obrázek 17: Klasifikace dat a evaluace

V některých případech bývá možné atributy nesoucí jen málo informací nebo žádnou informaci vyfiltrovat. Tímto způsobem lze dosáhnout zrychlení procesu klasifikace a úspory paměti využívané během klasifikace. V tomto experimentu je filtrování atributů provedeno filtrováním podle rozptylu uzlem *Low Variance Filter*. Pokud jsou hodnoty jednoho atributu konstantní, pak je jeho rozptyl nulový a tento atribut je pro klasifikaci nepřínosný. Data byla klasifikována i na vyfiltrovaných množinách atributů stejným způsobem popsaným výše.

Pro každý klasifikátor je dále otestována hypotéza, zda se liší celková správnost zařazených instancí použitým klasifikátorem od celkové správnosti získané klasifikací náhodným klasifikátorem. Hypotézu testujeme jednovýběrovým binomickým testem.

7.1.4 Výsledky experimentu

Byla provedena klasifikace na datech, která před zpracováním obsahovala 30 tisíc, 100 tisíc a 300 tisíc atributů a 2504 instancí. Z experimentu byla následně vyřazena klasifikace pomocí SVM, neboť se ukázalo, že klasifikace podpůrnými vektory je podstatně časově náročnější než ostatní druhy klasifikace (viz tabulky 11 a 12), a přitom vykazuje srovnatelné výsledky (viz tabulky 9 a 10).

Vypočtená přesnost klasifikace na celé množině atributů je uvedena v tabulce 9 a přesnost klasifikace na vyfiltrované množině v tabulce 10.

	K-NN	Náhodný les	SVM
30 000 atributů	92,9 %	97,8 %	93,7 %
99 997 atributů	94,8 %	99,3 %	94,8 %
299 994 atributů	95,6 %	99,4 %	-

Tabulka 9: Přesnost klasifikace na celé množině atributů

S filtrováním	K-NN	Náhodný les	SVM
4 889 atributů	93,7 %	97 %	94 %
14 240 atributů	94,8 %	97,8 %	94,4 %
41 878 atributů	95,2 %	98,2 %	97,4 %

Tabulka 10: Přesnost klasifikace na vyfiltrované množině atributů

Pro otestování hypotézy shodnosti přesnosti použitého klasifikátoru od náhodné klasifikace určíme úspěšnost náhodného klasifikátoru. Třídy *GBR* a *CHB* jsou největší a stejně početné (36 instancí), testovací množina obsahuje celkem 90 instancí. Úspěšnost náhodné klasifikace je tedy určena jako:

$$P_R = \frac{N_i}{N} = \frac{36}{90} = 40\ \% \tag{7}$$

Testování hypotézy pro K-NN na 30 000 atributech:

$$z = \frac{P_A - P_R}{\sqrt{(P_R(1-P_R))/N}} = \frac{0{,}929 - 0{,}4}{\sqrt{(0{,}4(1-0{,}4))/90}} = 10{,}25 \quad (8)$$

Protože $z \gg 1{,}96$, zamítáme nulovou hypotézu o shodnosti klasifikace metodou K-NN (na množině 30 000 atributů) s náhodnou klasifikací. A protože je přesnost této klasifikace z výše uvedených nejmenší, můžeme zamítnout nulovou hypotézu o shodnosti přesnosti klasifikace s přesností náhodnou klasifikací i pro zbývající metody a množiny atributů. Výše uvedené přesnosti svědčí o správně zvolených klasifikačních metodách a správně určených parametrech klasifikace.

Dalším cílem tohoto experimentu je ověření, zda je možné a vhodné použít v tomto experimentu centralizované zpracování s *out of box* algoritmy nástroje KNIME. V následujících tabulkách je uvedena doba běhu jednotlivých klasifikačních algoritmů (natrénování modelu a určení tříd testovaných dat).

	K-NN	Náhodný les	SVM
30 000 atributů	7,70 s	11,61 s	453 s
99 997 atributů	10,58 s	39,84 s	5275,3 s
299 994 atributů	33,41 s	293,82 s	-

Tabulka 11: Doba trvání trénování klasifikátoru a klasifikace v sekundách

S filtrováním	Filtr	K-NN	Náhodný les	SVM
4 889 atributů	13,48 s	1,32 s	2,04 s	26,66 s
14 240 atributů	23,65 s	1,86 s	2,54 s	94,09 s
41 878 atributů	130,61 s	6,72 s	11,95 s	927,68 s

Tabulka 12: Doba trvání trénování klasifikátoru a klasifikace v sekundách na vyfiltrované množině atributů

Klasifikace byla provedena na notebooku s procesorem Intel Core i7-2630QM s celkovou pamětí 8 GB RAM, přičemž cca 5 GB bylo přiděleno aplikaci. Ačkoliv uvedené časy v sekundách nejsou příliš velké, klasifikace velkých dat naráží na několik problémů.

Prvním z problémů je zápis dat na disk. Pokud jsou data příliš velká, lze v KNIME zvolit u jednotlivých uzlů možnost zápis na disk. V takovém případě jsou data ukládána po zpracování na disk. Ale i v tomto případě si KNIME udržuje část dat v paměti, čímž umožňuje jejich rychlý náhled. V případě některých uzlů (konkrétně například některých klasifikačních modelů) je nutné uchovávat veškerá data v paměti RAM, což vede k chybám z nedostatku paměti, a tím pádem i k přerušení celého vykonávaného procesu. Nedostatek paměti se nejvýrazněji projeví při rozsáhlém workflow, z tohoto důvodu bylo také workflow rozděleno do více částí. Tento problém lze částečně řešit vertikálním škálováním.

Druhým problémem, který souvisí s prvním, je dimenzionalita dat. KNIME si obecně (a stejně jako obecně některé klasifikační algoritmy) dokáže špatně poradit s velkým počtem atributů dat. Atributy jsou reprezentovány jako sloupce a většina grafického rozhraní zobrazuje vždy seznam všech sloupců, což vede k zamrzávání grafického rozhraní aplikace a někdy také k jejím pádům. Tento problém lze odstranit například spuštěním workflow bez grafického rozhraní.

Dalším problémem je neefektivní implementace některých algoritmů. Například neefektivní spojování více datasetů uzlem Join u datasetů s různým počtem řádků. Nebo neefektivní transformace kategorií na číselné hodnoty.

7.2 Kombinace centralizovaného přístupu s distribuovaným zpracováním dat

V předchozím experimentu se ukázalo, že zpracováváním velkých biologických dat naráží na výpočetní kapacitu jednoho počítačového systému. Cílem tohoto experimentu je prozkoumat možnost kombinace centralizovaného zpracování *out of box* funkcemi, které KNIME nabízí s distribuovaným zpracováním pomocí Apache Sparku a konektorů, které jsou v jednom z rozšíření KNIME implementovány.

Aby bylo možné KNIME používat v kombinaci s Apache Sparkem, je nutné podniknout několik kroků. V první řadě je potřeba získat rozšíření a licenci k rozšíření *Spark Executor*. Dále je nutné spustit nakonfigurovanou komponentu Spark Job Server na stejném uzlu, na kterém je běžící instance Sparku. Následně je nutné v KNIME vytvoření kontextu Sparku. Ten je vytvořen po připojení ke Spark Job Serveru s odesláním konfiguračních dat (například počet uzlů clusteru, paměť uzlu). Experiment předpokládá využití instalace Apache Sparku a HDFS v Metacentru.

7.2.1 Načtení vstupních dat a jejich transformace

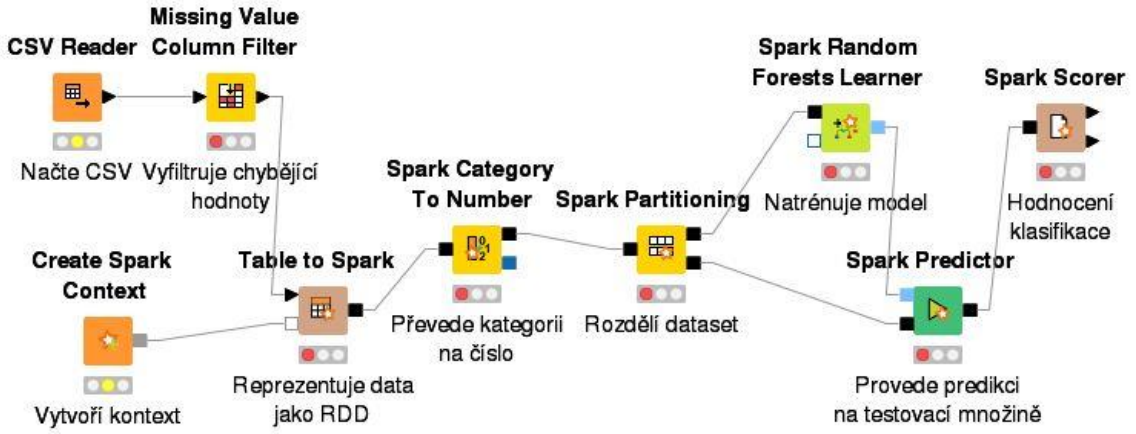
Pokud jsou data uložena na distribuovaném souborovém systému, lze je načítat prostřednictvím uzlu Spark Java Snippet (source), který obsahuje načtení vstupních dat (například prostřednictvím funkce *textFile("path")*). V případě, že data nejsou na distribuovaném systému uložena, lze je načíst lokálně přímo v KNIME například uzlem *File Reader* a následně tato data přenést do distribuovaného systému a transformovat je na RDD uzlem *Table To Spark.*

V našem případě využijeme již předzpracovaná data z předchozího experimentu, neboť KNIME nepodporuje například implementaci operace transpozice na Sparku a další operace předpokládají jako vstup již transponovaná data. V dalších krocích jsou ve workflow používány již jen reference na RDD objekty místo lokálně uložených dat.

7.2.2 Klasifikace dat

Klasifikace dat je provedena prostřednictvím Apache Sparku, konkrétně knihovny MLlib. KNIME v tomto případě slouží víceméně jen jako grafické rozhraní. Na druhou stranu při budování složitějších workflow je možné předzpracovaná data (resp. agregovaná data) přenést zpět do KNIME a provádět zbytek analýz nástroji implementovanými přímo v KNIME. To ovšem za předpokladu, že jsou agregovaná data již rozumně velká (jak již bylo uvedeno u předchozího experimentu).

Rozdělení RDD objektů na disjunktní množiny trénovacích a testovacích dat provede uzel Spark Partitioning. Trénovací data lze využít k natrénování některého z dostupných klasifikátorů, jejichž výčet je jednak mnohem menší než výčet klasifikátorů implementovaných klasickým způsobem v KNIME a jednak také menší než počet algoritmů implementovaných knihovnou MLlib. Vyhodnocení klasifikace provede následně uzel *Spark Scorer*, což je uzel analogický k uzlu *Scorer*. Ukázka možného workflow je zobrazena na obrázku 18.



Obrázek 18: Návrh klasifikace dat s použitím Apache Sparku v KNIME

7.2.3 Výsledky experimentu

Vypracování experimentu narazilo především na problém s nekompatibilní verzí dodávaného Spark Job Serveru a verzí Sparku používaného Metacentrem. Snaha překonat tento problém instalací odpovídající verze Spark Job Serveru umožnila vykonat pouze tu část workflow, která nepracovala s knihovnou MLlib, zde se dodávaná verze Spark Job Serveru přímo pro KNIME patrně liší od generické implementace.

Nicméně i tak lze z experimentu vyvodit některé závěry. Ukázalo se, že propojení KNIME se Sparkem má některé nedostatky. Například v případě referencování objektů typu RDD se část těchto objektů ukládá v operační paměti uzlu s instalací KNIME, což ve finále vede k problémům s pamětí. Dále se projevil také nedostatek zmíněný v předchozím experimentu, totiž velká dimenzionalita dat a problém s grafickým rozhraním.

KNIME a jeho Spark Executor navíc nepřidává v podstatě žádnou funkcionalitu navíc (pominu-li možnost kombinovat Spark uzly s uzly KNIME určenými pro centralizované zpracování) a slouží víceméně jen jako grafické rozhraní pro Spark. V případě používání složitějších transformací je navíc nutné napsat vlastní snippety.

7.3 Ryze distribuované zpracování

Předchozí experimenty ukázaly některé problémy, které je možné překonat distribuovaným zpracováním dat. Cílem tohoto experimentu je ukázat, že lze pro analýzu velkých biologických dat použít distribuovaný způsob s užitím Apache Sparku. To s sebou nese potřebu implementovat posloupnost transformací a volání příslušných funkcí knihovny MLlib v některém z programovacích jazyků Scala, Java anebo Python.

Při práci na implementaci algoritmu v programovacím jazyce Java pro Apache Spark bylo nalezeno již implementované řešení s názvem VariantSpark [80, 81]. VariantSpark je knihovna napsaná v jazyce Scala vydaná jako open source pod licencí organizace CISRO (Commonwealth Scientific and Industrial Research Organisation). Níže následuje popis implementace zpracování dat VariantSparkem.

7.3.1 Analýza VariantSparku

Předzpracování dat je ve VariantSparku implementováno v několika krocích. V prvním kroku jsou parsovány VCF soubory za účelem extrakce hlavičky souboru a dat. Extrakce hlavičky spočívá ve vytažení identifikátorů subjektů z dat (v tomto případě se jedná o názvy sloupců). Za tímto účelem je stěžejní použití transformace filtru (viz kód 7).

```
private def getHeadings(VcfFiles: RDD[String]) : List[String] = {
  VcfFiles
    .filter( _.startsWith("#CHROM") )
    .map(
      line => {
        val parser = new CSVParser('\t')
        parser.parseLine(line)
      } ).map(e => List(e:_*)).first()
}
```

Kód 7: Extrakce hlavičky sloubců z VCF souboru

V tomto kódu je použit sled transformací a jedna akce (viz kap. 5.2.2). V globální proměnné *VcfFiles* je RDD obsahující načtená data ze souboru funkcí *sc.textFile(path)*. Následně jsou definovány transformace. První z nich je *filter* s funkcí *startsWith*. Tato transformace vrátí jen ty řádky obsahující #CHROM na svém začátku. Z popisu souboru již víme, že se jedná o řádek obsahující hlavičky sloupců. Následně je aplikována transformace map, která použije objekt třídy *CSVParser* a metodu *parseLine*, která v této transformaci rozdělí řádek podle tabulátorů. Nad těmito rozdělenými hlavičkami je zavolána další transformace, který vrátí hlavičky sloupců jako list. Vše je vykonáno až zavoláním akce *first()*, který vrátí jen první list s rozparsovanými hlavičkami sloupců. Tyto operace probíhají paralelně na více uzlech clusteru s rozdělením souboru na řádky, což automaticky obstarává vnitřní implementace Sparku.

Dále je nutné rozparsovat samotná data a převést hodnoty reprezentující varianty alel na čísla (viz první experiment). Tyto operace jsou zapsány v následujícím kódu.

```
private val VcfLineRdd : RDD[(Long, Array[(String, Double)])] = {
   val VariantCutoff = this.VariantCutoff
   val NotAVariant = this.NotAVariant
   val Headings = getHeadings(VcfFilesRDD)
   VcfFilesRDD
   .filter( !_.startsWith("#") )
   .mapPartitions( lines => {
     val parser = new CSVParser('\t')
     lines.map( line => {
       parser.parseLine(line)
       .zip(Headings)
     } )
   } )
   .map( line => (
       "%s-%s".format(line(0)._1, line(1)._1 ),
       line.filter(v => !(NotAVariant contains v._2))
       .map(v => (v._2, variantDist(v._1, 0)))
       .filter(v => v._2 != 0)
   ))
   .filter(h => h._2.length > VariantCutoff) // Filter out 'rare' variants
   .zipWithIndex()
   .map(h => (h._2, h._1._2))
 }
```

Kód 8: Předzparcování dat VariantSparkem

Stejně jako v předchozím kroku se nejprve získají data z VCF souboru transformací *filter* s funkcí *startsWith*, která vrací pravdu, pokud řádek nezačíná znakem #. Následně jsou data rozparsována stejným způsobem zmíněným výše. Data se svážou s hlavičkou transformací *zip*. Na tato svázaná data (jako dvojice) je zavolána série transformací. Nejprve jsou odstraněny sloupce, které neobsahují data variant, ale obsahují metadata (viz popis VCF souboru). Poté je proveden samotný převod řetězců na číselné reprezentace samostatnou metodou *variantDist* a odstraněny vzácné varianty. Nulové varianty jsou odstraněny (obdoba užití *Low Variance Filteru* u prvního experimentu). Posledním krokem je vytvoření objektu *DataFrame*, který obsahuje vektory (varianty pro každou osobu), akcí *grupByKey* a spojení těchto vektorů s etnicitou jedince pomocí transformace *join*.

Natrénování modelu je po předzpracování dat záležitostí zavolání funkce knihovny MLlib, což je zobrazeno v následujícím kódu:

```
val model = KMeans.train(dataFrame.map(_._3), k, 300)
```

Kód 9: Vytvoření modelu knihovnou MLlib

Použité parametry jsou data, počet tříd a maximální počet iterací algoritmu. Samotná predikce třídy se pak provede metodou *predict* modelu.

7.3.2 Výsledky

VariantSpark byl podle citovaných zdrojů úspěšně použit na velkých datech [80, 81]. V uvedeném experimentu bylo srovnáváno shlukování pomocí k-průměrů s použitím různých implementací. Jmenovitě se jednalo například o distribuované zpracování použitím VariantSparku a algoritmu ADAM (oba využívající Apache Spark), pomocí Hadoopu a centralizovaně pomocí implementace v Pythonu a v R. Srovnání ukázalo, že se přesnost použitých implementací neliší. Byla stanovena na 84 %. Provedený experiment probíhal na chromozomu 22. V následující tabulce (tab. 13) jsou uvedeny naměřené časy předzpracování a shlukování k-průměry na jednom uzlu s využitím 32 GB RAM. V případě předzpracování bylo použito paralelní zpracování 8 vlákny (mimo implementace v Pythonu a R, kde bylo použito pouze jedno vlákno). Shlukování a predikce byly provedeny ve všech algoritmech s využitím 8 vláken [81].

Implementace	Předzpracování	Shlukování
VariantSpark (Spark)	2 minuty 58 sekund	1 minuta 20 sekund
ADAM (Spark)	12 minut 48 sekund	1 minuta 52 sekund
Hadoop	14 minut 22 sekund	14 minut 23 sekund
R	34 minut 30 sekund	7 minut 25 sekund
Python	34 minut 15 sekund	11 minut 29 sekund

Tabulka 13: Přehled doby nutné k předzpracování a shlukování variant genomů na 22. chromozomu použitím různých implementací [81]

Dále byla srovnávána možnost horizontálního škálování [80]. V tomto experimentu se ukázalo, že uvedená implementace s využitím Apache Sparku je horizontálně škálovatelná a je možné provádět analýzy i rozsáhlých datasetů. Například celý proces shlukování chromozomu 1 a 2 na 80 uzlech (každý s 3 GB RAM) trval pouhých 14 minut a 35 sekund [80].

8 Zhodnocení dosažených výsledků

V následující kapitole je uvedeno shrnutí a zhodnocení dosažených výsledků experimentů a zároveň je uvedeno optimální řešení reálného problému analýzy biologických dat, které vznikají na Katedře informatiky a výpočetní techniky Západočeské univerzity v Plzni.

8.1 Výsledky experimentů

V předcházejících experimentech bylo ukázáno, že ačkoliv probíhají práce na integraci nástrojů třetí generace do nástrojů generace první, není tato integrace stále ještě optimální a vede k problémům u mnohorozměrných dat.

Ačkoliv tato integrace umožňuje využívat jednoduchost, kterou nabízí první generace, je problematická, neboť jednak není v nástrojích první generace implementována veškerá funkce rychle se vyvíjejících nástrojů druhé, ale hlavně třetí generace, a pak také jednotlivé *mezigenerační* komponenty lze využívat jen obtížně. Pokud se ve workflow kombinují, pak se předpokládá, že výstupem integrovaných komponent (a zároveň vstupem komponent první generace) jsou již předzpracovaná data, případně jejich podmnožina, či jejich agregované hodnoty, které lze zpracovávat následně dle centrálního paradigmatu.

Navíc pokud jsou datasety příliš velké, že se nevejdou do operační paměti počítačového systému a je nutné provádět stránkování operační paměti na disk, pak se doba běhu celé analýzy ještě mnohonásobně prodlužuje a uvedené nástroje začínají selhávat. A protože je trend nárůstu dat exponenciální, nelze předpokládat, že by se tyto nástroje daly v budoucnu používat na rozsáhlé datasety.

Na druhou stranu lze provádět analýzy rozsáhlých dat nástroji třetí generace díky horizontálnímu škálování. Ovšem používání těchto nástrojů je obecně mnohem obtížnější než využívání nástrojů první generace s grafickým rozhraním a s možností zachycovat proces předzpracování a analýzy dat do přehledných workflow. Navíc v současné době je většina experimentů v biologii prováděna na malých podmnožinách dat a za tímto účelem je uzpůsobeno i vyhledávání v biologických databázích, které je obvykle vztažené k nějakému konkrétnímu problému (např. konkrétní typ rakoviny, konkrétní varianty genu, apod.). Data z těchto databází (někdy již předzpracovaná) je navíc potřeba stáhnout (obvykle prostřednictvím FTP nebo HTTP) a často také extrahovat z komprimovaných souborů. U často prováděných experimentů na malých množinách dat (nicméně složitých) lze použít s úspěchem i nástroje první generace.

Nicméně, jak již bylo zmíněno výše, trend nárůstu dat a nově navrhované experimenty ukazují na potřebu analýz stále větších datasetů a za tímto účelem se hodí nejlépe právě třetí generace zmíněných nástrojů.

8.2 Využití výsledků v EEG/ERP Portálu

S ohledem na dosažené výsledky a na růst dat, lze navrhnout i koncept pro zpracování dat na Katedře informatiky a výpočetní techniky ZČU pro projekt EEG/ERP Portál. Nicméně navržené řešení lze implementovat i do jiných projektů.

8.2.1 EEG/ERP Portál

Portál je webová aplikace, která umožňuje uchovávání, správu a analýzu elektrofyziologických experimentů. Samotná aplikace je implementována v Javě s využitím frameworků Spring a Hibernate. Data měření se ukládají do relační databáze Postgres, stejně jako některé administrativní údaje. Další metadata experimentů jsou uložena jako JSON objekty a zpřístupněna pomocí ElasticSearch. V současné době je velikost dat uložených v Portálu zhruba 40 GB s růstem cca 5 GB každý měsíc. Nicméně jednak se dá předpokládat, že růst velikosti dat bude časem větší. Jedním z důvodů je častější používání více kanálů při měření a dalším důvodem je také plánované rozšíření o data z dalších projektů a zapojení dalších výzkumných skupin do projektu. S rostoucími daty bude jednodušší provádět horizontální škálování zapojováním více počítačových systémů do clusteru než vertikálně škálovat jediný počítačový systém.

8.2.2 Ukládání naměřených dat

Ukládání naměřených dat do relační databáze PostgreSQL je z hlediska analýzy velkých dat neoptimální a problematické. Pokud by byla analýza velkých dat prováděna některými nástroji třetí generace, bylo by potřeba data z relační databáze přenášet a duplicitně ukládat do distribuovaného souborového systému. Za tímto účelem sice lze využít nástroje jako je Apache Sqoop [82], ale lepší variantou se jeví ukládání dat přímo do některého distribuovaného systému, ke kterému lze přistupovat jak z webové aplikace, tak na nich provádět analýzy velkých dat používáním nástrojů třetí generace.

Jako úložiště souborů s daty se jeví optimální HDFS s jedním NameNodem a jeho zálohou. Použití tohoto distribuovaného souborového systému umožní jednoduché ukládání souborů a v případě nedostatku kapacity úložiště lze velikost rozšířit přidáním dalšího zařízení do clusteru.

K tomuto úložišti lze přistupovat několika způsoby. Jednak příkazy z terminálu uzlu s Hadoop klientem nebo prostřednictvím REST API pomocí rozšíření WebHDFS a nebo také přímo z Java (Scala a Python) aplikací voláním příslušných metod. Stávající webová aplikace může místo ukládání souborů do PostgreSQL změnit příslušné metody a ukládat soubory přímo do HDFS. K tomu lze využít například třídu *FileSystem*. A provádět čtení a zápis prostřednictvím metod třídy *IOUtils* (viz kód níže).

```
// InputStream pro lokální soubor
InputStream in = new BufferedInputStream(new FileInputStream("file.txt"));

//Načte konfiguraci Hadoopu uloženou v xml souborech (např. core-default.xml)
Configuration cf = new Configuration();

//Cílové umístění souboru v HDFS
FileSystem fs = FileSystem.get(URI.create("hdfs://…"), cf);
OutputStream out = fs.create(new Path("hdfs://…"));

//Zkopíruje soubor po 4096 bajtech
IOUtils.copyBytes(in, out, 4096, true);
```

Kód 10: Zápis souboru do HDFS

8.2.3 Vyhledávání

Současná verze Portálu umožňuje vyhledávání pomocí SQL dotazů nad relační databází a také prostřednictvím nástroje ElasticSearch. Obojího lze využít i při analýze velkých dat. Apache Spark umožní jednak získat výsledky vyhledávání nad relační databází a jednak přistupovat k nástroji ElasticSearch pomocí rozšíření Elasticsearch for Hadoop.

Spark se může k relačním databázím připojit pomocí JDBC. Po připojení je možné vykonávat SQL dotazy. Po připojení k databázi se využívají objekty typu DataFrame, které odpovídají tabulce relační databáze. Jedná se o distribuovanou kolekci dat uspořádanou do pojmenovaných sloupců. DataFrame objekty lze převádět na objekty typu RDD [65]. Ukázka možného připojení je uvedena níže.

```
// z kontextu Sparku se vytvoří SQL kontext
SQLContext sqlContext = new org.apache.spark.sql.SQLContext(sc);
// nadefinují se parametry pro připojení
Map<String, String> options = new HashMap<String, String>();
options.put("url", "jdbc:postgresql:dbserver");
options.put("dbtable", "schema.tablename");
DataFrame jdbcDF = sqlContext.read().format("jdbc"). options(options).load();
```

Kód 11: Dotazování nad relační databází v Apache Sparku [65]

Pro připojení k ElasticSearch je nutné nakonfigurovat konektor ElasticSearch for Hadoop, který slouží jako prostředník mezi instancí ElasticSearch a mezi Apache Sparkem. Vyhledávání lze pak provádět přímo z kódu:

```
val sc = new SparkContext(conf)
sc.esRDD("radio/artists", "?q=me*")
```

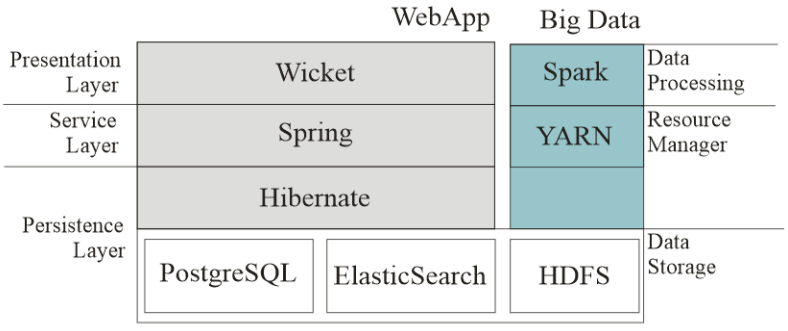
Kód 12: Použití ElasticSearch konektoru z Apache Sparku [83]

8.2.4 Vykonávání analýz

Analýzy lze vykonávat několika způsoby. Je možné zadávat příkazy prostřednictvím spark-shellu, což je užitečné pro jednoduché aplikace. Pro složitější analýzy je vhodnější použití aplikací zapsaných v Pythonu, Javě nebo Scale spouštěných přes příkaz spark-submit. Případně je možné spouštět tyto úlohy prostřednictvím REST API Spark Job Serveru nebo přímo z webové aplikace rozšířením Spring for Apache Hadoop [83].

Prvním krokem k provádění analýz je výběr vstupních dat. Za tímto účelem lze použít vyhledávání pomocí ElasticSearch a následně použít SQL dotaz k vrácení umístění souborů s daty umístěných na HDFS. Analýzy lze provádět také nad celou množinou dat, nebo lze data vybírat ručně zadáním příslušných adresářů či jednotlivých souborů. Spojení metadat s daty je možné provést transformací *join*, neboť jsou metadata stejně jako data reprezentována RDD objekty (případně jako objekty DataFrame).

Samotné analytické algoritmy bude potřeba vložit do kontextu celého skriptu vykonávaného Apache Sparkem. Některé bude možné využít přímo (například transformace používané na jednotlivé soubory), některé bude potřeba upravit (budování modelů). V oblasti výzkumu EEG existuje již několik implementací, které by bylo možné použít nebo se jimi inspirovat při úpravách stávajících algoritmů [85, 86]. Výsledky provedených algoritmů lze zapisovat jako soubory na HDFS a případně vizualizovat na Portálu.



Obrázek 19: Znázornění architektury Portálu s nově přidanými komponentami pro analýzu velkých dat

8.2.5 Bezpečnost řešení

Integrace analýzy velkých dat do Portálu s sebou přináší nutnost řešit bezpečnost nově přidaných částí. V původní aplikaci jsou oprávnění k jednotlivým souborům řešena pomocí uživatelských rolí, které jsou uloženy v relační databázi. Role jsou vztaženy mimo jiné ke konkrétním experimentům. Uživatel přiřazený ke konkrétnímu experimentu disponuje pak právy ke čtení příslušných souborů, které jsou k tomuto experimentu vztaženy.

V nových modulech je potřeba řešit bezpečnost odlišným způsobem. Je tak potřeba řešit bezpečnost na úrovních:

- HDFS – spravovat práva přístupu k jednotlivým souborům
- YARN – spravovat práva k prostředkům, které může spuštěná aplikace využívat, řešit přístup k administrativním funkcím
- ElasticSearch – spravovat práva vyhledávání a přístupu k metadatům
- Apache Spark – spravovat práva pro typy spouštěných aplikací, řešit oprávnění pro zobrazení stavu úloh apod.
- Síť – řešit zabezpečení přenosu souborů v rámci sítě

Autentizaci uživatelů při přístupu k HDFS a zadávání úloh pomocí Apache Sparku je možné provádět několika způsoby jako například pomocí Kerberos lístků nebo pomocí hesla. Práva k jednotlivým souborům je pak možné řešit pomocí ACL (access control list). Bude také potřeba vymyslet, jakým způsobem kontrolovat oprávnění při přístupu k metadatům pomocí ElasticSearch konektoru přímo v aplikaci napsané pro Apache Spark. Jedním z perspektivních řešení může být řešení bezpečnosti pomocí nástroje Apache Ranger. V současné době je tento nástroj ve vývoji a tak ze shora uvedených úrovní není řešeno použití ElasticSearch. Nicméně podpora by měla být přidána v některé z příštích verzí, stejně tak podpora uživatelských rolí [88].

9 Závěr

Cílem této diplomové práce byl především rozbor problematiky velkých dat v biologii. Vymezení termínu velkých dat jsem popsal ve druhé kapitole. Zde jsem ukázal na nejednoznačnost, která se v této terminologii vyskytuje. Ve třetí kapitole jsem analyzoval důležité biologické databáze, které obsahují velká data, a ukázal, v jakých formátech se v nich data ukládají, jestli se jedná o obecně přijímané standardy a jakým způsobem je možné z nich data získat a dále s nimi pracovat.

Další cílem této diplomové práce bylo prozkoumání metod analýzy velkých dat. Analytické metody, které se běžně používají k dolování dat, jsem popsal ve čtvrté kapitole. V této kapitole jsem také zasadil proces analýzy do širšího rámce získávání znalostí z dat a ukázal jeho jednotlivé kroky.

Dále jsem se v práci zabýval, jakými způsoby lze metody popsané v předcházející kapitole aplikovat na velká data z databází zmíněných v kapitole třetí. Došel jsem k závěru, že v podstatě existují dva základní přístupy zpracování velkých dat. Jedním z nich je centralizovaný způsob a druhý distribuovaný. Tyto přístupy jsem rozebral a u distribuovaného přístupu ukázal některé standardně používané postupy.

Cílem této diplomové práce bylo také poskytnout a srovnat nástroje, které se používají k analýze velkých dat. Tyto nástroje jsem detailně popsal v šesté kapitole. V této části jsou popsané nástroje rozděleny do jednotlivých generací a v každé generaci jsou uvedeny významné nástroje. Tyto nástroje jsou mezi s sebou srovnány a na základě jejich srovnání byl vybrán vhodný nástroj KNIME.

Tato práce se také zabývá experimentálním zpracováním velkých dat. K experimentům byla vybrána databáze variant lidského genomu. Ve dvou navržených experimentech jsem zkoumal vhodnost využití analytických metod, které tato práce popisuje ve čtvrté kapitole, ale také vhodnost užití jednotlivých paradigmat. Experimenty jsem zjistil, že centralizované zpracování velkých dat není pro tak rozsáhlé zpracování vhodné a také že KNIME lze pro distribuované zpracování velkých dat v kombinaci s Apache Sparkem použít jen obtížně.

Zaměřil jsem se tedy v této práci na ryze distribuované zpracování velkých dat Apache Sparkem a během implementace řešení, použitelného v této doméně jsem objevil existující implementaci, kterou jsem blíže prozkoumal. Ukázalo se, že Apache Spark je vhodný kandidát pro zpracovávání velkých dat.

Závěry získané z experimentů a ze zpracování této práce jsem formuloval v závěru práce do návrhu řešení zpracování velkých elektrofyziologických dat na Katedře informatiky a výpočetní techniky Západočeské univerzity v Plzni. Tento návrh byl zvolen na základě současných trendů v analýze a zpracování velkých dat a také s ohledem na použité perspektivní dynamicky se rozvíjející open source technologie.

Seznam zkratek

ACL	Access Control List
AGPL	Affero General Public License
ANN	Artificial neural network
API	Application Programming Interface
ARFF	Attribute-Relation File Format
BAM	Binary Alignment/Map
BLAST	Basic Local Alignment Search Tool
BSP	Bulk synchronous parallel
BT	Binary Tree
CART	Classification and regression tree
CISRO	Commonwealth Scientific and Industrial Research Organisation
CLI	Command Line Interface
CSV	Comma-separated values
DAG	Directed acyclic graph
DBSCAN	Density-based spatial clustering of applications with noise
DCT	Discrete cosine transform
DICOM	Digital Imaging and Communications in Medicine
DNA	Deoxyribonucleic acid
EDF	European Data Format
EEG	Electroencephalography
EKG	Elektrokardiografie
EMBL	The European Molecular Biology Laboratory
ERP	Event-related potential
FASTA	Fast Alignment Search Tool Format
FASTQ	Fast Alignment Search Tool with Quality Format
FTP	File Transfer Protocol
GB	Gigabyte
GMM	Gaussian mixture models
GNU GPL	GNU General Public License
GO	Gene Ontology
GPR	GenePix Results Format
GPU	Graphic processing unit
GSP	Generalized Sequential Patterns
GUI	Graphical User Interface
HDFS	Hadoop Distributed File System
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
ICA	Independent component analysis
ID3	Iterative Dichotomiser 3
IG Format	Institute of Genetics Format

JDBC	Java Database Connectivity
JSON	JavaScript Object Notation
JSONP	JSON with Padding
KDD	Knowledge discovery in databases
K-NN	K-Nearest Neighbors
LDA	Latent Dirichlet Allocation
LWL	Locally weighted learning
MDS	Multi Dimensional Scaling
MIAME	Minimum Information About a Microarray Experiment
MINSEQE	Minimum Information about a high-throughput Nucleotide SeQuencing Experiment
mmCIF	Macromolecular Crystallographic Information File
NBRF Format	National Biomedical Research Foundation
NCBI	National Center for Biotechnology Information
PCA	Principal Component Analysis
PDB	Protein Data Bank
PDBML	Protein Data Bank Markup Language
PDF	Portable Document Format
PIC	Power iteration clustering
PMML	Predictive Model Markup Language
PNG	Portable Network Graphics
RAM	Random Access Memory
RDD	Resilient Distributed Dataset
RDF	Resource Description Framework
REST	Representational State Transfer
RF	Random Forest
RNA	Ribonucleic Acid
SAM	Sequence Alignment/Map
SED	Stream Editor
SPARQL	SPARQL Protocol and RDF Query Language
SQL	Structured Query Language
SRS	Sequence Retrieval System
SVD	Singular Value Decomposition
SVG	Scalable Vector Graphics
SVM	Support Vector Machine
UCSC	University of California Santa Cruz
VCF	Variant Call Format
XLS	Microsoft Excel Spreadsheet
XML	Extensible Markup Language
XRFF	Extensible Attribute-Relation File Format
YARN	Yet Another Resource Negotiator

Seznam použité literatury a zdrojů informací

[1] GANDOMI, Amir a Murtaza HAIDER. Beyond the hype: Big data concepts, methods, and analytics. International Journal of Information Management [online]. 2015, 35(2), 137-144 [cit. 2016-04-10]. DOI: 10.1016/j.ijinfomgt.2014.10.007. ISSN 02684012. Dostupné z: http://linkinghub.elsevier.com/retrieve/pii/S0268401214001066

[2] VORHIES, Bill. How Many "V"s in Big Data: he Characteristics that Define Big Data. In: Data-Magnum [online]. 2013 [cit. 2016-04-10]. Dostupné z: http://data-magnum.com/how-many-vs-in-big-data-the-characteristics-that-define-big-data/

[3] ANDERSON, David R., Denis J. SWEENEJ, Thomas A. WILLIAMS, Jeffrey D. CAMM a James J. COCHRAN. Statistics for Business & Economics. 13th ed. Boston, USA: Cengage Learning, 2016. ISBN 9781305856790.

[4] CUKIER K., The Economist, Data, data everywhere: A special report on managing information [online] 2010 [cit. 2016-04-10 Dostupné z: http://www.economist.com/node/15557443

[5] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. Big data: Preliminary Report 2014. In: ISO/IEC JTC1 [online]. 2015 [cit. 2016-04-10] Dostupné z: http://www.iso.org/iso/big_data_report-jtc1.pdf

[6] RAZA, Khalid. Application of Data Mining in Bioinformatics. Indian journal of computer science and engineering. [ONLINE] Chennai, India: Engg Journals Publications, 2010. ISSN 2231-3850. Dostupné z: http://www.ijcse.com/ijcse-issue.html?issue=20100102

[7] Congressional Justification FY 2015. U.S. National Library of Medicine [online]. Bethesda, 2015 [cit. 2016-04-16]. Dostupné z: https://www.nlm.nih.gov/about/2015CJ.html

[8] GIBAS, Cynthia a Per JAMBECK. Developing bioinformatics computer skills. 1st ed. Cambridge: O'Reilly, 2001. ISBN 15-659-2664-1. [cit. 2016-04-16] Dostupné z: http://www.bio-nica.info/biblioteca/Gibas2001DevelopingBioinformatics.pdf

[9] LIÉBECQ, Claude. Biochemical nomenclature and related documents: a compendium. 2nd ed. Chapel Hill: Portland Press, 1992. ISBN 1855780054.

Zdroj: http://www.chem.qmul.ac.uk/iubmb/misc/naseq.html

[10] GENOMIX SOFTWARE. DNA Sequence formats [online]. 2015 [cit. 2016-04-10] Dostupné z: https://www.genomatix.de/online_help/help/sequence_formats.html

[11] MOUNT, David W. Bioinformatics: sequence and genome analysis. Cold Spring Harbor, N.Y.: Cold Spring Harbor Laboratory Press, c2001. ISBN 08-796-9608-7.

[12] BUFFALO, Vince. Bioinformatics data skills. First edition. Sebastopol, CA: O'Reilly, 2015. ISBN 14-493-6737-2.

[13] LEWITTER, Fran a George BELL. Relational Databases 101. In: GenomeWeb [online]. [cit. 2016-04-16]. Dostupné z: https://www.genomeweb.com/relational-databases-101

[14] ETZOLD, Thure; ARGOS, Patrick. SRS—an indexing and retrieval tool for flat file data libraries. Computer applications in the biosciences: CABIOS, 1993, 9.1: 49-57. [cit. 2016-04-16]. Dostupné z: http://bioinformatics.oxfordjournals.org/content/9/1/49.short

[15] GAT, Irit a Tal KOHEN. Algorithms for Molecular Biology: Sequence Retrieval System [online]. In: . 1999 [cit. 2016-04-17]. Dostupné z: http://www.cs.tau.ac.il/~rshamir/algmb/98/scribe/html/lec04/node6.html

[16] LESK, Arthur M. Introduction to bioinformatics. New York: Oxford University Press, 2002. ISBN 01-992-5196-7.

[17] ZDOBNOV, E. M., R. LOPEZ, R. APWEILER a T. ETZOLD. The EBI SRS server--recent developments. Bioinformatics. 2002, 18(2), 368-373. DOI: 10.1093/bioinformatics/18.2.368. ISSN 1367-4803. Dostupné také z: http://bioinformatics.oxfordjournals.org/cgi/doi/10.1093/bioinformatics/18.2.368

[18] NATIONAL CENTER FOR BIOTECHNOLOGY INFORMATION. Enterez Help. 2005-2014 [cit. 2016-04-17] Dostupné z: http://www.ncbi.nlm.nih.gov/books/NBK3836/

[19] Ensembl: Ensembl REST API Endpoints [online]. 2016 [cit. 2016-04-17]. Dostupné z: http://rest.ensembl.org/

[20] UniProt: SPARQL Query [online]. 2016 [cit. 2016-04-17]. Dostupné z: http://sparql.uniprot.org/

[21] UNIVERSITY OF HELSINKI, Department of Computer Science. Introduction to bioinformatics: Chapter 7: Rapid alignment methods: FASTA and BLAST [online]. 2007 [cit. 2016-04-17]. Dostupné z: https://www.cs.helsinki.fi/bioinformatiikka/mbi/courses/07-08/itb/slides/itb0708_slides_117-142.pdf

[22] MRÁZ, František. An Introduction to Bioinformatics Algorithms: An Introduction to Bioinformatics Algorithms BLAST and FASTA Heuristics in pairwise sequence alignment [online]. 2014 [cit. 2016-04-17]. Dostupné z: http://ksvi.mff.cuni.cz/~mraz/bioinf/BioAlg10-8.pdf

[23] U. S. NATIONAL LIBRARY OF MEDICINE. Genetics Home Reference: What is a genome? [online]. 2016 [cit. 2016-04-17]. Dostupné z: https://ghr.nlm.nih.gov/primer/hgp/genome

[24] The Variant Call Format Specification: VCFv4.3 and BCFv2.2 [online]. 2015 [cit. 2016-04-17]. Dostupné z: http://samtools.github.io/hts-specs/VCFv4.3.pdf

[25] EUROPEAN BIOINFORMATICS INSTITUTE. 1000 Genomes: The 1000 Genomes Browser [online]. 2015 [cit. 2016-04-17]. Dostupné z: http://browser.1000genomes.org/index.html

[26] WELLCOME TRUST SANGER INSTITUTE. Mouse Genomes Project: Query SNPs, indels or SVs [online]. 2016 [cit. 2016-04-17]. Dostupné z: http://www.sanger.ac.uk/sanger/Mouse_SnpViewer/rel-1505

[27] GENE ONTOLOGY CONSORTIUM. Gene Ontology Documentation [online]. 2015 [cit. 2016-04-17]. Dostupné z: http://geneontology.org/page/ontology-documentation

[28] KOMENDA, Martin. Analýza sekvencí DNA [online]. 2015 [cit. 2016-04-17]. Dostupné z: http://is.muni.cz/www/98951/41610771/43823411/43823458/Analyza_genomick/55450383/Sekvence150227.pdf

[29] COIMBATORE NARAYANAN, Buvaneswari, John WESTBROOK, Saheli GHOSH, Anton I. PETROV, Blake SWEENEY, Craig L. ZIRBEL, Neocles B. LEONTIS a Helen M. The Nucleic Acid Database: new features and capabilities [online]. 2013, 42(D1), D114-D122 [cit. 2016-04-17]. DOI: 10.1093/nar/gkt980. Dostupné z: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3964972/

[30] BERMAN, H. M. The Protein Data Bank. Nucleic Acids Research [online]. 28(1), 235-242 [cit. 2016-04-17]. DOI: 10.1093/nar/28.1.235. ISSN 13624962. Dostupné z: http://nar.oxfordjournals.org/lookup/doi/10.1093/nar/28.1.235

[31] RESEARCH COLLABORATORY FOR STRUCTURAL BIOINFORMATICS. How to launch the Download Tool using Java Web Start [online]. 2016 [cit. 2016-04-17]. Dostupné z: http://www.rcsb.org/pdb/static.do?p=help/viewers/downloadHelp.html

[32] RESEARCH COLLABORATORY FOR STRUCTURAL BIOINFORMATICS. The RCSB PDB RESTful Web Service interface [online]. 2016 [cit. 2016-04-17]. Dostupné z: http://www.rcsb.org/pdb/software/rest.do

[33] WORLDWIDE PROTEIN DATA BANK. Protein Data Bank Contents Guide: Atomic Coordinate Entry Format Description [online]. 2008 [cit. 2016-04-17]. Dostupné z: ftp://ftp.wwpdb.org/pub/pdb/doc/format_descriptions/Format_v33_A4.pdf

[34] WORLDWIDE PROTEIN DATA BANK. PDBx/mmCIF Dictionary Resources [online]. 2016 [cit. 2016-04-17]. Dostupné z: http://mmcif.wwpdb.org/pdbx-mmcif-home-page.html

[35] JANEČKOVÁ, Eva a Martina VAŠÍČKOVÁ. Genová exprese, aneb, Co kódují geny. V Tribunu EU vydání první. Brno: Tribun EU, 2015. ISBN 978-80-263-0905-5. Dostupné také z: http://www.mendel-brno.cz/index.php/component/html5flippingbook/publication/genova-exprese-aneb-co-koduji-geny/7#page/55

[36] JAGANNATHAN, Vidhya. Databases for Microarrays [online]. 2002 [cit. 2016-04-17]. Dostupné z: http://www.ch.embnet.org/CoursEMBnet/CHIP02/ppt/Vidhya.ppt

[37] EUROPEAN BIOINFORMATICS INSTITUTE. ArrayExpress: Functional Genomics Data [online]. 2013 [cit. 2016-04-17]. Dostupné z: https://www.ebi.ac.uk/arrayexpress/

[38] BRAZMA, Alvis, Pascal HINGAMP, John QUACKENBUSH, et al. Nature Genetics. 29(4), 365-371. DOI: 10.1038/ng1201-365. ISSN 10614036. Dostupné také z: http://www.nature.com/doifinder/10.1038/ng1201-365

[39] HUSKA, Matt a Alena van BÖMMEL. Using R/Bioconductor for Microarray Analysis [online]. 2013 [cit. 2016-04-17]. Dostupné z: http://lectures.molgen.mpg.de/swp13/pres.pdf

[40] SAN ANTONIO CENTER FOR MEDICAL MYCOLOGY. About us [online]. [cit. 2016-04-17]. Dostupné z: http://www.sacmm.org/faculty/kadosh_research.html

[41] THE ASSOCIATION OF ELECTRICAL EQUIPMENT AND MEDICAL IMAGING MANUFACTURERS. The DICOM Standard [online]. 2016 [cit. 2016-04-17]. Dostupné z: http://dicom.nema.org/standard.html

[42] THE CANCER IMAGING ARCHIVE. TCIA Programmatic Interface (REST API) Usage Guide [online]. 2014 [cit. 2016-04-17]. Dostupné z: https://wiki.cancerimagingarchive.net/display/Public/TCIA+Programmatic+Interface+%28REST+API%29+Usage+Guide

[43] PHYSIONET. Frequently Asked Questions about PhysioNet [online]. 2016 [cit. 2016-04-17]. Dostupné z: https://www.physionet.org/faq.shtml

[44] PHYSIONET. How to obtain PhysioBank data in text form [online]. 2016 [cit. 2016-04-17]. Dostupné z: https://physionet.org/tutorials/physiobank-text.shtml

[45] KEMP, Bob. European Data Format [online]. [cit. 2016-05-07]. Dostupné z: http://www.edfplus.info/

[46] FAYYAD, Usama, Gregory PIATETSKY-SHAPIRO a Padhraic SMYTH. From Data Mining to Knowledge Discovery in Databases. AI Magazine. 1996, 17(3). Dostupné také z: https://www.aaai.org/ojs/index.php/aimagazine/article/viewFile/1230/1131

[47] KELBEL, Jan a David ŠILHÁN. Shluková analýza. 2004. Dostupné také z: https://www.fd.cvut.cz/personal/nagyivan/Projekty/Classification/ShlukovaAnalyza.pdf

[48] HOLČÍK, Jiří, KOMENDA, Martin (eds.) a kol. Matematická biologie: e-learningová učebnice [online]. 1. vydání. Brno: Masarykova univerzita, 2015. ISBN 978-80-210-8095-9. Dostupné z: http://portal.matematickabiologie.cz/

[49] ZAKI, Mohammed J a Wagner MEIRA. Data mining and analysis: fundamental concepts and algorithms. New York, NY: Cambridge University Press, 2014. ISBN 9780521766333. Dostupné z: http://www.cs.rpi.edu/~zaki/PaperDir/DMABOOK.pdf

[50] EKŠTEIN, Kamil. Teorie kognitivních systémů, přednášky: Support Vector Machines. 2012.

[51] VOMLELOVÁ, Marta. Rozšířené „rozhodovací stromy“[online]. 2013 [cit. 2016-04-24]. Dostupné z: http://kti.mff.cuni.cz/~marta/su11.pdf

[52] GÁBIK, Jozef. Kreditný skóring pomocou náhodných lesov. Bratislava, 2010. Dostupné také z: http://www.iam.fmph.uniba.sk/studium/efm/diplomovky/2010/gabik/diplomovka.pdf. Diplomová. Univerzita Komenského v Bratislavě.

[53] KRISHNAN, Krish. Data warehousing in the age of big data. Amsterdam: Morgan Kaufmann is an imprint of Elsevier, 2013. ISBN 978-012-4058-910.

[54] SAINI, H. S., Rishi SAYAL a Sandeep Singh RAWAT. Innovations in Computer Science and Engineering: Proceedings of the Third ICICSE, 2015. Ilustrované vydání. Springer, 2016. ISBN 9789811004193

[55] MAHMOOD, Syed F. a Huzefa RANGWALA. GPU-Euler: Sequence Assembly using GPGPU. 2011. Dostupné také z: https://cs.gmu.edu/~tr-admin/papers/GMU-CS-TR-2011-1.pdf

[56] LIAO, Gang, Qi SUN, Longfei MA, Sha DING a Wen XIE. SICHUAN UNIVERSITY.Ultra-fast Multiple Genome Sequence Matching Using GPU. Chengdu, 2015. Dostupné také z: http://arxiv.org/pdf/1303.3692.pdf

[57] TUNYS, Tomáš. MapReduce Introduction [online]. 2016 [cit. 2016-04-24]. Dostupné z: https://docs.google.com/presentation/d/1UTU67JZ3kALOn8wiKssuFkFpVk1o_ovEwDz9EKjRiRU

[58] WHITE, Tom. Hadoop: the definitive guide. 3rd ed. Sebastopol: O'Reilly, 2012. ISBN 978-1-4493-1152-0.

[59] PREE, Wolfgang. MapReduce: Simplified Data Processing on Large Clusters[online]. 2004 [cit. 2016-04-24]. Dostupné z: http://www.uni-salzburg.at/fileadmin/multimedia/SRC/docs/teaching/SS12/VS/MapReduce_1.pdf

[60] MACLEAN, Diana. A Very Brief Introduction to MapReduce [online]. 2011 [cit. 2016-04-24]. Dostupné z: http://hci.stanford.edu/courses/cs448g/a2/files/map_reduce_tutorial.pdf

[61] DEAN, Jeffrey a Sanjay GHEMAWAT. GOOGLE, INC. MapReduce: Simplified Data Processing on Large Clusters [online]. 2004 [cit. 2016-04-24]. Dostupné z: http://static.googleusercontent.com/media/research.google.com/cs//archive/mapreduce-osdi04.pdf

[62] CLOUDERA. Introduction to YARN and MapReduce 2 [online]. 2013 [cit. 2016-04-24]. Dostupné z: http://www.cloudera.com/resources/recordedwebinar/introduction-to-yarn-and-mapreduce-2-slides.html

[63] FERNÁNDEZ, Alberto, Sara DEL RÍO, Victoria LÓPEZ, Abdullah BAWAKID, María J. DEL JESUS, José M. BENÍTEZ a Francisco HERRERA. Big Data with Cloud Computing: an insight on the computing environment, MapReduce, and programming frameworks. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery. 2014, 4(5), 380-409. DOI: 10.1002/widm.1134. ISSN 19424787. Dostupné také z: http://doi.wiley.com/10.1002/widm.1134

[64] ZAHARIA, Matei, Mosharaf CHOWDHURY, Tathagata DAS, et al.Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. UNIVERSITY OF CALIFORNIA. Berkeley, 2012, , -. Dostupné také z: http://www.cs.berkeley.edu/~matei/papers/2012/nsdi_spark.pdf

[65] THE APACHE SOFTWARE FOUNDATION. Spark Programming Guide [online]. 2016 [cit. 2016-04-24]. Dostupné z: http://spark.apache.org/docs/latest/programming-guide.html

[66] UNIVERSITY OF LJUBLJANA. Orange [online]. 2016 [cit. 2016-04-24]. Dostupné z: http://orange.biolab.si/

[67] UNIVERSITY OF WAIKATO. Weka 3: Data Mining Software in Java [online]. [cit. 2016-04-24]. Dostupné z: http://www.cs.waikato.ac.nz/ml/weka/

[68] RAPIDMINER. RapidMiner [online]. 2016 [cit. 2016-04-24]. Dostupné z: https://rapidminer.com/

[69] KNIME. KNIME: Open for Innovation [online]. 2016 [cit. 2016-04-24]. Dostupné z: https://www.knime.org

[70] JOVIC, A., K. BRKIC a N. BOGUNOVIC. An overview of free software tools for general data mining. 2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO) [online]. IEEE, 2014, : 1112-1117 [cit. 2015-11-01]. DOI: 10.1109/MIPRO.2014.6859735. ISBN 978-953-233-077-9. Dostupné z: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6859735

[71] THE APACHE SOFTWARE FOUNDATION. Apache Mahout: Scalable machine learning and data mining[online]. 2016 [cit. 2016-04-24]. Dostupné z: http://mahout.apache.org

[72] DRIVEN. Cascading [online]. 2016 [cit. 2016-04-24]. Dostupné z: http://www.cascading.org

[73] CONCURRENT. Cascading Pattern: License [online]. 2015 [cit. 2016-04-24]. Dostupné z: https://github.com/Cascading/pattern/blob/wip-1.0/LICENSE.txt

[74] DRIVEN. Pattern [online]. 2016 [cit. 2016-04-24]. Dostupné z: http://www.cascading.org/projects/pattern/

[75] THE APACHE SOFTWARE FOUNDATION. Spark Documentation [online]. 2016 [cit. 2016-04-24]. Dostupné z: http://spark.apache.org/docs/latest/ml-guide.html

[76] MORITZ, Philipp, Robert NISHIHARA, Ion STOICA a Michael JORDAN. UNIVERSITY OF CALIFORNIA. SparkNet: Training Deep Networks in Spark [online]. 2016 [cit. 2016-04-24]. Dostupné z: http://www.cs.berkeley.edu/~pcmoritz/iclr2016_submission.pdf

[77] LAWRENCE, Michael. UNIVERSITY OF CALIFORNIA. Introduction to Variant Calling [online]. 2014 [cit. 2016-04-24]. Dostupné z: https://www.bioconductor.org/help/course-materials/2014/CSAMA2014/3_Wednesday/lectures/VariantCallingLecture.pdf

[78] CAVALLI-SFORZA, L. Luca a Marcus W. FELDMAN. The application of molecular genetic approaches to the study of human evolution. Nature Genetics.33(3s), 266-275. DOI: 10.1038/ng1113. ISSN 10614036. Dostupné také z: http://www.nature.com/doifinder/10.1038/ng1113

[79] THE INTERNATIONAL GENOME SAMPLE RESOURCE. 1000 Genomes Project [online]. 2016 [cit. 2016-04-24]. Dostupné z: http://www.1000genomes.org/data

[80] O'BRIEN, Aidan a Denis BAUER. CSIRO. VariantSpark: Applying Spark-based machine learning methods to genomic information [online]. 2015 [cit. 2016-04-24]. Dostupné z: http://ceur-ws.org/Vol-1468/bd2015_obrien.pdf

[81] O'BRIEN, Aidan, Neil SAUNDERS, Yi GUO, Fabian BUSKE, Rodney SCOTT a Denis BAUER. VariantSpark: Population scale clustering of genotype information. BMC Genomics. 2015, 16. DOI: 10.1186/s12864-015-2269-7.

[82] THE APACHE SOFTWARE FOUNDATION. Apache Sqoop [online]. 2016 [cit. 2016-04-30]. Dostupné z: http://sqoop.apache.org/

[83] ELASTICSEARCH. Elasticsearch for Apache Hadoop: Apache Spark support[online]. 2016 [cit. 2016-04-30]. Dostupné z: https://www.elastic.co/guide/en/elasticsearch/hadoop/current/spark.html

[84] LEAU, Costin, Thomas RISBERG a Janne VALKEALAHTI. PIVOTAL SOFTWARE. Spring for Apache Hadoop: Reference Documentation [online]. 2016 [cit. 2016-04-30]. Dostupné z: http://docs.spring.io/spring-hadoop/docs/current/reference/html/springandhadoop-spark.html

[85] BURNS, Matthew. Large Scale Electroencephalography Processing With Hadoop [online]. 2013 [cit. 2016-04-30]. Dostupné z: http://seed.ucsd.edu/mediawiki/images/3/32/Large_Scale_EEG_Processing_With_Hadoop.pdf

[86] BERRADA, Ghita, Maurice KEULEN a Mena HABIB. UNIVERSITY OF TWENTE. Hadoop for EEG Storage and Processing: a Feasibility Study [online]. 2014 [cit. 2016-04-30]. Dostupné z: http://doc.utwente.nl/91923/1/B237.pdf

[87] THE APACHE SOFTWARE FOUNDATION. Apache Hama [online]. 2016 [cit. 2016-05-07]. Dostupné z: https://hama.apache.org/

[88] THE APACHE SOFTWARE FOUNDATION. Apache Ranger [online]. 2016 [cit. 2016-05-08]. Dostupné z: http://ranger.apache.org/

[89] THE APACHE SOFTWARE FOUNDATION. HDFS Architecture [online]. 2016 [cit. 2016-05-08]. Dostupné z: http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html

Seznam obrázků

Seznam tabulek

PŘÍLOHY

A. Obsah DVD

Přiložené DVD má následující strukturu:

- Workspaces
    - GenomesPreprocessing – obsahuje předzpracování dat aplikací KNIME
    - GenomesClassification – obsahuje klasifikaci dat aplikací KNIME
    - GenomesSpark – obsahuje workflow pro KNIME a Apache Spark
- Statistics.xlsx – obsahuje výsledky experimentů pro jednotlivá opakování
- KASAL_A14N0076P_DP.pdf
- KASAL_A14N0076P_DP.docx