MATEMATICKO-FYZIKÁLNÍ FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Matyáš Lamprecht

Modelování molekulární podobnosti pomocí fragmentů

Katedra softwarového inženýrství

Vedoucí bakalářské práce:	Mgr. Petr Škoda
Studijní program:	Informatika
Studijní obor:	Obecná informatika

Praha 2019

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ..... Podpis autora

Chtěl bych poděkovat panu Mgr. Petru Škodovi za jeho pomoc a rady při zpracování bakalářské práce. Dále bych chtěl poděkovat své rodině za podporu během studia.

Název práce: Modelování molekulární podobnosti pomocí fragmentů

Autor: Matyáš Lamprecht

Katedra: Katedra softwarového inženýrství

Vedoucí bakalářské práce: Mgr. Petr Škoda, Katedra softwarového inženýrství

Abstrakt: Nedílnou součástí vývoje léčiv je tzv. virtuální screening, jehož cílem je počítačová identifikace biologicky aktivních molekul. Jednou z variant virtuálního screeningu je ligandový virtuální screening, jenž je založen na využití známých biologicky aktivních molekul a podobnostního vyhledávání. Molekulu lze reprezentovat jako graf, molekulární podobnost lze pak modelovat na základě stejných fragmentů (podgrafů) mezi dvěma molekulami. Běžnou praxí je fragmenty hashovat do omezeného číselného intervalu a používat tato hashovaná čísla pro výpočet molekulární podobnosti. Při tomto hashování ovšem může dojít ke kolizím. Obecně jsou kolize považovány za nežádoucí, neb dochází ke ztrátě informace o molekule. Našim cílem bylo vyzkoušet, zda-li mohou kolize fragmentů vést k lepším výsledkům. Za tímto účelem jsme navrhli několik podobnostních modelů postavených na fragmentech. Pro účely vyhodnocení jsme implementovali testovací prostředí, jenž umožňuje snadné testování a vyhodnocení různých modelů. Z provedených experimentů plyne, že vybrané kolize vedou k lepším výsledkům, než jsou výsledky běžně používaných metod. Dokonce existují kolize, které v určitém modelu dosahují AUC přesahující 0.99.

Klíčová slova: cheminformatika molekulární reprezentace virtuální screening

Title: Modeling of fragment-based molecular similarity

Author: Matyáš Lamprecht

Department: The Department of Software Engineering

Supervisor: Mgr. Petr Škoda, The Department of Software Engineering

Abstract: Virtual screening is a part of computer-aided drug design, which aims to identify biologically active molecules. The ligand-based virtual screening employs known biologically active molecules and similarity search. A common approach to computation of molecular similarity is to utilize molecular fingerprints. Hashed structural molecular fingerprints hash fragments (subgraphs) of molecular graphs into a bit string reducing the problem of molecular similarity to the bit string similarity. Due to the hashing two distinct fragments may collide, which causes information loss. For this reason collisions are considered unwanted and they are generally believed to decrease a performance. Our goal was, contrary to the general believe, test whether collisions can have positive impact on the performance. For this purpose we designed several similarity models based on fragments. In order to make testing and evaluation easy we implemented testing environment. Results of our experiments prove that some collisions can outperform commonly used methods. Moreover some collisions in a specific model can lead to a performance of AUC over 0.99.

Keywords: cheminformatics molecular representation virtual screening

Obsah

Úvod **3**

1 Molekuly a jejich podobnosti **5**
1.1 Molekulové deskriptory 5
1.2 Molekulové fragmenty 5
1.3 Molekulové otisky 6
1.3.1 Atom pairs 6
1.3.2 Topological Torsion 7
1.3.3 Extended-Connectivity fingerprints 7
1.4 Výpočet podobnosti 8
1.5 LBVS benchmarking 9
1.6 Vyhodnocení LBVS - AUC, EF 10
1.7 Fixace rozdělení 12

2 Cíl experimentu **13**
2.1 Návrh experimentů 13
2.1.1 Počáteční porovnání 13
2.1.2 Hashování 13
2.1.3 Vytváření skupin 13
2.1.4 Ověření indexů 14
2.1.5 Všechny aktivní molekuly 14
2.2 Optimalizace 14
2.3 Softwarová podpora 15

3 Uživatelská dokumentace **17**
3.1 Instalace 17
3.2 Vstupní soubory 17
3.3 Případ použití benchmarkovacího programu 18
3.4 Případ použití výpočtu skupin 19
3.5 Případ použití kreslení obrázků 23

4 Programátorská dokumentace **25**
4.1 Benchmarkovací program 25
4.2 Skupiny 28
4.3 Velmi používané funkce 29
4.4 Kreslení grafů 29

5 Experimenty **30**
5.1 Všechny indexy samostatně 30
5.1.1 Základní indexový model 31
5.1.2 Lineární regrese 32
5.1.3 Rozhodovací stromy 32
5.1.4 Deskriptor model 32
5.1.5 Model tvořený všemi aktivními indexy 32
5.2 Hashování 33
5.2.1 Hashovací model 33

5.3 Indexy do skupin . . . 33
5.3.1 Aktivní skupiny . . . 34
5.3.2 Přidej index . . . 35
5.3.3 Odstraň stejné indexy . . . 35
5.3.4 Ořízni indexy . . . 35
5.3.5 Ořízni a přidej indexy . . . 36
5.3.6 Velikost skupiny . . . 36

6 Výsledky **37**
6.1 Počáteční porovnání . . . 37
6.2 Hashování . . . 38
6.3 Vytváření skupin . . . 39
6.4 Ověření indexů . . . 44
6.5 Všechny aktivní molekuly . . . 46

Závěr **49**

Seznam použité literatury **50**

A Přílohy **52**
A.1 Seznam skriptů a jejich argumentů . . . 52
A.1.1 Benchmarkovací program . . . 52
A.1.2 Skupiny . . . 54
A.1.3 Kreslení obrázků . . . 56
A.2 Názvy skriptů k modelům . . . 56
A.3 Porovnání základních modelů . . . 56
A.4 Porovnání indexů . . . 59

Úvod

Proteiny jsou makromolekuly, které jsou obsažené ve všech živých organismech. Jsou složeny z jednoho či více dlouhých řetězců aminokyselin. Aminokyselina je molekula, která obsahuje karboxylovou (-$COOH$) a aminovou (-NH_2) funkční skupinu. Uspořádání aminokyselin určuje funkci proteinu, mezi ně patří např. řízení biologických a chemických reakcí, replikace DNA a transport látek. Na proteinu existují místa, kam se mohou vázat jiné molekuly a tím ovlivnit činnost proteinu. Tyto molekuly nazýváme biologicky aktivní. Biologicky aktivních molekul se využívá ve farmaceutickém průmyslu při výrobě léčiv.

Při objevování nových léků je cílem tyto biologicky aktivní molekuly najít. To se dělá pomocí testování v laboratoři při procesu tzv. laboratorního screeningu. Molekuly se screeningují v sadách, které obsahují i milion molekul. Cena testu jedné molekuly se pohybuje v rozmezí 0.1\$ $-$ 1\$, tedy provést molekulový screening na sadu o milionu molekul stojí nejméně 100 000\$ [1]. To již je poměrně vysoká částka.

Řešením vysoké částky je omezení sady molekul, které se budou screeningovat. K tomuto omezení se dnes využívají počítače a přístup zvaný virtuální screening (VS). Využití počítačů a informatických postupů při řešení chemických problémů nazýváme chemickou informatikou.

Cílem VS je z databází molekul vybrat molekuly, které mají největší pravděpodobnost k interakci s proteinem. Tyto molekuly jsou poté poslány na laboratorní screening. Existují dva přístupy VS: strukturální (SBVS) a ligandový (LBVS) [2].

SBVS je založen na znalosti struktury cíle, což je nejčastěji protein. U proteinu jsme schopni detekovat vazebné místo, kde dochází k interakci s molekulou [3]. Hlavní metoda SBVS je tzv. docking [4], který si vezme molekulu a vazebné místo a snaží se vyhodnotit, jestli a případně jak může dojít k vytvoření interakce. Docking nejprve musí najít možná umístění molekuly ve vazebném místě a poté tyto pozice vyhodnotit. Výsledkem je seznam pozic molekuly ve vazebném místě a jejich ohodnocení. Molekuly s nejlepším ohodnocením jsou poté vybrány a poslány na laboratorní screening.

LBVS používá princip podobnosti vlastností [2]. Tento princip říká, že více podobné molekuly mají větší pravděpodobnost k vykazování stejné biologické aktivity než méně podobné molekuly. Takže se můžeme dívat na hledání nových biologicky aktivních molekul jako na hledání nejvíce podobných molekul k již známým biologicky aktivním molekulám. Tedy potřebujeme mít sadu již známých biologicky aktivních molekul a na jejich základě ohodnocujeme podle podobnosti molekuly z databáze molekul. Poté opět setřídíme molekuly podle ohodnocení a molekuly s nejlepším ohodnocením vybereme a pošleme na laboratorní screening.

Na úspěšnost použití LBVS má zásadní vliv samotná reprezentace molekul, podle které budeme molekuly porovnávat na podobnost. Nejvíce používaná reprezentace je reprezentace pomocí molekulových otisků - sekce 1.3. Molekulové otisky využívají reprezentace molekuly jako grafu, kde molekulový otisk rozdělí molekulu na fragmenty (podgrafy) a ke každému fragmentu vypočítá index, jenž je hashován. Při tomto hashování může docházet ke kolizím (2 či více různých indexů se zahashuje na stejné místo). Obecně se kolize považují za problematické,

jelikož po hashování může být podobnost 2 molekul jiná než před hashováním. Na druhou stranu je možné na kolize nahlížet jako na schopnost molekulárních otisků generalizovat.

V této práci se podíváme na to, jestli bychom mohli díky kolizím dosáhnout lepších výsledků při nacházení nových biologicky aktivních molekul. Tuto hypotézu budeme ověřovat pomocí série testů, jež jsou provedeny a vyhodnoceny pomocí implementovaného podpůrného softwaru.

1. Molekuly a jejich podobnosti

Na hledání nových biologicky aktivních molekul se můžeme dívat jako na úlohu strojového učení. Budeme mít informace o vlastnostech biologicky aktivních a neaktivních molekul. Tyto informace využijeme pro vytvoření tzv. modelu. Tento model poté použijeme při predikování aktivity molekul. Nyní si popíšeme reprezentace molekul, které se využívají v chemické informatice. Tyto reprezentace popisují chemické a strukturální vlastnosti molekul.

1.1 Molekulové deskriptory

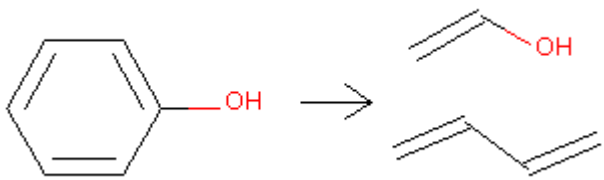
Molekulové deskriptory reprezentují vlastnosti molekuly. Tyto vlastnosti můžeme použít při predikování biologické aktivity. Příklad vlastností pro molekulu z Obrázku 1.1 můžeme vidět v Tabulce 1.1.

Deskriptor	**Hodnota**
Váha molekuly	94,11
Váha molekuly bez vodíků	88,06
Počet valenčních elektronů	36

Tabulka 1.1: Příklad vlastností molekuly z Obrázku 1.1.

1.2 Molekulové fragmenty

Typem molekulových deskriptorů jsou molekulové fragmenty. Molekulu můžeme reprezentovat pomocí strukturálního grafu - Obrázek 1.1, jenž znázorňuje atomy molekuly a jednotlivé vazby mezi nimi.



Obrázek 1.1: Vlevo je příklad strukturálního grafu molekuly. Vrcholy šestiúhelníku reprezentují atomy uhlíku, na které jsou navázány atomy vodíku. Vpravo jsou 2 ukázky molekulových fragmentů této molekuly.

Podgrafy strukturálního grafu budeme nazývat fragmenty. Fragment tedy obsahuje část atomů molekuly a vazby mezi nimi. Každému fragmentu můžeme vypočítat číselný identifikátor, který budeme nazývat index.

1.3 Molekulové otisky

Jednotlivé molekulové otisky používají určitý typ molekulových fragmentů. Molekulové otisky se reprezentují dvěma možnými způsoby:

1. bitovým vektorem, kde každý bit značí, jestli se daný fragment vyskytuje v molekule či nikoliv

2. count otiskem, kde ke každému fragmentu je přiřazen i počet výskytů v molekule (daný fragment se může v molekule vyskytovat vícekrát)

V následujících podsekcích budeme popisovat různé molekulové otisky. Žádný z následujících molekulových otisků neuvažuje při tvorbě fragmentů vodíkové atomy.

1.3.1 Atom pairs

Atom pairs (AP) je reprezentace molekuly, kde bereme všechny dvojice atomů v molekule a nejkratší vzdálenost mezi nimi [5]. Ke každému atomu přidáme jeho popis, jenž se skládá z chemického typu atomu, počtu navázaných nevodíkových atomů a počtu π elektronů. Tedy reprezentace jedné dvojice atomů vypadá takto:

<popis atomu 1>-<vzdálenost>-<popis atomu 2>

Index dvojice atomů se spočítá z popisu atomů ve dvojici a jejich vzdálenosti. Nejkratší vzdálenost započítává oba krajní atomy. Tedy pokud jsou dva atomy spojeny vazbou přímo spolu, tak mají vzdálenost 2. Tabulka 1.2 zobrazuje AP pro molekulu z Obrázku 1.1:

Počet	**AP**	**Index**
4	C,2,1-(2)-C,2,1	689473
4	C,2,1-(3)-C,2,1	689474
2	C,2,1-(4)-C,2,1	689475
2	C,2,1-(2)-C,3,1	705857
2	C,2,1-(3)-C,3,1	705858
1	C,2,1-(4)-C,3,1	705859
2	C,2,1-(3)-O,1,0	1590594
2	C,2,1-(4)-O,1,0	1590595
1	C,2,1-(5)-O,1,0	1590596
1	C,3,1-(2)-O,1,0	1590625

Tabulka 1.2: AP s indexy pro molekulu z Obrázku 1.1.

Na první řádce Tabulky 1.2 vidíme, že se v molekule vyskytují 4 dvojice uhlíků, které jsou přímo spojené vazbou, na obou uhlíkách jsou navázané 2 nevodíkové atomy a oba uhlíky obsahují jeden π elektron. Tímto způsobem se dají interpretovat všechny řádky tabulky.

1.3.2 Topological Torsion

Topological torsion (TT) je reprezentace molekuly, kde vezmeme sekvenci 4 na sebe navazujících nevodíkových atomů, kde ke každému atomu přidáme jeho popis [6]. Schematicky ilustrujeme:

(popis atomu 1)-(popis atomu 2)-(popis atomu 3)-(popis atomu 4)

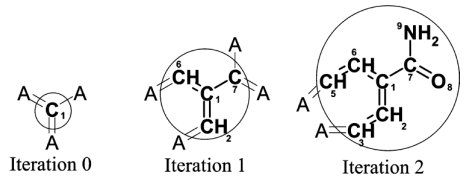
Z těchto informací vypočítáme index. Pro molekulu z Obrázku 1.1 bychom dostali Tabulku 1.3.

Počet	**TT**	**Index**
2	(C,2,1)-(C,2,1)-(C,2,1)-(C,2,1)	5513433129
2	(C,2,1)-(C,2,1)-(C,3,1)-(C,2,1)	5513695273
2	(C,2,1)-(C,2,1)-(C,2,1)-(C,3,1)	5647650857
2	(C,2,1)-(C,2,1)-(C,3,1)-(O,1,0)	12895670313

Tabulka 1.3: TT pro molekulu z Obrázku 1.1. Popis jednotlivých atomů je stejný jako v Tabulce 1.2.

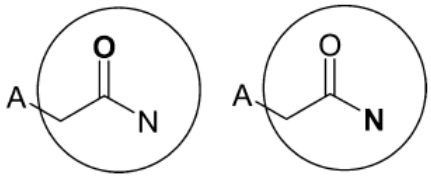
1.3.3 Extended-Connectivity fingerprints

Široce používaná reprezentace molekul se nazývá Extended-Connectivity fingerprints (ECFP) [7]. Tvorba ECFP probíhá následovně. Nejdříve každému atomu v molekule přiřadíme číselný index, který se vypočítá z následujících informací atomu: počet nevodíkových sousedních atomů, počet nevodíkových vazeb, atomové číslo, atomová hmotnost, náboj atomu, počet navázaných vodíků. Indexy všech nevodíkových atomů v molekule tvoří množinu otisků. Poté každý atom společně s atomy, se kterými je spojený vazbou, zapíšeme do pole. Na toto pole použijeme hashovací funkci, abychom opět dostali jednočíselný index. Když všechny atomy vygenerují své nové indexy, tak tyto nové indexy nahradí staré indexy. Nové indexy atomů jsou přidány do množiny otisků. Takto iterujeme až po předem zadaný počet iterací. Ukázka iterace pro počet iterací 2 je na Obrázku 1.2.



Obrázek 1.2: Ilustrace iterace [7].

Po určitém počtu iterací se může stát, že 2 rozdílné atomy obsahují informaci o stejném podgrafu molekuly. Můžeme to vidět na Obrázku 1.3, kde po 2 iteracích je podgraf vygenerovaný kyslíkem (O) stejný jako podgraf vygenerovaný dusíkem (N). Takovéto duplicity se odstraňují.



Obrázek 1.3: Vygenerování stejného podgrafu ze dvou různých počátečních atomů O a N.

Kromě ECFP existuje FCFP, které se liší pouze v přiřazování počátečních indexů, kde je toto přiřazování abstraktnější. K názvu ECFP nebo FCFP přidáváme parametr, který charakterizuje maximální průměr vygenerovaného fragmentu. Např. pro 3 iterace je maximální možný průměr fragmentu roven šesti vazbám, tedy píšeme ECFP6 a FCFP6.

Rogers, Hahn [7] uvedli vhodná čísla pro počet iterací. Pro hledání podobnosti dvou molekul udávají 2 iterace, zatímco pro hledání biologické aktivity udávají větší množství iterací, tedy 3 a více.

Pro molekulu z Obrázku 1.1 a ECFP4 bychom dostali Tabulku 1.4.

Počet	**ECFP4**	**Index**
1	Oc	26234434
3	c(c)c	98513984
2	c(cc)c(c)o	251179073
1	c(O)(cc)cc	859799282
1	O	864662311
2	c(c)c	951226070
1	c(cc)cc	2763854213
1	c(c)(c)O	2905660137
1	c	3217380708
5	c	3218693969
2	c(cc)(cc)	3999906991

Tabulka 1.4: ECFP4 pro molekulu z Obrázku 1.1.

1.4 Výpočet podobnosti

Jak jsme zmínili v úvodu, tak při LBVS se používá podobnostní princip. První způsob výpočtu podobnosti 2 molekul je přes molekulové deskriptory, jako jsou např. váha, počet valenčních elektronů apod. Mějme 2 molekuly a k nim přiřazené deskriptory. Na základě počtu stejných deskriptorů můžeme spočítat jejich podobnost.

Další možnost určení podobnosti je využít přístup, který říká, že jsou 2 molekuly podobné, pokud jedna obsahuje druhou jako podgraf. Tento přístup má ovšem nevýhodu, že vyhledávání na úplný podgraf je poměrně pomalé [8].

Řešením je aproximace pomocí fragmentů. Pokud jedna molekula není podgrafem druhé, pak molekuly nesdílí stejné fragmenty. Ale pokud jedna molekula

je podgrafem druhé, tak druhá molekula obsahuje všechny fragmenty první molekuly. K získání fragmentů používáme molekulové otisky. Nejprve určíme, jaký molekulový otisk budeme používat a ten poté aplikujeme na 2 molekuly. To nám vygeneruje 2 množiny fragmentů a podle počtu stejných fragmentů určíme podobnost. Molekulové otisky lze také zapsat jako bitové vektory, a tak můžeme podobnost 2 molekul redukovat na výpočet podobnosti 2 bitových vektorů. Tuto podobnost pak můžeme počítat pomocí následujících vzorců:
Tanimotův podobnostní koeficient:
Jedná se o nejvíce používaný podobnostní koeficient pro výpočet podobnosti mezi molekulami. Pro molekuly A a B je definovaný následovně:

$$S_{AB} = \frac{s}{a + b - s} \tag{1.1}$$

kde S_{AB} je výsledná podobnost molekul A a B, s je počet stejných bitů nastavených na 1 v bitových vektorech pro A a pro B, a je počet bitů nastavených na 1 v bitovém vektoru pro A a b je počet bitů nastavených na 1 v bitovém vektoru pro B. Tanimotův podobnostní koeficient nabývá hodnot mezi 0 a 1.

Dále se používá **Dicův podobnostní koeficient**, který je definovaný následovně:

$$S_{AB} = \frac{2s}{a + b} \tag{1.2}$$

nebo také **Cosinový podobnostní koeficient**:

$$S_{AB} = \frac{s}{\sqrt{ab}} \tag{1.3}$$

V Dicově i Cosinově podobnostních koeficientech je definice S_{AB}, a, b, s shodná s definicí u Tanimotova podobnostního koeficientu.

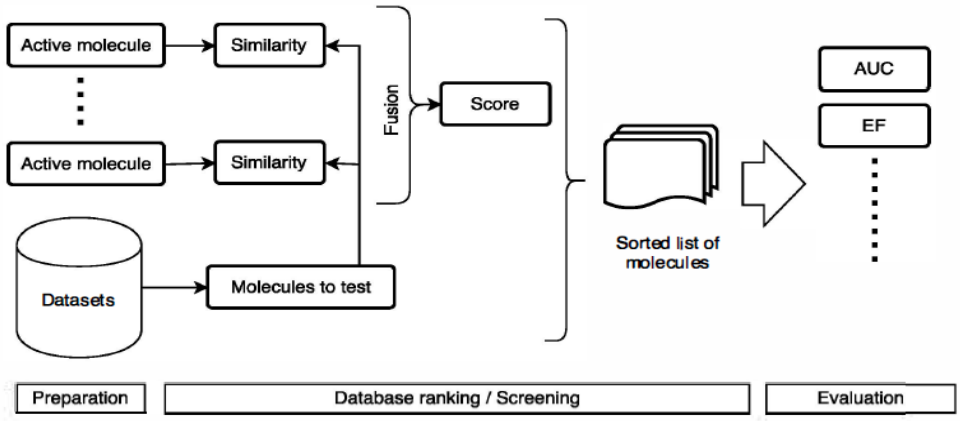
1.5 LBVS benchmarking

Různé výpočty podobností fungují různě. Pro dosažení nejlepších výsledků si musíme vybrat ten nejlepší přístup. Porovnání různých přístupů výpočtu podobností se zjišťuje pomocí benchmarků. Abychom mohli porovnávat různé přístupy výpočtu podobností molekul, potřebujeme mít sadu molekul, u kterých již jejich aktivitu známe. Když tuto sadu molekul máme, tak následují 3 kroky (Obrázek 1.4): příprava dat, provedení LBVS, vyhodnocení.

V přípravě dat dochází k rozdělení na trénovací a testovací sadu molekul. Toto rozdělení se většinou dělá náhodně.

V kroku provedení LBVS dochází k výpočtu podobnosti molekul, kde použijeme jednu z technik uvedených v sekci 1.4. Tedy vezmeme si aktivní molekulu z trénovací sady a přiřazujeme její podobnosti pro všechny molekuly v testovací sadě. Pokud máme více aktivních molekul v trénovací sadě, tak spočítáme všechny podobnosti mezi aktivními molekulami a testovací molekulou. Následně musíme z více podobností udělat jednu. Existují různé přístupy, příkladem mohou být:

1. vybereme nejvyšší dosaženou podobnost a tu použijeme (max-fusion)
2. zprůměrujeme všechny dosažené podobnosti



Obrázek 1.4: Postup při LBVS benchmarkování. Obrázek převzat z [2].

My budeme používat první techniku, protože se používá více [9]. Tuto podobnost molekuly bereme jako její skóre. Molekuly setřídíme od nejvyššího skóre po nejnižší skóre.

V kroku vyhodnocení použijeme jednu z vyhodnocovacích technik. Nejčastěji používané jsou AUC a EF, které jsou detailně popsány v následující sekci 1.6.

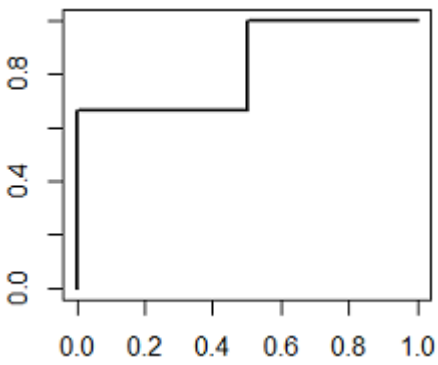
1.6 Vyhodnocení LBVS - AUC, EF

AUC a EF se používají pro vyhodnocení výkonnosti LBVS. Z LBVS máme molekuly setříděné podle jejich skóre od nejvyšší po nejnižší. Podíváme se na testovací sadu a zjistíme, které z testovacích molekul jsou aktivní a které nejsou. Příklad prvních pár řádků hypotetického setříděného seznamu molekul je uveden v Tabulce 1.5.

Pozice	**Skóre**	**Aktivita**
1.	0.9	A
2.	0.85	A
3.	0.82	N
4.	0.8	A
5.	0.78	N

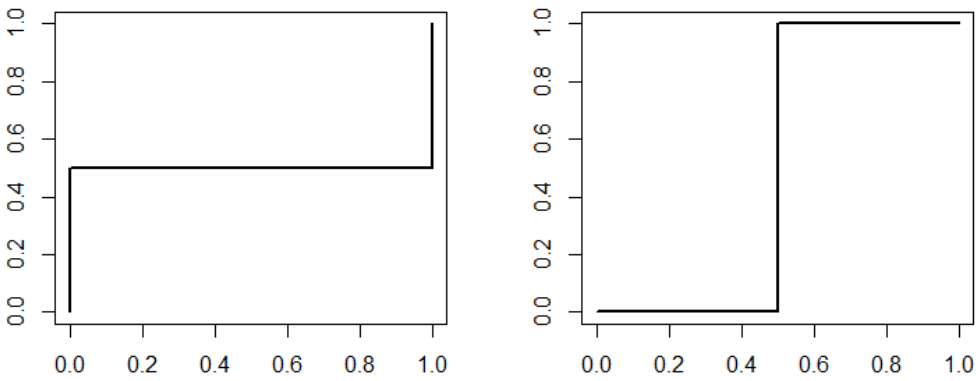
Tabulka 1.5: Hypotetická tabulka setříděného seznamu molekul podle skóre; aktivní (A), neaktivní (N).

Pomocí těchto hodnot poté sestavíme graf, který se nazývá ROC. ROC (Obrázek 1.5) je definována jako TPR vůči FPR, kde FPR je na x-ové ose a TPR je na y-ové ose. TPR je počet aktivních molekul, které se do určité pozice vyskytují ve výsledcích, vydělen počtem aktivních molekul. FPR je počet neaktivních molekul, které se do určité pozice vyskytují ve výsledcích, vydělen počtem neaktivních molekul. ROC pro hodnoty uvedené v Tabulce 1.5 je znázorněna na Obrázku 1.5. AUC počítá plochu pod ROC křivkou [10]. Nejlepší výsledek je 1, nejhorší je 0. AUC rovno 0.5 čekáme od náhodného výběru.



Obrázek 1.5: ROC křivka pro uvedenou Tabulku 1.5.

AUC může dát pro více různých posloupností stejnou plochu pod křivkou (Obrázek 1.6), což nemusí být vhodné. Nás totiž také zajímá úspěšnost pro několik molekul s nejvyšším skóre, které poté pošleme na laboratorní testování.



Obrázek 1.6: Stejná AUC pro 2 rozdílné ROC křivky.

Tedy k tomu využijeme vyhodnocení nazvané enrichment factor (EF). EF je definováno následujícím způsobem:

$$EF(\theta) = \frac{\sum_{i=1}^{n} \delta(r_i)}{(\theta)n}, \text{kde } \delta(r_i) = \begin{cases} 1, & r_i \leq \theta N \\ 0, & r_i > \theta N \end{cases} \tag{1.4}$$

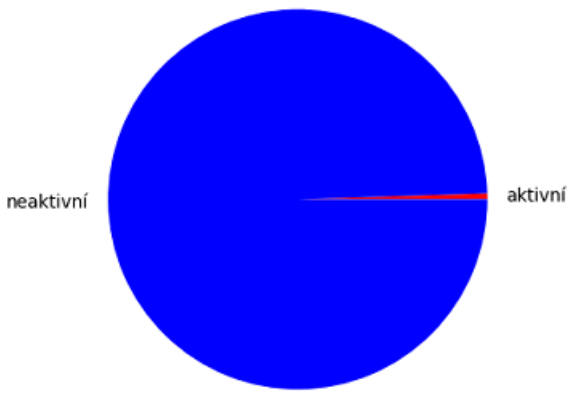
EF měří pouze prvních několik procent, většinou od 0.5% do 5% - což vyhodnocuje kvalitu VS pouze na základě molekul, u kterých jsme spočítali, že mají nejvyšší skóre. Volba parametru (procento, na kolik molekul s nejvyšším skóre se budeme dívat) se u EF považuje za problém, jelikož pro jeho správnou volbu potřebujeme předem znát počet molekul v testovací sadě, protože např. pokud bychom měli 1000 molekul v testovací sadě a parametr 1%, tak měříme úspěšnost prvních 10 molekul. Pokud bychom ovšem měli v testovací sadě miliony molekul, tak 1% je již poměrně velké číslo a to už nám neřekne moc informací o kvalitě vyhodnocení pro několik málo molekul s nejvyšším skóre.

Kombinace AUC a EF nám dá informaci, jak je daný přístup VS dobrý.

1.7 Fixace rozdělení

Jak jsme zmínili v sekci 1.5, tak rozdělení na trénovací a testovací data většinou probíhá náhodně. Ovšem v článku Škody, Hokszy [2] je zmíněné, že toto náhodné rozdělení nemusí být vždy vhodné, jelikož náhodné rozdělení vede k nezopakovatelným výsledkům, protože při více spuštěních stejného výpočtu dostaneme na začátku jiné rozdělení dat a to většinou ovlivní i výsledky. Navíc chemicko-informatické úlohy mají tu vlastnost, že je zde veliký nepoměr aktivních a neaktivních molekul [10] (viz Obrázek 1.7). Tedy my při našich experimentech budeme používat tzv. fixované rozdělení. To znamená, že budeme mít předem určené:

1. aktivní trénovací molekuly
2. neaktivní trénovací molekuly
3. testovací molekuly



Obrázek 1.7: Poměr aktivních a neaktivních molekul v námi používané testovací sadě.

2. Cíl experimentu

Ve zbytku práce platí, že pokud budeme mluvit o deskriptorech, tak tím myslíme deskriptory uvedené v sekci 1.1, ale ne už molekulové fragmenty z sekce 1.2.

V této kapitole si nejprve popíšeme experimenty, které budeme provádět. Poté si popíšeme problémy, které byly potřeba vyřešit pro návrh a realizaci experimentů.

2.1 Návrh experimentů

V našich experimentech se budeme zaměřovat na molekulové otisky a především na ECFP, protože je to široce používaný a dobře interpretovatelný otisk. Jednotlivé experimenty jsou implementovány modely. V práci se zaměříme na 5 experimentů, které jsou popsány v následujících podsekcích.

2.1.1 Počáteční porovnání

Abychom mohli určit, jestli námi definované přístupy jsou dobré či špatné, tak si potřebujeme obstarat referenční výsledky, s kterými budeme naše přístupy porovnávat. Tyto referenční výsledky budou obsahovat výsledky běžně používaných metod a metod používaných ve strojovém učení.

Za běžně používané metody považujeme molekulové otisky, tedy nejprve provedeme měření výkonnosti jednotlivých molekulových otisků a porovnáme je. U kruhových otisků (ECFP, FCFP) provedeme porovnání s různě nastavenými průměry otisků. Dále tyto základní, běžně používané metody, porovnáme s tradičními metodami strojového učení. Nakonec je porovnáme s námi navrženými přístupy.

2.1.2 Hashování

Při reprezentaci molekulových otisků pomocí bitových vektorů se používá hashování. To funguje tak, že se vezme bitový vektor molekuly a pozice bitů se vymodulí číslem n. Což znamená, že ze 2 (či více) různých fragmentů můžeme dostat stejný index. Tím vlastně 2 (či více) rozdílné fragmenty považujeme za stejné. Obecně se kolize považují za problematické, protože tím ztrácíme informaci o molekule. My máme hypotézu, že by kolize mohly pomoci, což je v rozporu s obecným pohledem na kolize. Tuto hypotézu bychom chtěli otestovat.

2.1.3 Vytváření skupin

Jak jsme uvedli v minulé podsekci, tak máme hypotézu, že kolize mohou pomoci výkonnosti. Na kolize můžeme nahlížet jako na vytváření skupin, což poté vede k otázce, jestli jsou nějaké skupiny vhodnější než jiné. Tedy v tomto experimentu budeme vytvářet různé skupiny indexů a ty vyhodnocovat. O indexech, které se vyskytují ve stejné skupině, řekneme, že jsou stejné.

Např. pokud budeme chtít vytvářet skupiny z indexů, které dostaneme z ECFP4, tak si nejprve vezmeme hodnoty, které vyšly pro ECFP4 v podsekci 2.1.1

(hodnoty AUC a EF pro různé parametry). Poté si určíme, podle kterého kritéria budeme skupiny vybírat tzn. že např. budeme vybírat pouze takové skupiny, které dosáhly lepších výsledků pro AUC a EF 1%. Poté hodnoty AUC a EF 1%, které jsme dostali pro ECFP4, budeme nazývat základními hodnotami a ECFP4 budeme nazývat základním modelem. My se budeme zaměřovat na AUC, tedy budeme mít jako základní hodnotu, hodnotu AUC. Následně již vezmeme indexy všech aktivních molekul, které dostaneme pro určitý molekulový otisk a dáme je dohromady. Odstraníme vícenásobné výskyty stejných indexů, abychom nedostávali skupiny stejných indexů. Poté z těchto indexů začneme skládat dvojice. Tyto dvojice použijeme pro výpočet skóre testovacích molekul. Vyhodnocení jednotlivých dvojic porovnáme se základní hodnotou, a pokud bude mít určitá dvojice lepší vyhodnocení, než je základní, tak si ji ponecháme. Ponechané dvojice poté skládáme do trojic, pokud dané 2 dvojice mají stejný index, nebo do 2 dvojic, pokud stejný index nemají. Tyto skupiny indexů následně vyhodnotíme, porovnáme se základní hodnotou a opět si ponecháme skupiny, které mají lepší vyhodnocení, než je základní. Takto budeme iterovat, abychom mohli porovnávat různé velikosti skupin a najít nejlepší možnou výkonnost.

Ve zbytku práce budeme označovat skupiny, kde jsou všechny indexy samostatně (tedy žádné skupiny neexistují) jako skupiny 1. iterace. Dvojice budeme označovat jako skupiny 2. iterace, trojice či 2 dvojice jako skupiny 3. iterace atd.

2.1.4 Ověření indexů

Zde bychom chtěli zjistit, jestli jsme schopni určit skupiny, které dávají lepší výsledek, než je základní, pouze z množiny indexů. Tedy se zaměříme na skupiny z předešlého experimentu, které měly lepší hodnotu AUC než byla základní. Jelikož budeme mít více modelů, které budou vytvářet skupiny, tak nás bude zajímat, jestli existují nějaké indexy, které se vyskytují ve skupinách, jenž dávají lepší hodnotu ve více modelech.

2.1.5 Všechny aktivní molekuly

V posledním experimentu by nás zajímalo, jestli bychom mohli dostat ještě lepší výsledky, kdybychom daný model učili na všech aktivních molekulách z trénovací sady aktivních molekul a testovací sady dohromady. Aktivní molekuly z testovací sady totiž mohou obsahovat jiné indexy, než obsahují aktivní molekuly z trénovací sady. Z těchto indexů opět budeme skládat skupiny 2. iterace, skupiny 3. iterace atd. a budeme pozorovat, jestli znalost všech indexů může pomoci výkonnosti.

2.2 Optimalizace

Jedno jádro počítače o výkonnosti 1.5GHz vypočítá cca výsledky 20 skupin za minutu. Když dáme všechny indexy aktivních molekul pro ECFP4 z naší trénovací sady aktivních molekul dohromady a odstraníme vícenásobné výskyty stejného indexu, tak dostaneme 148 indexů. Tedy skupin 2. iterace bude celkově:

$$\binom{148}{2} = 10878 \tag{2.1}$$

Vypočítat vyhodnocení skupin 2. iterace na jednom jádře o uvedené výkonnosti by trvalo zhruba 9 hodin.

Řekněme, že by skupin 2. iterace, pro které vyšel lepší výsledek AUC, než je základní, bylo 3000 (což nám i u některých experimentů vyšlo více). Pak by skupin 3. iterace bylo:

$$\binom{3000}{2} = 4498500 \tag{2.2}$$

Vypočítat vyhodnocení všech skupin 3. iterace na jednom jádře by tedy trvalo zhruba 156 dní. My sice budeme výpočty pouštět na více jádrech, které mají vyšší výkonnost než je 1.5GHz, ale i tak by výpočet běžel několik dní a to jsme pouze u skupin 3. iterace jednoho modelu. My ale chceme provádět více iterací a chceme vyzkoušet více modelů. Tedy kdybychom tyto výpočty chtěli spustit pro všechny skupiny 3. iterace, 4. iterace atd., které budou mít lepší výkonnost, pro všechny modely, tak by výpočty trvaly i několik stovek dní. Takže my ze skupin 2. iterace, skupin 3. iterace atd. budeme vybírat pouze skupiny s nejlepšími výsledky.

2.3 Softwarová podpora

Abychom mohli porovnávat různé přístupy LBVS, tak potřebujeme benchmarkovací program. Ten funguje následovně.

Jak jsme zmínili v sekci 1.7, tak na vstupu máme fixně rozdělené 3 sady molekul:

1. trénovací sada obsahující aktivní molekuly
2. trénovací sada obsahující neaktivní molekuly
3. testovací sada

Každá sada je uložena ve svém souboru. K testovacím molekulám ještě potřebujeme znát jejich aktivitu, což je 4. vstupní soubor.

Experimenty jsou založeny na indexech fragmentů molekul, tedy musíme být schopni extrahovat fragmenty molekulových otisků a ke každému fragmentu musíme umět spočítat jeho index. Některé experimenty jsou založeny na deskriptorech např. strojové učení. Tedy potřebujeme pro samotnou molekulu nebo pro její fragmenty umět vypočítat deskriptory.

Poté vytváříme z informací o molekulách (jejich fragmenty, deskriptory) z trénovacích sad model. Jaké informace se pro tvorbu daného modelu využijí, závisí vždy na konkrétní implementaci modelu.

Modelů budeme vytvářet velké množství, tedy bychom chtěli jednotlivé modely umět rozlišit, abychom mohli určit, který se má zrovna spustit. To uděláme tak, že každému modelu přiřadíme jméno a podle tohoto jména se rozhodne, který model se spustí.

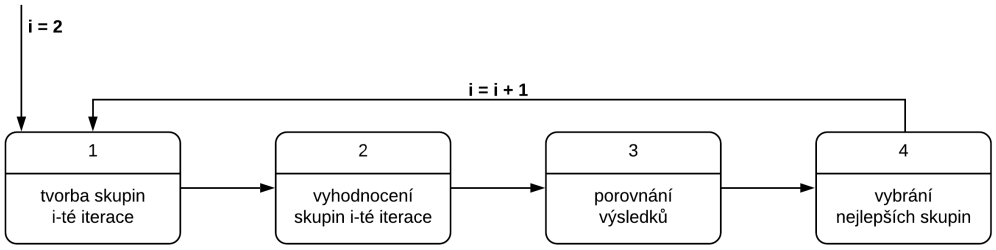
Model můžeme následně použít k výpočtu skóre jednotlivých molekul z testovací sady. Opět záleží na jméně modelu, podle kterého se určí, jaký výpočet skóre se má pro testovací molekuly použít.

Poté k testovacím molekulám přidáme jejich aktivitu, setřídíme je podle skóre a nakonec vypočítáme hodnoty AUC a EF (budeme nazývat vyhodnocením modelu). Toto již je pro všechny modely stejné. Chtěli bychom, aby tento benchmarkovací program byl konfigurovatelný pomocí souboru. Pro snadnější porovnávání

výkonnosti jednotlivých modelů budeme chtít vykreslit grafy, které tyto modely porovnávají.

U EF jsme se rozhodli počítat EF pro 2 parametry a to EF 1% a EF 5%. Naše testovací sada obsahuje 3920 molekul, tedy EF 1% bude obsahovat vyhodnocení pro 39 molekul s nejvyšším skóre a EF 5% bude obsahovat vyhodnocení pro 196 molekul s nejvyšším skóre. Rozhodli jsme se pro tyto hodnoty, protože v kombinaci s AUC dostaneme dobrou představu o kvalitě daného modelu.

Jak jsme již zmínili, tak naším hlavním cílem je zkoumání skupin indexů. Tedy potřebujeme umět si ze všech indexů molekulového otisku daných aktivních molekul ponechat od každého indexu pouze jeden výskyt a následně vygenerovat všechny skupiny 2. iterace. Poté potřebujeme umět modely obsahující skupiny 2. iterace indexů vyhodnotit, o což se postará benchmarkovací program, který jsme popsali výše. Dále musíme výsledky všech skupin 2. iterace porovnat se základními hodnotami. Pokud by skupin 2. iterace, které by dosáhly lepších výsledků, bylo velké množství, tak by se nám hodilo umět na základě určitých kritérií vybrat pouze určitý počet nejlepších skupin. Následně musíme být schopni z těchto skupin 2. iterace udělat skupiny 3. iterace, ty vyhodnotit a následně porovnat se základními hodnotami. Takto musíme být schopni iterovat, dokud budeme dostávat lepší výsledky, než jsou základní nebo dokud bude uživatel požadovat. Postup iterací je zobrazen na Obrázku 2.1.



Obrázek 2.1: Postup iterací.

3. Uživatelská dokumentace

V této kapitole si na případech použití ukážeme běh programu. Detailní popis skriptů, kde ke každému skriptu jsou uvedeny všechny argumenty je v příloze A.1.

3.1 Instalace

Pro běh programu budeme potřebovat Python 3.5. Je potřeba si ručně stáhnout a nainstalovat knihovnu RDKit. Zde je průvodce instalací [1]. Poté budeme potřebovat ještě další 2 knihovny - scikit-learn a Matplotlib, které si můžeme nainstalovat následujícími příkazy:

```
pip install -U scikit-learn
pip install -U matplotlib
```

Námi napsané skripty si můžeme stáhnout z:
`https://github.com/LamprechtMatyas/MolecularSimilarity.git`
nebo za použití Gitu pomocí příkazu:

```
git clone https://github.com/LamprechtMatyas/MolecularSimilarity.git
```

Další možnost je použít Docker. Dockerfile vypadá následujícím způsobem:

```
FROM skodapetr/rdkit:Release_2018_09_1

RUN apk upgrade --no-cache \
    && apk update \
    && apk add --no-cache g++ python3-dev git py3-scipy py-numpy-dev freetype-dev \
    && pip3 install --upgrade pip \
    && pip3 install --no-cache-dir cython scikit-learn matplotlib

WORKDIR /opt/molecule-similarity

RUN git clone https://github.com/LamprechtMatyas/MolecularSimilarity.git
```

Pokud níže uvedené případy použití programu budeme spouštět v Dockeru, tak místo *python* budeme psát *python3*.

3.2 Vstupní soubory

Pro běh benchmarkovacího programu a výpočtu vyhodnocení skupin potřebujeme 5 vstupních souborů (ukázky vstupních souborů jsou v adresáři *data*):

1. soubor s aktivními molekulami - *data/actives.smi*
2. soubor s neaktivními molekulami - *data/inactives.smi*
3. soubor s testovacími molekulami - *data/test.smi*
4. soubor obsahující aktivitu testovacích molekul - *data/test_activity.json*

[1] `https://www.rdkit.org/docs/Install.html`

5. konfigurační soubor modelu - jakýkoliv soubor z adresáře *data/configuration_files*

Soubor *data/actives.smi* obsahuje 5 aktivních molekul, soubor *data/inactives.smi* obsahuje 100 neaktivních molekul a soubor *data/test.smi* obsahuje nejprve 3900 neaktivních molekul (jejich jméno začíná na "ZINC") a poté 20 aktivních molekul (jejich jméno je samostatné číslo). Soubor s aktivními molekulami z *data/actives.smi* vypadá následovně:

```
COc1ccc2[nH]cc(CC[NH+](C)C)c2c1        1832
Cc1oc2ccc(NC(=O)c3cccs3)nc2c1CC[NH+](C)C        12094290
C[N@H+]1CC[C@H](Oc2cccc(NC(=O)c3ccc(F)cc3F)c2F)CC1        44216159
C[N@H+]1CC[C@H](c2c[nH]c3ccc(C(=O)CCc4cn[nH]c4)cc32)CC1        23003787
Cc1oc2ccc(NC(=O)c3ccncc3)cc2c1CC[NH+](C)C        19064681
```

Vidíme, že na každé řádce souboru je jedna molekula - její chemická reprezentace a jméno molekuly (jméno první molekuly je 1832). Tyto 2 informace jsou odděleny tabulátorem. Stejným způsobem vypadá i soubor s neaktivními molekulami i soubor s testovacími molekulami.

Soubor obsahující aktivitu testovacích molekul z *data/test_activity.json* vypadá následovně:

```
{"activity": [0, 0, 0, ..., 0, 0, 1, 1, ..., 1]}
```

kde pozice nul a jedniček odpovídá řádkům neaktivních a aktivních molekul z souboru *data/test.smi*. 0 reprezentuje neaktivní molekulu, 1 reprezentuje aktivní molekulu.

Příklady konfiguračních souborů můžeme vidět v adresáři *data/configuration_files*. Každý konfigurační soubor musí obsahovat jméno modelu, typ molekulového otisku jako je vidět např. v *rdkit_ecfp_model.json*. Typ molekulového otisku musí mít následující tvar:

- tt.{velikost}
- ecfp.{velikost}
- fcfp.{velikost}
- ap

kde velikost pro tt je počet na sebe navazujících atomů, pro ecfp a fcfp je to průměr fragmentu. Zbytek konfigurace záleží na konkrétním modelu.

3.3 Případ použití benchmarkovacího programu

Řekněme, že chceme změřit výkonnost modelu *rdkit_ecfp_model.json*, který se nachází v adresáři *data/configuration_files*, pro soubory s molekulami, které jsme uvedli v minulé sekci 3.2. Výsledek bychom chtěli vypsat do souboru *data/evaluation.json*.

Na to použijeme skript *run_all_scripts.py* následujícím způsobem:

```
python run_all_scripts.py
   -m data/configuration_files/rdkit_ecfp_model.json
   -o data/evaluation.json
```

kde *-m* je soubor obsahující konfiguraci modelu; *-o* je soubor, kam se vypíše vyhodnocení. Nemusíme nastavovat cesty k souborům s molekulami, ani k souboru s aktivitou testovacích molekul, které máme v adresáři *data*, protože jsou v programu implicitně nastavené. Kdybychom chtěli použít jiné sady molekul, tak to už bychom cesty k nim nastavovat museli.

3.4 Případ použití výpočtu skupin

Řekněme, že bychom chtěli otestovat skupiny pro *add_group_model*. Chtěli bychom dělat skupiny pro indexy z ECFP2 a chtěli bychom se zaměřit na AUC. Tedy za základní hodnoty budeme považovat pouze jednu hodnotu a to hodnotu AUC, která vyšla ve výše uvedeném benchmarkovacím programu, jelikož ten jsme pouštěli pro ECFP2. Pokud bychom výsledků, které přesáhly základní hodnotu, měli mnoho, tak bychom chtěli vybrat 50 výsledků, které dosáhly nejvyššího AUC. Jako vstupní soubory s molekulami použijeme soubory, které jsme uvedli v sekci 3.2. Pro výpočet budeme chtít použít 3 jádra.

Nejprve musíme spustit benchmarkovací program, tentokrát ovšem za účelem získání fragmentů a indexů, z kterých následně budeme dělat skupiny (argument *-d*). Tedy spustíme model *add_group_model* s prázdnými skupinami (skupiny 1. iterace):

```
python run_all_scripts.py
   -m data/configuration_files/baseline_add_group_model.json
   -o data/output_of_run_all_scripts/evaluation.json
   -d data/output_of_run_all_scripts
```

Máme fragmenty a indexy. Nyní začneme skládat indexy do skupin 2. iterace a ty vyhodnocovat. O to se stará skript *run_all_pair.py*:

```
python run_all_pair.py
   -i data/output_of_run_all_scripts/fragmentsa.json
   -t data/output_of_run_all_scripts/fragmentst.json
   -a data/test_activity.json
   -m add_group_model
   -n 3
   -o data/pairs
```

kde *-i* je soubor, který obsahuje fragmenty s indexy aktivních molekul, byl vypsán pod argumentem *-d* z *run_all_scripts.py*; pro *-t* platí to stejné jako pro *-i*, akorát obsahuje fragmenty s indexy testovacích molekul; *-a* je soubor s aktivitou testovacích molekul; *-m* je jméno modelu; *-n* je počet jader, na kterých má běžet výpočet; *-o* je jméno adresáře, kam se mají uložit výstupní adresáře.

Výstupní adresáře jsou celkem 4:

- configurationfiles - obsahuje konfigurační soubory modelů, každý model obsahuje jednu skupinu 2. iterace
- scorefiles - obsahuje soubory s testovacími molekulami a k nim vypočítané jejich skóre, pro každý model jeden soubor
- activities - přidá k molekulám ze souborů ze scorefiles jejich skutečnou aktivitu
- evaluations - obsahuje soubory s vyhodnocením, každý soubor obsahuje vyhodnocení pro určitou skupinu 2. iterace

Poté chceme vybrat pouze ty skupiny 2. iterace, které dosáhly lepšího výsledku, než je základní hodnota. Na to použijeme skript *pair_analysis.py*:

```
python pair_analysis.py
   -f data/output_of_run_all_scripts/fragmentsa.json
   -b data/evaluation.json
   -d data/pairs/evaluations
   -o data/pairs_analysis
```

kde *-f* je opět soubor obsahující fragmenty s indexy aktivních molekul; *-b* je soubor obsahující vyhodnocení pro ECFP2; *-d* je adresář souborů s vyhodnocením, který se vypsal v *run_all_pair.py*; *-o* je adresář, kam se vypíší výstupní soubory. Skupiny 2. iterace jsou roztříděny do souborů podle toho, v jakých aspektech vyhodnocení dosáhly lepších výsledků, než je vyhodnocení pro ECFP2. Tedy se ve výstupním adresáři objeví následujících 9 souborů:

1. auc.json - obsahuje skupiny 2. iterace, které dosáhly lepšího výsledku v měření AUC
2. aucef1.json - obsahuje skupiny 2. iterace, které dosáhly lepších výsledků v měření AUC a EF 1%
3. aucef5.json - obsahuje skupiny 2. iterace, které dosáhly lepších výsledků v měření AUC a EF 5%
4. aucef1ef5.json - obsahuje skupiny 2. iterace, které dosáhly lepších výsledků v měření AUC, EF 1% a EF 5%
5. ef1ef5.json - obsahuje skupiny 2. iterace, které dosáhly lepších výsledků v měření EF 1% a EF 5%
6. ef1.json - obsahuje skupiny 2. iterace, které dosáhly lepšího výsledku v měření EF 1%
7. ef5.json - obsahuje skupiny 2. iterace, které dosáhly lepšího výsledku v měření EF 5%
8. greater.json - obsahuje skupiny 2. iterace, které dosáhly lepšího výsledku v alespoň jednom měřitelném kritériu (AUC, EF 1%, EF 5%)
9. baseline.json - obsahuje informace o vyhodnocení základního modelu - v tomto případě ECFP2

My se zaměřujeme na AUC, tedy si vybereme soubor *auc.json* (pokud bychom za základní hodnoty považovali EF 1% a EF 5%, tak bychom si vybrali soubor *ef1ef5.json*). Jak jsme zmínili, tak vždy budeme vybírat pouze 50 nejlepších výsledků podle AUC. Na to použijeme skript *select_best_results.py*:

```
python select_best_results.py
   -f data/pairs_analysis/auc.json
   -b 50
   -t AUC
   -o data/pairs_best_auc_results/best_auc.json
```

kde *-f* je soubor, ze kterého dané nejlepší výsledky vybíráme; *-b* je číslo, kolik nejlepších výsledků chceme vybrat; *-t* je kritérium, podle kterého budeme nejlepší výsledky vybírat, my chceme nejlepší hodnoty podle AUC; *-o* je výstupní soubor, který bude obsahovat 50 skupin 2. iterace, které dosáhly nejlepších výsledků, seřazených podle AUC.

Samozřejmě se může stát, že z *pair_analysis.py* dostaneme méně než 50 lepších výsledků, než je základní hodnota a pouštět *select_best_results.py*, pak nedává smysl.

Dále bychom chtěli ze skupin 2. iterace dělat skupiny 3. iterace a tyto skupiny vyhodnotit. O to se stará skript *run_all_groups.py*. Pokud jsme měli více než 50 lepších výsledků a tedy jsme spustili *select_best_results.py*, tak použijeme následující příkaz:

```
python run_all_groups.py
   -i data/output_of_run_all_scripts/fragmentsa.json
   -g data/pairs_best_auc_results/best_auc.json
   -t data/output_of_run_all_scripts/fragmentst.json
   -a data/test_activity.json
   -m add_group_model
   -n 3
   -o data/triples
```

kde *-i* je soubor obsahující fragmenty s indexy aktivních molekul; *-g* je soubor, ze kterého budeme skládat skupiny 3. iterace, dostali jsme ho z *select_best_results.py*; *-t* je soubor obsahující fragmenty s indexy testovacích molekul; *-a* je soubor obsahující aktivitu testovacích molekul; *-m* obsahuje jméno modelu; *-n* je počet jader, na kterých poběží výpočet; *-o* je adresář, kam se uloží 4 výstupní adresáře, které jsou shodné s výstupními adresáři z *run_all_pair.py*, akorát výstupní soubory neobsahují skupiny 2. iterace, ale skupiny 3. iterace.

Pokud jsme ovšem nedostali ani 50 lepších výsledků, než je základní hodnota a tedy jsme nepouštěli *select_best_results.py*, tak příkaz bude následující:

```
python run_all_groups.py
   -i data/output_of_run_all_scripts/fragmentsa.json
   -g data/pairs_analysis/auc.json
   -t data/output_of_run_all_scripts/fragmentst.json
   -a data/test_activity.json
   -m add_group_model
   -n 3
   -o data/triples
```

kde se liší pouze soubor, ze kterého budeme skládat skupiny 3. iterace, nyní použijeme soubor, který jsme dostali přímo z *pair_analysis.py*.

Poté potřebujeme vybrat skupiny 3. iterace, které dosáhly lepšího výsledku, než je základní hodnota, na což tentokrát použijeme skript *group_analysis.py*:

```
python group_analysis.py
   -b data/evaluation.json
   -c data/triples/configurationfiles
   -e data/triples/evaluations
   -o data/triples_analysis
```

kde *-b* je soubor obsahující vyhodnocení pro ECFP2; *-c* je adresář obsahující konfigurační soubory pro skupiny 3. iterace; *-e* je adresář obsahující vyhodnocení jednotlivých skupin 3. iterace; *-o* je adresář, kam se uloží výsledky roztříděné do stejných souborů jako z *pair_analysis.py*.

Vybereme 50 nejlepších výsledků pomocí *select_best_results.py* (pokud máme alespoň 50 lepších výsledků):

```
python select_best_results.py
   -f data/triples_analysis/auc.json
   -b 50
   -t AUC
   -o data/triples_best_auc_results/best_auc.json
```

Poté můžeme pokračovat a iterovat v pořadí:

1. *run_all_groups.py*
2. *group_analysis.py*
3. *select_best_results.py*

Takto můžeme iterovat, dokud dostáváme lepší výsledky, než je základní hodnota. Akorát musíme měnit vstupní a výstupní soubory, tedy v dalším kroku bychom změnili *pairs* na *triples* a *triples* např. na *quatres*.

Dá se předpokládat, že do určité velikosti skupin se výsledky nejvyšších dosažených AUC budou zvyšovat, a pak od této velikosti se výsledky nejvyšších AUC budou postupně snižovat, ale pořád budou nad hranicí základní hodnoty AUC, kterou jsme dostali z *run_all_scripts.py*. Jelikož tyto snižující se hodnoty již nemusí být tolik zajímavé, tak v *group_analysis.py* funguje to, že ze vstupního souboru, který dáme do argumentu *-b* se vypočítávají průměrné hodnoty vyhodnocení, které se v tomto souboru vyskytují, tedy tím můžeme zvedat základní hodnotu. To znamená, že pokud mu dáme soubor z *run_all_scripts.py*, tak se zde nemá co průměrovat, protože je zde obsaženo pouze jedno vyhodnocení. Tedy základní hodnota bude pořád stejná. Pokud bychom ale chtěli, aby se základní hodnota zvyšovala a tím pádem bychom pro snižující se hodnoty pro skupiny již nedostali žádné výsledky, tak bychom mohli dát jako vstupní soubor argumentu *-b* soubor, který dostaneme z *pair_analysis.py* nebo z *group_analysis.py* nebo z *select_best_results.py* .

Pokud bychom např. dostávali velmi dobré výsledky a chtěli bychom otestovat, jestli se výsledky ještě zlepší, kdybychom vybírali vždy 100 nejlepších výsledků, tak můžeme celý výše uvedený proces spustit stejným způsobem, akorát v *select_best_results.py* dáme argumentu *-b* 100. Při tomto výpočtu se již přeskočí vyhodnocené skupiny.

Pokud pouštíme celý výpočet modelu poprvé, je důležité, aby adresář, kam se mají vypsat výsledky z *run_all_pair.py* nebo *run_all_groups.py*, neobsahoval tyto podadresáře: *activities*, *configurationfiles*, *evaluations*, *scorefiles*. V případě, že je obsahuje, tak musejí být prázdné. Pokud stejný proces pouštíme znova pro větší počet skupin, tak tyto podadresáře existovat budou, ale nyní je to v pořádku v případě, že tyto adresáře obsahují pouze soubory vygenerované menším počtem skupin.

3.5 Případ použití kreslení obrázků

Pro jeden soubor z *pair_analysis.py* nebo *group_analysis.py* nebo *select_best_results.py* můžeme vykreslit obrázky, které obsahují grafy a tabulky za použití programu *print_group_graphs.py*. Např. vykreslení obrázků pro soubor *auc.json*, který jsme dostali z *pair_analysis.py*:

```
python print_group_graphs.py
   -b data/pairs_analysis/baseline.json
   -i data/pairs_analysis/auc.json
   -o data/graphs/pairs
```

kde *-b* je soubor *baseline.json*, který jsme vypsali v *pair_analysis.py*; *-i* je soubor, který jsme dostali z *pair_analysis.py* kromě souboru *baseline.json*; *-o* je adresář, kam se uloží výsledné png soubory. V adresáři s výstupními soubory najdeme boxploty pro každý index, který se vyskytoval ve vstupním souboru. Každý boxplot je pojmenován podle indexu. Dále se zde vyskytují 2 histogramy:

1. AUC.png - obsahuje hodnoty AUC a k nim jejich počet
2. fragments.png - znázorňuje kolikrát se kolik indexů vyskytlo ve výsledcích

Nakonec se vykreslí obrázky, které obsahují tabulky, jenž měří indexy: jejich AUC a jejich počet. Dostaneme celkem 5 takovýchto obrázků - každý vždy obsahuje 15 nejlepších hodnot:

1. index_auc.png - obsahuje indexy seřazené podle nejvyššího AUC
2. index_auc_group.png - obsahuje indexy seřazené podle nejvyššího AUC a k nim přiřazený počet skupin 2. iterace, ve kterých se daný index vyskytuje
3. index_groups.png - obsahuje indexy seřazené podle počtu skupin 2. iterace, ve kterých se daný index vyskytuje
4. index_groups_AUC.png - obsahuje indexy seřazené podle počtu skupin 2. iterace, ve kterých se daný index vyskytuje, a k nim přiřazené nejvyšší dosažené AUC daného indexu
5. index_number_ranking.png - obsahuje indexy seřazené podle počtu skupin 2. iterace, ve kterých se daný index vyskytuje, a k nim přiřazené pořadí jejich nejvyššího AUC

Druhá možnost vykreslení obrázků je vykreslení obrázků pro všechny analýzy skupin. Můžeme použít soubory z *pair_analysis.py* a *group_analysis.py* nebo z *select_best_restults.py*. Tedy např. vykreslení obrázků pro soubory z *select best_results.py* pro skupiny 2. iterace a skupiny 3. iterace by bylo následovné:

```
python print_all_analyses_graphs.py
   -i data/pairs_best_auc_results/best_auc.json,
      data/triples_best_auc_results/best_auc.json
   -n 2.iterace,3.iterace
   -o data/graphs/all_add_group
```

kde *-i* jsou vstupní soubory oddělené čárkou; *-n* jsou jejich jména, pod kterými budou reprezentovány ve výsledných obrázcích; *-o* je adresář, kam se vykreslí výstupní soubory. Vykreslí se stejné grafy jako v *print_group_graphs.py* a navíc ještě *highest_auc.png*, kde se pro každou iteraci vykreslí její maximální AUC. Obrázky obsahující tabulky jsou následující:

1. index_number_totally.png - indexy seřazené podle počtu skupin, ve kterých se vyskytují napříč vstupními soubory a ke každému vstupnímu souboru je vypsáno číslo, kolikrát se daný index vyskytuje ve skupinách
2. table_auc_groups.png - indexy seřazené podle nejvyššího AUC a k nim je přiřazen počet všech skupin, ve kterých se daný index vyskytuje
3. table_groups_auc.png - indexy seřazené podle počtu skupin a k nim je přiřazeno jejich nejvyšší dosažené AUC
4. table_highest_auc.png - AUC seřazené od nejvyššího a ke každému AUC dodáno, v jakém vstupním souboru se toto AUC vyskytlo

4. Programátorská dokumentace

Celý program je implementován v Pythonu [11]. Vybrali jsme si Python, protože ve světě chemické informatiky je velice používaný a tedy v něm existuje mnoho knihoven, které obsahují výpočetní, vyhodnocovací, statistické a grafové funkce, které budeme používat.

Pro běh celého programu jsou potřeba následující 3 knihovny:

- RDKit
- scikit-learn
- Matplotlib

RDKit je knihovna pro chemicko-informatické účely [12]. Obsahuje mnoho funkcí přes samotné čtení a zapisování molekul až po naimplementované všechny uvedené molekulové otisky (AP, TT, ECFP, FCFP). Dále obsahuje funkce pro vyhodnocení, tedy pro výpočet AUC a EF.

Jak jsme psali v podsekci 2.1.1, tak budeme chtít porovnat molekulové otisky s tradičními přístupy strojového učení. Pro strojové učení jsme se rozhodli použít knihovnu scikit-learn [13], protože je poměrně rozšířená, obsahuje dobrou dokumentaci a obsahuje všechny metody strojového učení, které jsme potřebovali.

Matplotlib je knihovna pro kreslení grafů. Nejdříve jsme zkoušeli knihovnu Plotly, ale zde jsme nebyli spokojeni s dokumentací, tak jsme přešli na knihovnu Matplotlib [14] a zde již bylo vše bez problému.

Software je implementován pomocí sady skriptů, které na sebe navazují. Rozhodli jsme se to udělat tímto způsobem, protože jednotlivé skripty mají jasnou funkci a zároveň pokud máme správné vstupní soubory, tak se dají používat nezávisle na sobě. Navíc se tímto způsobem velice snadno přidávají nové modely.

Na začátku každého skriptu je popis, co daný skript dělá. Na to navazuje funkce *__main()*. Poté následují funkce v pořadí, ve kterém se volají. Pokud má daná funkce parametry, tak je u nich uveden jejich typ. Stejně tak pokud funkce vrací nějakou hodnotu, tak je u funkce uveden typ návratové hodnoty. V pojmenování funkcí, tříd a proměnných se držíme PEP8.

Skripty se dají rozdělit do 4 kategorií:

- skripty týkající se benchmarkovacího programu
- skripty týkající se skupin
- 2 skripty pro velmi používané funkce
- skripty pro kreslení grafů

V následujících sekcích popíšeme jednotlivé kategorie.

4.1 Benchmarkovací program

Celý postup benchmarkovacího programu je znázorněn na Obrázku 4.1. Výpočet začíná ve skriptu *extract_fragments.py*, který pro zadané soubory s molekulami a pro zadaný molekulový otisk extrahuje pro každou molekulu její fragmenty a indexy.

Na to navazuje skript *compute_descriptors.py*, který si bere výstupní soubory z *extract_fragments.py*. Tento skript počítá deskriptory, buď pro molekuly, nebo pro jejich fragmenty.

Dále máme skript *model_interface.py*, který obsahuje třídu IModel. Tato třída je následně děděna v každé implementaci modelu. Obsahuje 4 funkce:

1. *name()* - vrací jméno modelu. Abychom mohli podle zadaného jména spustit správný model.
2. *create_model(active_fragments: str, inactive_fragments: str, active_descriptors: str, inactive_descriptors: str, model_configuration: dict)* - vytvoří model.
3. *save_to_json_file(output_file: str, model: dict)* - uloží model do json souboru
4. *score_model(model_configuration: dict, fragments_file: str, descriptors_file: str, output_file: str)* - přečte si zadaný model a vypočítá skóre testovacích molekul

Tvorba modelu a výpočet skóre testovacích molekul

Zde jsme použili návrhový vzor factory. Každý skript, který implementuje model, dědí třídu z *model_interface.py*. Pro spuštění tvorby modelu se používá skript *create_model.py*, který použije skript *model_factory.py* pro vyhledání a vrácení správného modelu. Vyhledávání modelu probíhá na základě jména modelu. Poté, co nám *model_factory.py* vrátí *model*, tak v *create_model.py* můžeme spustit *model.create_model* a následně *model.save_to_json_file* funkce, které jsme pro daný *model* implementovali.

Daný *model* poté použijeme ve skriptu *score_molecules.py*, který vypočítá skóre testovacích molekul. Podle vstupního souboru s *modelem* pozná, o jaký *model* se jedná a přes *model_factory.py* ho získá. Poté zavolá funkci *model.score_model* pro zadaný *model* a výsledek uloží do souboru.

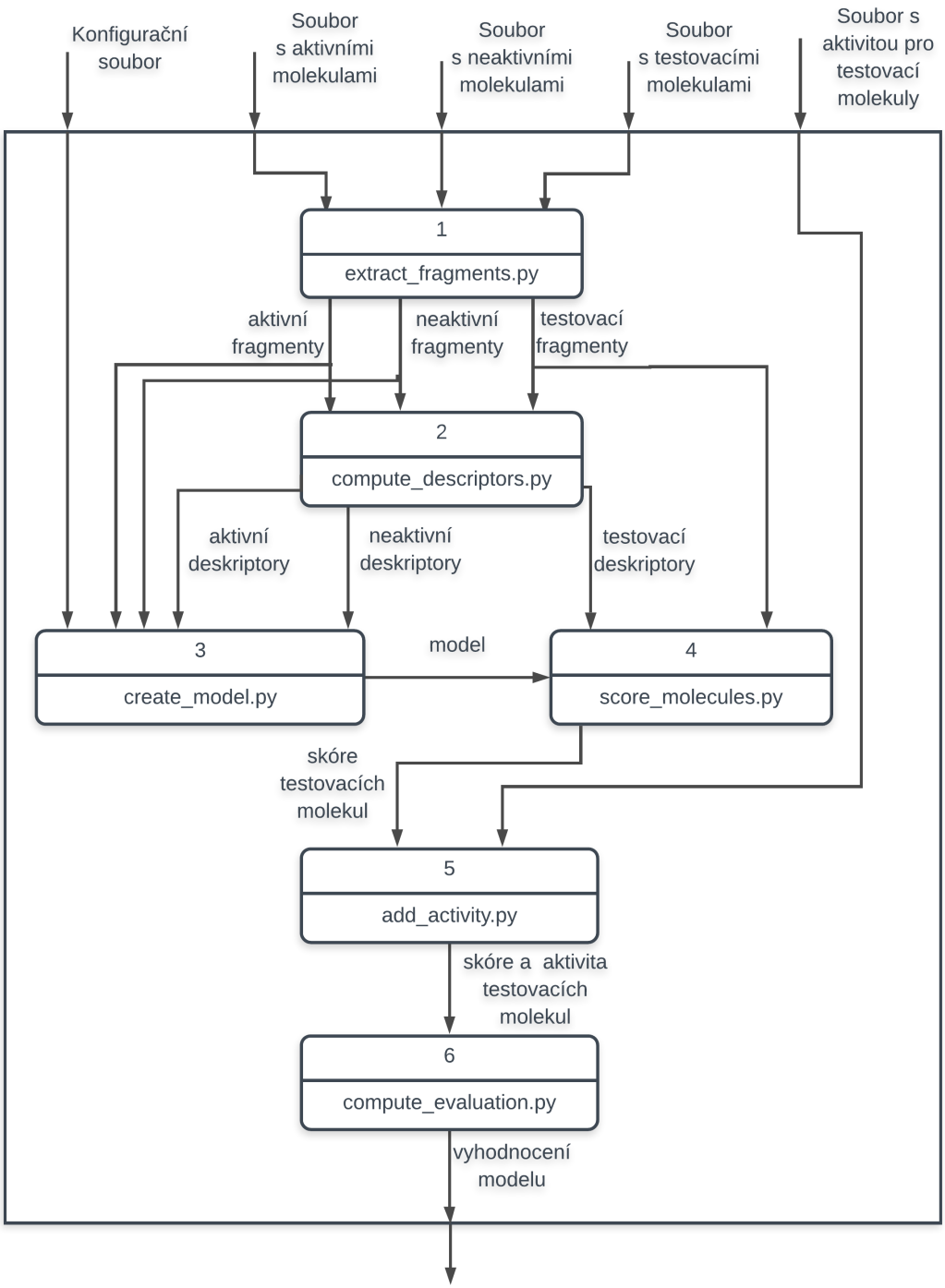
Tímto způsobem se dá přidávat libovolné množství nových modelů s minimem změn. Vždy stačí vytvořit skript, který obsahuje třídu, která dědí z *model_interface.py*, funkce třídy naimplementovat, zaregistrovat model pomocí funkce *register_model* z *model_factory.py* a skript nového modelu poté přidat do *model_factory.py*, konkrétně do funkce *_discover_models()*. Vše ostatní zůstává stejné. Námi vytvořené modely se nacházejí ve složce *model*.

Nakonec nám zbývá vyhodnocení modelu a tam musíme udělat 2 věci:

1. přidat skutečnou aktivitu k testovacím molekulám
2. spočítat vyhodnocení modelu

První část dělá skript *add_activity.py*, který k molekulám se skóre z *score_molecules.py* přidá jejich aktivitu. Druhou část dělá skript *compute_evaluation.py*, který dostane soubor ze skriptu *add_activity.py* a spočítá vyhodnocení modelu.

Jelikož budeme většinou spouštět celý výše uvedený proces najednou a budeme ho spouštět mnohokrát, tak pokaždé spouštět postupně 6 skriptů by nebylo uživatelsky přívětivé, tak jsme napsali skript *run_all_scripts.py*, který tyto skripty spustí za nás. *run_all_scripts.py* je konfigurovatelný pomocí souboru.



Obrázek 4.1: Průběh benchmarkovacího programu.

4.2 Skupiny

Skript *run_all_pair.py* z indexů molekulárního otisku aktivních molekul vytvoří skupiny 2. iterace, přidá k nim jméno modelu a vytvoří konfigurace modelů. Tyto modely poté spustí a vyhodnotí za pomoci již nám známých skriptů: *create_model.py*, *score_molecules.py*, *add_activity.py*, *compute_evaluation.py*. Jelikož modelů bude velké množství, tak bychom chtěli urychlit výpočet tím, že tento proces spustíme na více jádrech. Tedy podle počtu jader, které uživatel zadá, že chce použít pro výpočet, rozdělíme konfigurace modelů postupně do zhruba stejně velkých souborů. Poté každé jádro dostane svůj soubor, který postupně čte a vytváří modely, které následně vyhodnotí. Pokud by došlo k přerušení výpočtu, tak se celý proces nebude počítat znova, protože při dalším spuštění téhož se přeskočí již vyhodnocené skupiny 2. iterace a tedy výpočet pokračuje tam, kde přestal. Přeskočení již vyhodnocených skupin 2. iterace je zaručeno tím, že pro každý model předem víme jeho výstupní soubor a tedy vždy před daným výpočtem modelu stačí zkontrolovat, jestli daný výstupní soubor již existuje či ne. Jména výstupních souborů známe předem, jelikož jim přiřazujeme čísla od 0 do n - 1, kde n je počet skupin 2. iterace.

Dále chceme vybrat pouze ty skupiny 2. iterace, které mají lepší vyhodnocení než základní model. Problém je v tom, že soubor s vyhodnocením neobsahuje skupinu 2. iterace, pro kterou se toto vyhodnocení spočítalo. Skript *pair_analysis.py* vezme adresář, kde se vyskytuje vyhodnocení všech skupin 2. iterace. Každý soubor v tomto adresáři obsahuje číslo, to udává pořadí, ve kterém byla určitá skupina 2. iterace vygenerována. Tedy nyní generujeme skupiny 2. iterace stejným způsobem jako v *run_all_pair.py* a podle čísla v názvu souboru poznáme, jaké vyhodnocení patří k jaké skupině 2. iterace. Poté jednotlivé vyhodnocení skupin 2. iterace porovnáme s vyhodnocením základního modelu a skupiny 2. iterace, které mají lepší výsledky, vypíše do souborů podle toho v jakých aspektech (AUC, EF 1%, EF 5%) měly lepší výsledky.

Skript *select_best_results.py* setřídí na základě kritéria výsledky z jednoho výstupního souboru z *pair_analysis.py*. Poté vypíše n (zadáno uživatelem) nejlepších výsledků do výstupního souboru.

Dále následuje skript *run_all_groups.py*, který bere skupiny indexů z *pair_analysis.py* nebo *group_analysis.py* nebo *select_best_results.py* a skládá je dohromady způsobem každý s každým. Zároveň kontroluje, jestli již daná skupina nebyla vygenerována. Daná skupina totiž může být vygenerována více způsoby, např. ze skupin [a,b] a [b,c] dostaneme [a,b,c] a ze skupin [a,c] a [b,c] dostaneme [a,c,b]. To jsou ale stejné skupiny, protože na pořadí prvků v jednotlivých skupinách nezáleží. Každou skupinu si ponechá pouze jednou a vytvořené skupiny pouští podobně jako *run_all_pair.py*. Změna je v tom, že pro každou skupinu se do konfigurace modelů přidá název výstupního souboru. Název výstupního souboru je pro každou skupinu unikátní, což umožňuje, že pokud bychom nejprve spouštěli celý proces pro nejlepších 50 výsledků a poté ten samý proces pro 100 nejlepších výsledků, tak se již vyhodnocené skupiny znova vyhodnocovat nebudou. Unikátnost bylo poměrně složité vyřešit navíc v kombinaci s možností přerušení výpočtu programu. Vyřešili jsme to tak, že nejprve zkontrolujeme, jestli již nějaké konfigurační soubory pro dané skupiny i-té iterace existují. Zde poté následují 3 možnosti:

1. Pokud neexistují, tak víme, že žádný výpočet ještě neproběhl a podle počtu jader n, na kterých má běžet výpočet vytvoříme konfigurační soubory se jmény (nechť $m = n$ - 1): *configuration1_0.json*, *configuration1_1.json*,..., *configuration1_m.json*, kde jednička v názvu znamená, že tento výpočet probíhá poprvé, tomuto číslu budeme říkat *první číslo*. Stejným způsobem jsou i přiřazována jména k výstupním souborům, kde máme *evaluation1_z.json*, kde z je pořadí, ve kterém byla daná skupina vygenerována.

2. Pokud již konfigurační soubory existují, ale počet modelů v konfiguračních souborech je různý od počtu již vyhodnocených skupin, tak si vezmeme konfigurační soubory, jejichž *první číslo* je nejvyšší a pokračujeme ve výpočtu.

3. Poslední možností je, že již máme konfigurační soubory a počet modelů v konfiguračních souborech se rovná počtu souborů s vyhodnocením. V tomto případě víme, že chceme provádět ten samý výpočet ale pro větší množství skupin. Tedy z již existujících konfiguračních souborů si vezmeme skupiny a dáme je do množiny skupin. Poté ze zadaného souboru postupně generujeme skupiny a pro každou skupinu se podíváme, jestli již je v množině skupin. Pokud v ní není, tak ji tam přidáme. Pokud v ní je, tak ji nepřidáme. Dále odstraníme z množiny skupin již vyhodnocené skupiny. Zbylé skupiny z množiny skupin rozdělíme do konfiguračních souborů, ve kterých bude *první číslo* o 1 větší než bylo dosud nejvyšší.

Skript *group_analysis.py* si ze vstupního souboru vezme všechny hodnoty AUC, EF 1%, EF 5% a zprůměruje je. Pomocí konfiguračních souborů, kde je u každé skupiny uvedeno jméno souboru s vyhodnocením, zjistí, jaké vyhodnocení patří k dané skupině. Poté hodnoty vyhodnocení porovná se zprůměrovanými hodnotami AUC, EF 1%, EF 5% a skupiny roztřídí stejně jako v *pair_analysis.py*.

4.3 Velmi používané funkce

Většina skriptů vypisuje svoje výstupy do souborů json, tedy jsme udělali knihovnu *inputoutput_utils.py*, která toto provádí.

Modely, na jejichž výpočet fragmentů a indexů jsme použili přímo funkci z knihovny RDKit (modely ze složky *configuration_files*, které mají v názvu *rdkit*), tak tyto modely mají podobnou implementaci, a tak jsme vytvořili knihovnu *rdkitmodel_utils.py*, která funkce pro tvorbu modelů obsahuje.

4.4 Kreslení grafů

Skripty obsahující v názvu *graph* nebo *graphs* vykreslují různé obrázky a grafy. *active_inactive_graph.py* vykreslí Obrázek 1.7. *print_evaluation_graphs.py* a *print_graph.py* vykreslují porovnání skupin 1. iterace. *print_all_analyses_graphs.py* a *print_group_graphs.py* vykreslují porovnání skupin. *graph_of_pairs.py* vykreslí histogram, kde je znázorněno, kolik skupin dosáhlo určitého výsledku. Ve všech těchto skriptech jsme použili knihovnu Matplotlib.

5. Experimenty

Experimenty jsou realizované různými modely. Všechny vytvořené modely si v této kapitole popíšeme. Řekneme, proč jsme je vytvořili a u většiny z nich ukážeme výpočet podobnosti. Pro názorné ukázky budeme používat molekuly, které se vyskytují v naší sadě aktivních molekul a sadě testovacích molekul.

Postup výpočtu podobnosti molekul je stejný pro všechny molekulové otisky. Pro názorné ukázky jsme si vybrali indexy vygenerované pomocí ECFP2, protože v této práci se zabýváme hlavně ECFP a pro ECFP2 dostaneme menší množství indexů než pro větší průměry, tedy názorné ukázky budou přehlednější.

Pokud budeme obecně mluvit o aktivních molekulách, budeme je označovat jako A a testovací molekuly budeme obecně označovat jako Q. Podobnost molekul A_1 a Q_1 budeme značit jako A_1Q_1, stejně tak i A_1Q_2 pro molekuly A_1 a Q_2. Skóre molekuly Q_j budeme značit jako AQ_j, kde A označuje všechny molekuly ze sady aktivních molekul. $|M|$ bude označovat počet indexů, které nejsou ve skupinách, plus počet skupin v molekule M. Jako $|AQ|$ budeme značit počet stejných indexů a stejných skupin molekul A a Q. Pokud budeme dávat všechny aktivní molekuly dohromady, budeme je značit A_L.

Každý model jsme naimplementovali v našem programu. Jména skriptů pro všechny modely jsou uvedeny v příloze A.2. Modely si rozdělíme do 3 skupin:

1. všechny indexy samostatně
2. hashování
3. indexy do skupin

V následujících sekcích si jednotlivé skupiny popíšeme.

5.1 Všechny indexy samostatně

Tato skupina modelů odpovídá experimentu uvedenému v podsekci 2.1.1. Indexy aktivní molekuly A_1:

```
[3624155, 784740959, 847867887, 864674487, 951226070, 951226070,
994485099, 2076190208, 2132511834, 2245384272, 2245384272, 2246728737,
2246728737,2246728737, 2297887526, 2720313463, 2752034647, 3106190496,
3162837314, 3217380708, 3217380708, 3217380708, 3217380708, 3218693969,
3218693969,3218693969, 3218693969, 3824050433, 3824050433, 3975275337,
3983062349, 4089138501]
```

Indexy neaktivní molekuly Q_1 z testovací sady:

```
[3624155, 207661762, 1173346717, 1521457891, 2130998634, 2132511834,
2245384272, 2245384272, 2245897107, 2245900962, 2297887526, 2581192669,
3217380708, 3218693969, 3218693969, 3492293582, 3585958780, 3633369638]
```

Indexy aktivní molekuly Q_2 z testovací sady:

```
[3624155, 98513984, 98513984, 568857350, 847694221, 951226070,
951226070, 2132511834, 2245384272, 2245384272, 2297887526, 2583199761,
2720313463, 2752034647, 3217380708, 3217380708, 3217380708,
3218693969, 3218693969, 3218693969, 3218693969, 3218693969,
3983062349, 4089138501]
```

Nyní se zde nevyskytují žádné skupiny (všechny indexy jsou samostatně), tedy:
$|A_1| = 32$
$|Q_1| = 18$
$|Q_2| = 24$

Jelikož jedna molekula může mít stejný index vícekrát např. A_1 má index 3217380708 celkem 4x, ale Q_2 jenom 3x, tak tento index do počtu stejných indexů přičte 3. Tedy z molekul vidíme, že:
$|A_1Q_1| = 8$
$|A_1Q_2| = 18$

Ve většině molekulových sad se nachází více aktivních molekul než 1 (zde pouze pro ukázku výpočtu podobností máme jednu). S více molekulami budeme zacházet dvěma způsoby:

1. max-fusion - sekce 1.4.
2. dáme indexy všech aktivních molekul dohromady a poté každou testovací molekulu porovnáváme s těmito indexy.

Pokud u modelů explicitně neřekneme, že budeme dávat indexy aktivních molekul dohromady (přístup 2), tak se jedná o přístup 1.

Většina modelů používá výpočet podobnosti, který vychází z Tanimotova podobnostního koeficientu. Rozdíl je v tom, že Tanimotův podobnostní koeficient používá bitové vektory a tedy vícenásobné indexy nezapočítává, zatímco my s nimi budeme pracovat. Tedy výpočet podobnosti bude mít následující vzorec:

$$A_iQ_j = \frac{|A_iQ_j|}{|A_i| + |Q_j| - |A_iQ_j|} \tag{5.1}$$

Do této skupiny modelů patří následující modely.

5.1.1 Základní indexový model

Tento model vypočítá výsledky zadaného molekulárního otisku. Vezmeme indexy aktivních molekul a testovací molekuly a spočítáme skóre:

$$AQ_j = \max_{A_i \in A} \frac{|A_iQ_j|}{|A_i| + |Q_j| - |A_iQ_j|} \tag{5.2}$$

Pro příklad:
A_1Q_1: $|A_1| = 32$, $|Q_1| = 18$, $|A_1Q_1| = 8$. Číselně:

$$A_1Q_1 = \frac{8}{32 + 18 - 8} \doteq 0.19 \tag{5.3}$$

A_1Q_2: $|A_1| = 32$, $|Q_2| = 24$, $|A_1Q_2| = 18$. Číselně:

$$A_1Q_2 = \frac{18}{32 + 24 - 18} \doteq 0.4737 \tag{5.4}$$

5.1.2 Lineární regrese

Výpočet se zakládá na deskriptorech. Při lineární regresi využijeme při tvorbě modelu i sadu neaktivních molekul, protože kdybychom použili pouze sadu aktivních molekul, tak by lineární regrese určila všechny testovací molekuly za aktivní.

5.1.3 Rozhodovací stromy

Stejně jako lineární regrese je založen na deskriptorech a při tvorbě modelu používá i sadu neaktivních molekul.

5.1.4 Deskriptor model

Model, který využívá pro výpočet podobností deskriptory. Tento model dává všechny deskriptory aktivních molekul dohromady A_D. Deskriptory molekuly Q si označme Q_D. Potom vzorec výpočtu skóre vypadá takto:

$$AQ_j = \frac{k_1 * (|A_D \cap Q_{jD}|) - k_2 * (|Q_{jD}| - |A_D \cap Q_{jD}|)}{|Q_{jD}|} \tag{5.5}$$

kde k_1 a k_2 jsou koeficienty, které můžeme libovolně nastavovat. Takto se dají testovat různé kombinace koeficientů např. to, jestli budeme dávat velký důraz na nalezení aktivních deskriptorů a k_1 bude mít velkou hodnotu v porovnání s k_2, nebo budeme "penalizovat" molekuly, které nebudou mít určitý deskriptor a nastavíme velké k_2 v porovnání s k_1.

Pro náš příklad je A_D rovno A_1, protože máme pouze jednu aktivní molekulu. Pro $k_1 = 1$ a $k_2 = 1$ podobnosti vychází následovně:

$$A_1Q_1 \doteq 0.111 \tag{5.6}$$

$$A_1Q_2 = 0.75 \tag{5.7}$$

5.1.5 Model tvořený všemi aktivními indexy

Tento model funguje stejně jako předchozí 5.1.4 až na malou změnu, že nepočítá s deskriptory, ale s indexy. Vzorec:

$$AQ_j = \frac{k_1 * (|A_L \cap Q_j|) - k_2 * (|Q_j| - |A_L \cap Q_j|)}{|Q_j|} \tag{5.8}$$

Řekněme, že $k_1 = 1$ a $k_2 = 1$:

A_1Q_1: $|A_1| = 32$, $|Q_1| = 18$, počet indexů v Q_1 vyskytujících se i v A_1 je 8, počet rozdílných indexů je tedy 10 (počet indexů v Q_1 mínus počet stejných indexů). Tedy celková suma je -2 a tu podělíme počtem indexů v Q_1. Celkem:

$$A_1Q_1 = \frac{-2}{18} \doteq -0.1111 \tag{5.9}$$

A_1Q_2: $|A_1| = 32$, $|Q_2| = 24$, počet stejných indexů je 19 (zde je to 19 a ne 18, protože nám stačí, když se určitý index objeví v aktivní molekule, ale nepočítáme počty stejných výskytů). Počet rozdílných je 5, tedy celkem:

$$A_1Q_2 = \frac{14}{24} \doteq 0.5833 \tag{5.10}$$

5.2 Hashování

Tento model jsme vytvořili pro experiment uvedený v podsekci 2.1.2, jelikož je naším hlavním cílem práce seskupování indexů do skupin a hashování se dá považovat za vytváření skupin. Tomuto popisu odpovídá následující model.

5.2.1 Hashovací model

Řekněme, že budeme hashovat číslem 1025, potom molekula A_1 má následující indexy:

```
[780, 959, 187, 887, 445, 445, 374, 433, 359, 822, 822, 487, 487,
487, 501, 363, 872, 771, 964, 8, 8, 8, 8, 244, 244, 244, 244, 933,
933, 412, 499, 426]
```

Indexy Q_1 jsou:

```
[780, 862, 517, 166, 59, 359, 822, 822, 132, 912, 501, 769, 8, 244,
244, 707, 380, 888]
```

A indexy Q_2 jsou:

```
[780, 209, 209, 800, 771, 445, 445, 359, 822, 822, 501, 911, 363,
872, 8, 8, 8, 244, 244, 244, 244, 244, 499, 426]
```

Vzorec pro výpočet skóre je shodný s vzorcem 5.2.
Pro příklad:
A_1Q_1: $|A_1| = 32$, $|Q_1| = 18$, $|A_1Q_1| = 8$, tedy celkově:

$$A_1Q_1 = \frac{8}{32 + 18 - 8} \doteq 0.19. \tag{5.11}$$

A_1Q_2: $|A_1| = 32$, $|Q_2| = 24$, $|A_1Q_2| = 19$, celkově:

$$A_1Q_2 = \frac{19}{32 + 24 - 19} \doteq 0.5135 \tag{5.12}$$

Stejné modely jsme udělali i pro AP, FCFP a TT.

5.3 Indexy do skupin

Tato skupina modelů odpovídá experimentům uvedených v podsekcích 2.1.3, 2.1.4, 2.1.5. Jak jsme psali v podsekci 2.1.3, tak při seskupování indexů do skupin z vícenásobných výskytů indexů ponecháme pouze jeden výskyt. Tedy indexy A_1 jsou nyní:

```
[3624155, 784740959, 847867887, 864674487, 951226070, 994485099,
2076190208, 2132511834, 2245384272, 2246728737, 2297887526, 2720313463,
2752034647, 3106190496, 3162837314, 3217380708, 3218693969, 3824050433,
3975275337, 3983062349, 4089138501]
```

a $|A_1| = 21$. Indexy pro Q_1:

```
[3624155, 207661762, 1173346717, 1521457891, 2130998634, 2132511834,
2245384272, 2245897107, 2245900962, 2297887526, 2581192669,
3217380708, 3218693969, 3492293582, 3585958780, 3633369638]
```

a $|Q_1| = 16$. Indexy pro Q_2:

```
[3624155, 98513984, 568857350, 847694221, 951226070, 2132511834,
2245384272, 2297887526, 2583199761, 2720313463, 2752034647,
3217380708, 3218693969, 3983062349, 4089138501]
```

a $|Q_2| = 15$.
$|A_1Q_1| = 6$
$|A_1Q_2| = 11$

Začneme skládat indexy do skupin 2. iterace, tedy např. z indexů 3624155 a 784740959 bychom udělali [3624155, 784740595]. Tyto modely obecně fungují pro jakékoliv množství skupin, kde každá skupina má jakékoliv množství indexů (ovšem každý index se vyskytuje v maximálně jedné skupině). My si jejich fungování a výpočet ukážeme na skupinách 2. iterace, protože se jedná o první krok při dávání indexů do skupin. Pro každý model si vybereme takovou skupinu 2. iterace, na které se dobře ukáže, co zadaný model dělá. Skupiny 2. iterace vytváříme z indexů aktivních molekul pro daný molekulární otisk.

Tyto modely používají 3 vzorce výpočtů.

První je shodný se vzorcem 5.2. Do této skupiny patří modely z podsekcí: 5.3.1, 5.3.2, 5.3.3.

Druhý je založen na ořezávání. To znamená, že si vezmeme všechny indexy aktivních molekul dohromady a ponecháme si pouze ty indexy, které se vyskytují ve stejném nebo vyšším počtu aktivních molekul než je oříznutí. Řekněme, že máme 5 aktivních molekul a oříznutí je 60%, pak ponecháme pouze takové indexy, které se vyskytují ve 3 a více molekulách. Vzorec (velikost oříznutí značme o a počet aktivních molekul jako A_P):

$$AQ_j = \frac{|A_qQ_j|}{|A_q| + |Q_j| - |A_qQ_j|}, \text{kde: } \forall a_i \in A_L \wedge \frac{|a_i|}{|A_P|} \geq o => a_i \in A_q \tag{5.13}$$

Do této skupiny výpočtů patří modely z podsekcí: 5.3.4, 5.3.5.

Do poslední skupiny patří model z podsekce 5.3.6. Vzorec pro výpočet je následující:

$$AQ_j = \max_{A_i \in A} \frac{|A_iQ_j|}{S * (|A_i| + |Q_j| - |A_iQ_j|)} \tag{5.14}$$

kde S je počet indexů ve skupinách (pokud se jedná o 1. iteraci, tak $S = 1$).

5.3.1 Aktivní skupiny

Pokud to jde, přidáme molekule zadanou skupinu 2. iterace. Pro skupinu 2. iterace [a, b] by to bylo následovně: [a, b, c] -> [[a, b], c].

Příklad: Mějme skupinu 2. iterace [3624155, 408913501]. Tato skupina 2. iterace se vyskytuje v A_1 a v Q_2. Jelikož jsme v A_1 vytvořili skupinu 2. iterace, tak se $|A_1|$ zmenší o 1.

A_1Q_1: $|A_1| = 20$, $|Q_1| = 16$ (stejné, protože jsme skupinu 2. iterace nevytvořili), $|A_1Q_1| = 5$, protože máme skupinu 2. iterace [3624155, 408913501], ale v Q_1 je 3624155 samostatně, tudíž se již nerovnají. Celkově:

$$A_1Q_1 = \frac{5}{20 + 16 - 5} \doteq 0.1613 \tag{5.15}$$

A_1Q_2: $|A_1| = 20$, $|Q_2| = 14$ (vytvořili jsme skupinu 2. iterace), $|A_1Q_2| = 10$ (z 2 stejných indexů jsme vytvořili skupinu 2. iterace - celková podobnost se zmenšila o 1), celkově:

$$A_1Q_2 = \frac{10}{20 + 14 - 10} \doteq 0.41667 \tag{5.16}$$

5.3.2 Přidej index

Mějme skupinu 2. iterace [*a*, *b*] a v naší molekule *M* by se vyskytovalo pouze *a*, pak tento model přidá index *b* do molekuly. Tedy: [*a*, *c*, *d*,...] -> [*a*, *b*, *c*, *d*,...]. Mějme např. skupinu 2. iterace [864674487, 4089138501], potom jsou podobnosti následující:

A_1Q_1: $|A_1| = 21$, protože oba indexy již A_1 obsahuje, tudíž nic nepřidáváme. $|Q_1| = 16$, ani jeden index ze skupiny 2. iterace se nevyskytuje v Q_1, tedy se nic nemění. Celkově:

$$A_1Q_1 = \frac{6}{21 + 16 - 6} \doteq 0.1935 \tag{5.17}$$

A_1Q_2: $|A_1| = 21$; $|Q_2| = 16$, protože v Q_2 se vyskytuje index 4089138501, ale ne index 864674487, tedy index 864674487 přidáme, tudíž se zvýší počet indexů. $|A_1Q_2| = 12$, protože jsme přidali jeden index. Celkově:

$$A_1Q_2 = \frac{12}{21 + 16 - 12} = 0.48 \tag{5.18}$$

5.3.3 Odstraň stejné indexy

Mějme skupinu [*a*, *b*, *c*] a v molekule *M* se vyskytuje *a*, *b* i *c*, pak *b* i *c* odstraníme z indexů molekuly. Pokud by molekula měla pouze *b*, tak bychom *b* nahradili *a*. Mějme skupinu 2. iterace [3624155, 4089138501], potom jsou podobnosti:

A_1Q_1: $|A_1| = 20$ (odstranili jsme index 4089138501), $|Q_1| = 16$, $|A_1Q_1| = 6$, celkově:

$$A_1Q_1 = \frac{6}{20 + 16 - 6} = 0.2 \tag{5.19}$$

A_1Q_2: $|A_1| = 20$, $|Q_2| = 14$ (odstranili jsme index 4089138501), $|A_1Q_2| = 10$, celkově:

$$A_1Q_2 = \frac{10}{20 + 14 - 10} \doteq 0.41667 \tag{5.20}$$

5.3.4 Ořízni indexy

Model, který vezme indexy aktivních molekul dohromady, provede oříznutí a následně každý index, který prošel oříznutím, ponechá pouze jednou. Nazvěme tuto množinu indexů jako množinu oříznutých indexů. Můžeme se na tuto množinu oříznutých indexů dívat jako na indexy jedné aktivní molekulu. V tomto

modelu výpočet podobnosti probíhá stejně jako v podsekci 5.3.1, akorát místo jednotlivých indexů aktivních molekul, které jsou navíc rozdělené, máme množinu oříznutých indexů. Tedy na každé molekule z testovací sady a množině oříznutých indexů vytvoříme danou skupinu a poté vypočítáme podobnost testovací molekuly. Jelikož v našem příkladě máme pouze jednu aktivní molekulu, tak oříznutí nebude mít vliv. Tudíž množina oříznutých indexů bude přesně odpovídat indexům aktivní molekuly A_1 a tedy by podobnosti pro stejnou skupinu, jakou jsme použili v podsekci 5.3.1, vyšly stejně jako v podsekci 5.3.1.

5.3.5 Ořízni a přidej indexy

Výpočet modelu probíhá následovně: máme zadanou skupinu 2. iterace a procházíme aktivní molekuly popořadě a započítáváme četnosti jednotlivých indexů. Když narazíme na aktivní molekulu, která má pouze jeden index zadané skupiny 2. iterace, tak připočítáme četnost i druhému indexu ze skupiny 2. iterace. Poté provedeme oříznutí a ponecháme si od každého indexu, který prošel oříznutím pouze jeden výskyt. Nazvěme si opět tyto indexy jako množinu oříznutých indexů. Na tuto množinu se opět můžeme dívat jako na indexy jedné aktivní molekuly. Výpočet podobností zde probíhá stejně jako v podsekci 5.3.2 akorát místo jednotlivých indexů aktivních molekul máme množinu oříznutých indexů. Ze stejných důvodů jako v podsekci 5.3.4 by podobnosti molekul pro stejnou skupinu, jakou jsme použili podsekci 5.3.2, vyšly stejně jako v podsekci 5.3.2.

5.3.6 Velikost skupiny

Pokud to jde, tak molekule přidáme zadanou skupinu 2. iterace: pro skupinu 2. iterace [a, b]: [a, b, c] -> [[a, b], c]. Poté každou testovací molekulu porovnáváme se všemi aktivními molekulami, a pokud má daná aktivní molekula skupinu 2. iterace a testovací molekula má pouze 1 index z skupiny 2. iterace, tak do podobnosti nezapočítáme +1, ale +2. Celkovou podobnost poté vydělíme počtem indexů ve skupinách, což je tedy pro skupinu 2. iterace dva, abychom se dostali do rozmezí [0, 1] (pokud nemáme žádné skupiny, tak dělíme jedničkou).

Vezměme si skupinu 2. iterace [864674487, 4089138501] a spočítejme podobnosti:

A_1Q_1: $|A_1| = 20$ (vznikla 1 skupina 2. iterace), $|Q_1| = 16$, $|A_1Q_1| = 6$ (nic se nezměnilo). Celkově:

$$A_1Q_1 = \frac{6}{2 * (20 + 16 - 6)} = 0.1 \tag{5.21}$$

A_1Q_2: $|A_1| = 20$, $|Q_2| = 15$, $|A_1Q_2| = 12$ (11 shod + obsahuje 1 člen skupiny 2. iterace), celkem:

$$A_1Q_2 = \frac{12}{2 * (20 + 15 - 12)} \doteq 0.26087 \tag{5.22}$$

6. Výsledky

Pro získání výsledků jsme použili námi implementovaný software. Výsledky jsme rozřadili do 5 kategorií podle experimentů, které jsme uvedli v sekci 2.1.

6.1 Počáteční porovnání

Jak jsme uvedli v podsekci 2.1.1, tak první experiment se zabývá porovnáním mezi běžně používanými metodami, strojovým učením a námi definovanými přístupy. Výsledky jsou vidět v Tabulce 6.1 (AUC jsme zaokrouhlovali na 3 desetinná místa).

Model	**Typ**	**AUC**	**EF 1%**	**EF 5%**
ECFP	ECFP4	0.927	63.7	17
ECFP	ECFP6	0.925	63.7	17
FCFP	FCFP4	0.92	53.9	15
FCFP	FCFP6	0.926	58.8	16
AP	AP	0.913	63.7	16
TT	TT4	0.914	58.8	16
rozhodovací strom	ECFP4	0.962	39.2	8
lineární regrese	ECFP4	0.819	4.9	6
via	ECFP4	0.93	58.8	17
deskriptor model	AP	0.891	83.8	17
aktivní skupiny	ECFP4	0.928	68.6	17
přidej indexy	ECFP4	0.928	68.6	17
odstraň stejné indexy	ECFP4	0.928	68.6	17
velikost skupiny	ECFP4	0.928	68.6	17
ořízni indexy_0	ECFP4	0.918	58.8	15
ořízni a přidej indexy_0	ECFP4	0.918	58.8	15
ořízni indexy_40	ECFP4	0.924	58.8	15
ořízni a přidej indexy_40	ECFP4	0.924	58.8	15
ořízni indexy_60	ECFP4	0.776	9.8	3
ořízni a přidej indexy_60	ECFP4	0.776	9.8	3

Tabulka 6.1: Porovnání běžně používaných metod, strojového učení a námi definovaných přístupů.

via je zkratka pro *model tvořený všemi aktivními indexy*. *model tvořený všemi aktivními indexy* a *deskriptor model* obsahují výsledky pro $k_1 = 1$ a $k_2 = 1$. Všechny uvedené výsledky skupinových modelů jsou pro 1. iteraci. U skupinových modelů, kde se dělá oříznutí je v názvu uvedené číslo, které v procentech uvádí velikost tohoto oříznutí. Názvem sloupce *Typ* myslíme typ molekulového otisku. V Tabulce 6.1 jsme uvedli pro nás nejzajímavější hodnoty. Všechny naměřené hodnoty jsou uvedeny v příloze A.3.

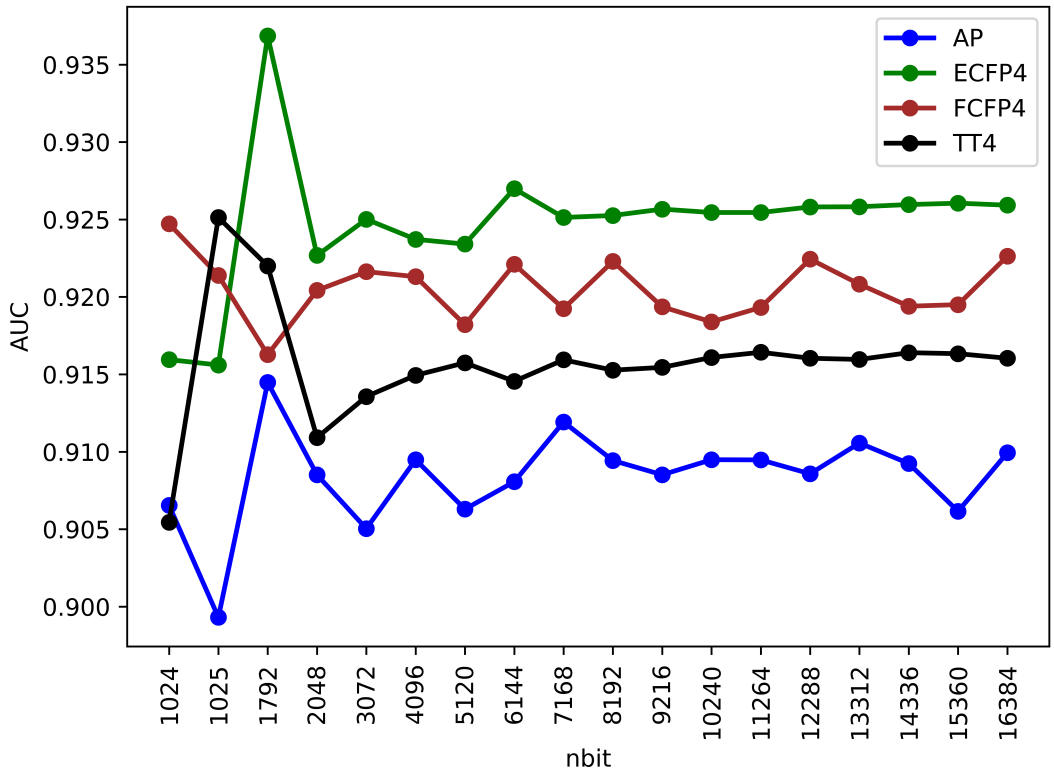
Z Tabulky 6.1 je vidět, že nejhorší výsledky mají *lineární regrese*, *ořízni indexy_60* a *ořízni a přidej indexy_60*. Zjevně se tedy lineární regrese pro tuto

úlohu nehodí a oříznutí 60% je již příliš veliké. Nejlépe, co se týče AUC, vychází *rozhodovací strom*. Ten ovšem má podprůměrné výsledky pro EF 1% a EF 5%. Nejlepšího výsledku pro EF 1% dosahuje *deskriptor model* pro molekulový otisk AP. Jenže tento model má poměrně špatné AUC. Ostatní modely se ve výsledcích dost podobají. Tedy se zde nevyskytuje žádný model, který by byl celkově lepší než ostatní.

Dále vidíme, že modely: *aktivní skupiny*, *přidej indexy*, *odstraň stejné indexy* a *velikost skupiny* mají stejné hodnoty. Toto dává smysl, jelikož všechny skupiny nechávají aktivní molekuly rozdělené, navíc první 3 skupiny mají stejný vzorec výpočtu a pro model *velikost skupiny* je ve vzorci $S = 1$, tedy vzorec je stejný s vzorcem prvních třech skupin. Žádné skupiny nevytváříme, tedy se počítá to samé. Stejně tak to platí i pro *ořízni indexy_0* a *ořízni a přidej indexy_0*; *ořízni indexy_40* a *ořízni a přidej indexy_40*; *ořízni indexy_60* a *ořízni a přidej indexy_60*.

6.2 Hashování

Pro získání výsledků hashování pro jednotlivé molekulové otisky a pro různé hodnoty hashování jsme spustili modely, které používají funkce, jež jsou implementované v knihovně RDKit a hashují jednotlivé molekulové otisky. Výsledky jsou vidět v Obrázku 6.1.



Obrázek 6.1: Porovnání hashování molekulových otisků. na x-ové ose je uvedeno číslo, kterým jsme hashovali.

Z Obrázku 6.1 je vidět, že pro určité hodnoty hashování jsme schopni dostat

lepší hodnotu AUC, než je hodnota stejného molekulového otisku uvedená v Tabulce 6.1. Např. pro AP je to hodnota 1792; pro ECFP4 jsou to hodnoty 1792 a 6144; pro FCFP4 je to více hodnot, nejlepší AUC je pro 1024; a pro TT4 je to také více hodnot, nejlepší AUC je pro 1025. Tedy je vidět, že pro každý molekulový otisk existují určité skupiny, díky kterým můžeme dostat lepší výsledek než je základní. Tím se ukazuje, že kolize mohou být prospěšné.

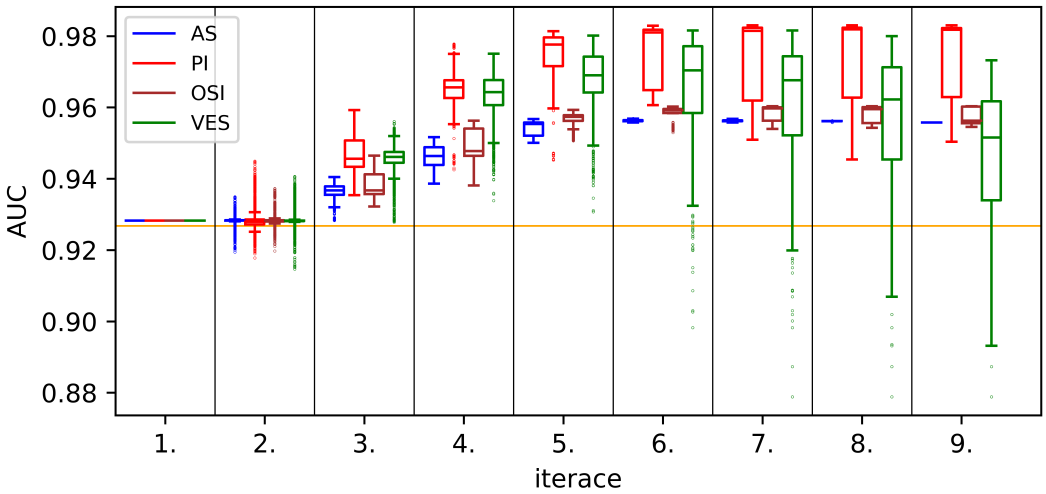
6.3 Vytváření skupin

Jak jsme zmínili v podsekci 2.1.3, tak jsme se zaměřovali na ECFP. My jsme si konkrétně vybrali ECFP4, protože jak můžeme vidět z Tabulky 6.1, tak dosáhl lepšího výsledku než ECFP6 a navíc pro ECFP4 dostaneme méně indexů, což je pro naše časově náročné výpočty výhodné. Jako základní hodnotu, podle které budeme vybírat výsledky pro skupiny všech provedených iterací, jsme brali hodnotu AUC, která vyšla pro ECFP4, což je tedy zhruba 0.927. Ze všech výsledných skupin určité iterace jsme si ponechali pouze takové skupiny, které dosáhly lepšího výsledku, než je základní hodnota. Z těchto skupin jsme poté vybrali 100 nejlepších (pokud jich alespoň 100 bylo) podle AUC, které jsme použili pro další iteraci. Pro každý model jsme provedli 9 iterací. Výpočet jsme provedli pro 8 modelů:

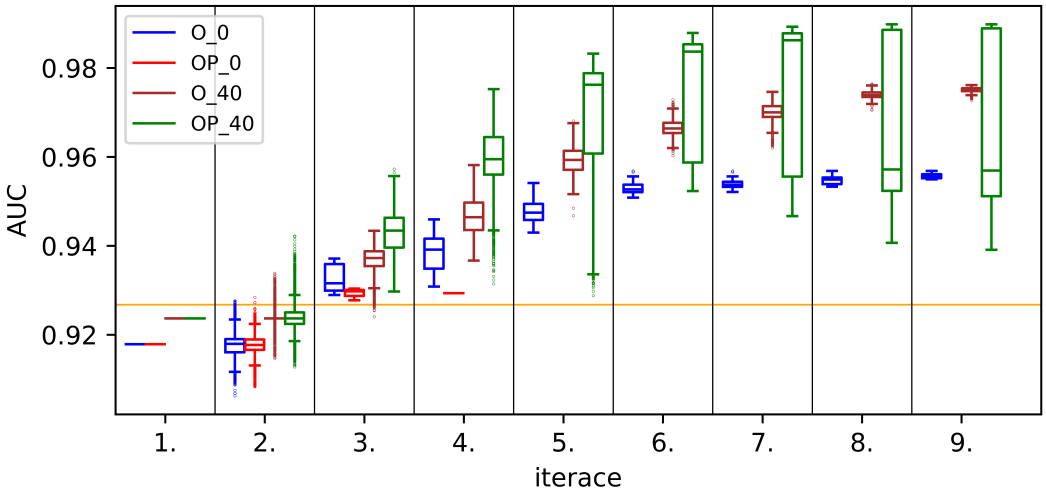
- Aktivní skupiny (AS)
- Přidej index (PI)
- Odstraň stejné indexy (OSI)
- Velikost skupiny (VES)
- Ořízni indexy pro oříznutí 0% (O_0)
- Ořízni indexy pro oříznutí 40% (O_40)
- Ořízni a přidej indexy pro oříznutí 0% (OP_0)
- Ořízni a přidej indexy pro oříznutí 40% (OP_40)

Nejprve jsme porovnali první 4 modely mezi sebou (nepoužívají oříznutí a nechávají aktivní molekuly rozdělené) - Obrázek 6.2, poté poslední 4 mezi sebou (používají oříznutí a dávají indexy aktivních molekul dohromady) - Obrázek 6.3 a nejlepší z těchto dvou skupin poté mezi sebou - Obrázek 6.4.

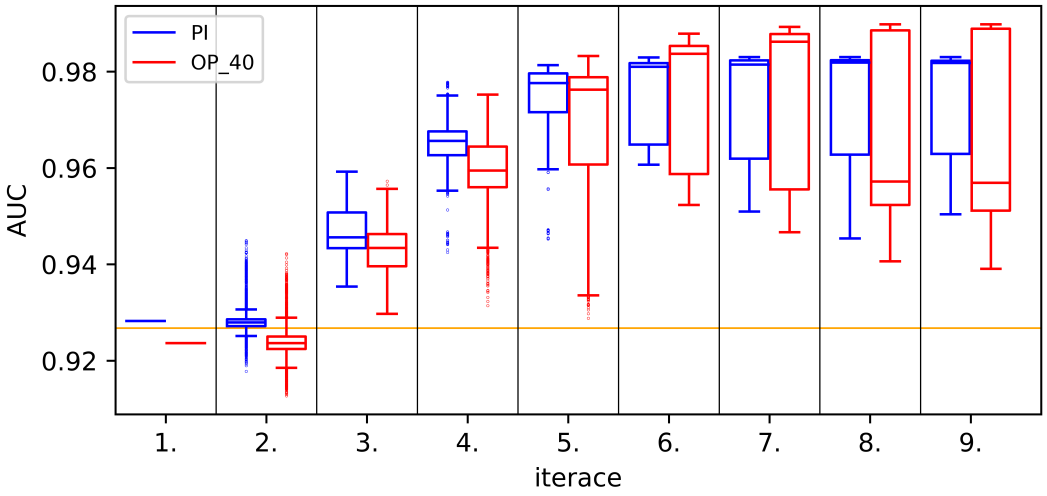
V Obrázku 6.2 je zajímavé pozorovat výsledky 4. iterace, kde všechny modely mají boxploty poměrně podobné, ale modely *PI* a *VES* mají průměrné hodnoty zhruba o 2 setiny lepší a mají také vzdálené hodnoty (outliers). Nakonec modely *PI* a *VES* dosahují vyšších průměrných i maximálních hodnot než modely *AS* a *OSI*. Model *VES* dosahuje nejlepších výsledků pro skupiny 6. iterace. Sice dosahuje i pro 7. iteraci nejvyšší hodnoty AUC, ale celkově pro 6. iteraci dává vyšší průměrné hodnoty. Podobně je na tom model *AS*, který dosáhl nejvyšší hodnoty AUC pro skupiny 6. iterace. Pro modely *PI* a *OSI* dostáváme nejvyšší AUC pro 7., 8. i 9. iteraci. Avšak pro model *PI* můžeme vidět, že pro 8. a 9. iteraci dosahuje většího rozptylu hodnot, než pro 7. iteraci. Tedy tento model dává nejlepší



Obrázek 6.2: Porovnání modelů, které nepoužívají oříznutí. Oranžová čára udává hodnotu AUC pro ECFP4.



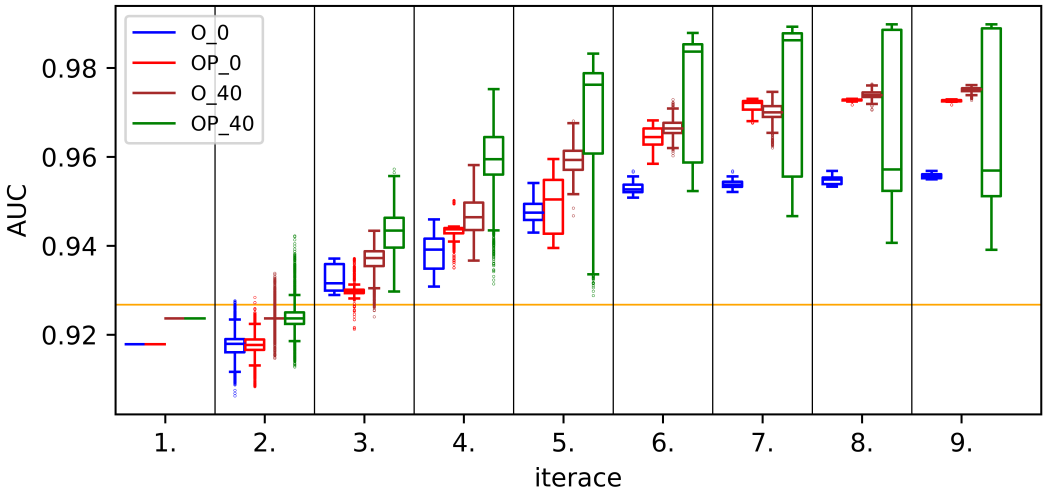
Obrázek 6.3: Porovnání modelů, které používají oříznutí. Oranžová čára udává hodnotu AUC pro ECFP4.



Obrázek 6.4: Porovnání nejlepších modelů z 2 skupin. Oranžová čára udává hodnotu AUC pro ECFP4.

výsledky pro 7. iteraci. Model *OSI* dává pro 7., 8. a 9. iteraci nejvyšší AUC, ale pro 9. iteraci má nižší průměrné hodnoty, tedy nejlepší iterace pro tento model je 7. a 8. iterace.

Z Obrázku 6.3 je vidět, že nejhůře si vedl model *OP_0*. To je způsobeno tím, že pro 2. iteraci jsme dostali velice málo hodnot, které přesáhly základní hodnotu AUC, konkrétně to byly 3 hodnoty. Ze tří skupin 2. iterace se poté skládají obtížně skupiny vyšších iterací, tedy pro 4. iteraci nám vyšla jediná skupina, která měla lepší hodnotu, než byla základní, a tedy pro 5. iteraci jsme již nedostali žádné skupiny. Proto jsme zkusili tomuto modelu posunout hranici základní hodnoty a to konkrétně na hodnotu AUC $\doteq 0.918$, kterou jsme dostali pro 1. iteraci tohoto modelu. Výsledky jsou vidět v Obrázku 6.5.



Obrázek 6.5: Posunutá hranice hodnot pro model *OP_0*. Oranžová čára udává hodnotu AUC pro ECFP4.

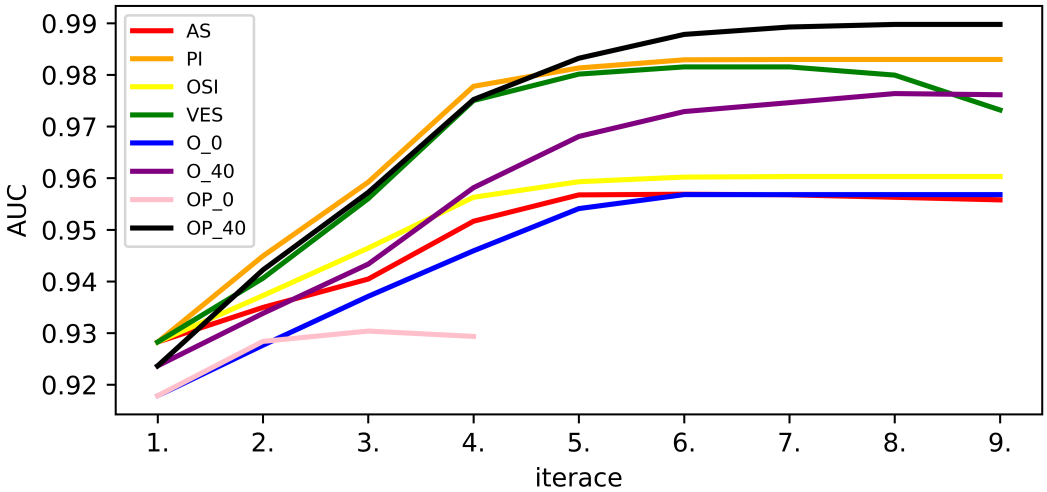
Z tohoto Obrázku 6.5 je vidět, že model *OP_0* dosahuje poměrně dobrých výsledků a pro 4. iteraci již všechny vygenerované skupiny měly lepší hodnotu AUC než ECFP4. Tedy z faktu, že model má špatné výsledky pro skupiny 2. iterace nemůžeme rozhodnout, že bude dosahovat špatných výsledků i v dalších iteracích.

Dále je z Obrázku 6.3 vidět, že nejlepších výsledků dosahuje model *OP_40*. Tento model dosahuje nejlepší hodnoty AUC pro 8. a 9. iteraci. Přesto bychom řekli, že 7. iterace dává pro tento model nejlepší výsledky, protože nejvyšší dosažené AUC pro 7. iteraci je pouze o 0.0005 nižší než pro 8. a 9. iteraci, ale průměrné hodnoty jsou mnohem vyšší. Model *O_0* dává nejvyšší AUC pro 6., 7., 8. i 9. iteraci. Zde jsou výsledky těchto iterací v podstatě totožné. Model *O_40* dosáhl nejvyššího AUC pro 8. iteraci. U tohoto modelu je tato iterace nejlepší. Dále vidíme, že pro modely s oříznutím je dobré nějaké oříznutí provést, protože modely s oříznutím 40% dávají mnohem lepší výsledky než stejné modely, kde oříznutí je 0%.

Ze všech modelů nejvyšší dosažené hodnoty AUC dosáhl model *OP_40*. Dále vidíme, že modely, které měly velký rozptyl hodnot u pozdějších iterací, dosahovaly vyšších hodnot než ty s malým rozptylem. Celkově bychom označili za nejlepší iteraci 7. iteraci, protože jak jsem hodnotili výše, tak pro 4 z 8 modelů jsme vybrali 7. iteraci jako nejlepší, přičemž model *OP_0* jsme nehodnotili, protože tam jsme měli problém s množstvím skupin. Tedy bychom mohli říci, že 7. iterace je nejlepší pro 4 ze 7 modelů. Navíc u modelů, u kterých jsme nevybrali 7. iteraci za nejlepší, tak jsme vybrali buď 6. nebo 8. iteraci a pro tyto iterace nebyly výsledné hodnoty o mnoho lepší než pro 7. iteraci.

Nejlepší skupina z modelu *OP_40* měla následující hodnoty: AUC $\doteq$ 0.990 (zaokrouhleno na 3 desetinná místa), EF 1% = 73.5 a EF 5% = 18. To jsou hodnoty, které v porovnání s Tabulkou 6.1 jsou na 1., 2. a 1. místě, tedy tento model s těmito skupinami dává velice nadprůměrné výsledky.

Nejvyšší dosažené hodnoty AUC pro jednotlivé modely jsou ještě znázorněny v Obrázku 6.6 .



Obrázek 6.6: Nejvyšší dosažené AUC pro jednotlivé modely a iterace.

To, že můžeme pro určité skupiny dostat lepší výsledek, než je základní jsme již zjistili v sekci 6.2. V této sekci výše jsme si ukázali, že lepších výsledků můžeme

dosáhnout i s našimi modely a toto zlepšení může být poměrně značné. Byly zde ovšem modely, které dávaly velice dobré výsledky, ale občas měly problém, že nějaké skupiny daly naopak poměrně špatné výsledky (horší než základní). To byl např. model *VES*. Tedy by nás zajímalo, kolik z celkově vygenerovaných skupin dané iterace skutečně přesáhlo danou základní hodnotu. Tabulka 6.2 uvádí pro určitou iteraci modelu počet lepších výsledků než je základní hodnota. Tabulka 6.3 udává počet vygenerovaných skupin pro určitý model a určitou iteraci.

Model	**2.**	**3.**	**4.**	**5.**	**6.**	**7.**	**8.**	**9.**
AS	10152	4938	2837	1791	475	189	45	1
PI	9119	4932	4138	2995	2104	801	442	379
OSI	10690	4950	3458	1917	235	79	41	20
VES	9781	4950	3963	2177	637	370	203	96
O_0	11	55	495	706	119	60	25	7
O_40	218	4717	3872	2264	932	1002	1474	696
OP_0	3	3	1	0	0	0	0	0
OP_40	1176	4950	4264	3173	1822	967	413	162

Tabulka 6.2: Počet lepších skupin v určité iteraci pro daný model.

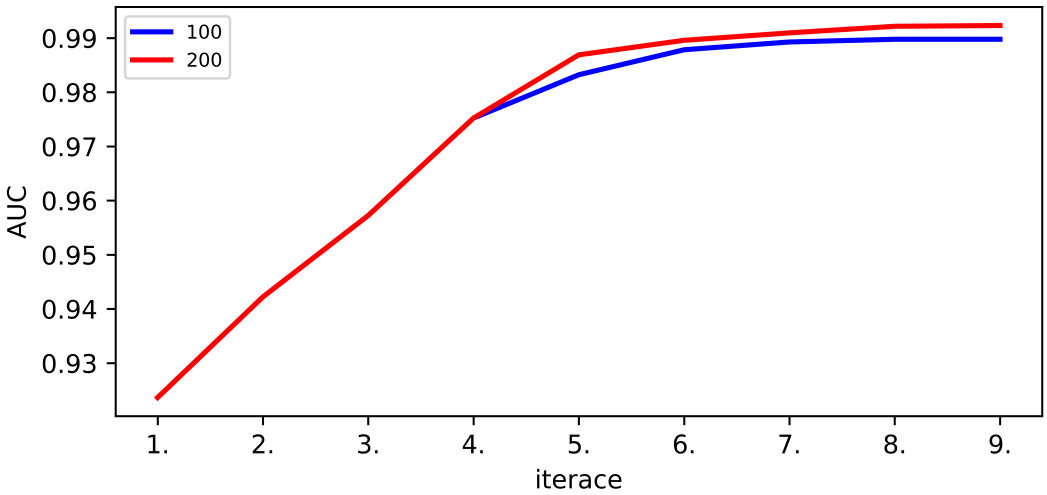
Model	**2.**	**3.**	**4.**	**5.**	**6.**	**7.**	**8.**	**9.**
AS	10878	4938	2837	1791	475	189	45	1
PI	10878	4932	4138	2995	2104	801	442	379
OSI	10878	4950	3458	1917	235	79	41	20
VES	10878	4950	3963	2177	658	396	224	116
O_0	10878	55	495	706	119	60	25	7
O_40	10878	4728	3872	2264	932	1002	1474	696
OP_0	10878	3	1	0	0	0	0	0
OP_40	10878	4950	4264	3173	1822	967	413	162

Tabulka 6.3: Počet vygenerovaných skupin v určité iteraci pro daný model.

Z Tabulky 6.3 vidíme, že až na modely *O_0*, *O_40* počty vygenerovaných skupin klesají. Toto dává smysl, protože ve výpočtu odstraňujeme stejné skupiny. Tedy díky Tabulce 6.3 můžeme pozorovat, jak moc má daný model rozdílné skupiny, protože pokud bychom měli všechny skupiny s rozdílnými indexy, tak z těchto skupin bychom v příští iteraci nedostali žádné stejné skupiny. V kombinaci s Obrázkem 6.6 můžeme vidět, že modely, které mají v pozdějších iteracích více skupin, dávají lepší výsledky. V modelu *O_0* je tomu jinak, protože v tomto modelu v 2. iteraci dostaneme pouze 11 skupin lepších než je základní hodnota. Z těchto skupin vygenerujeme 55 rozdílných skupin, tedy zde nejprve počet skupin roste, ale poté si začne být více skupin podobných a počty začnou klesat. Dále je zajímavé pozorovat, že pro 3. iteraci a dál počty vygenerovaných skupin a počty skupin, které z nich přesáhly základní hodnotu, se liší pouze v několika málo případech a pouze minimálně. Pro 4. iteraci a dál je to dokonce pouze model *VES*. To tedy znamená, že pokud bychom si vzali skupiny, které měly lepší

výsledky pro 2. iteraci, tak skupiny, které vzniknou jejich sloučením, budou mít s vysokou pravděpodobností také lepší výsledky.

Poté jsme zjišťovali, jestli bychom pro model *OP_40*, který dosáhl nejvyššího AUC, mohli dostat ještě vyšší nejvyšší AUC, pokud bychom v každé iteraci nevybírali 100 nejlepších výsledků (*100*) ale 200 (*200*). Výsledek je vidět v Obrázku 6.7.



Obrázek 6.7: Model *OP_40* pro rozdílný počet výběru nejlepších výsledků.

Z Obrázku 6.7 je vidět, že můžeme dostat ještě vyšší nejvyšší AUC. Konkrétně výsledky vyšly (zaokrouhleno na 5 desetinných míst): pro 200 nejlepších - AUC $\doteq$ 0.99232; pro 100 nejlepších - AUC $\doteq$ 0.98978.

6.4 Ověření indexů

Nyní chceme zjistit, jestli je možné pouze na základě množiny indexů určit skupiny, které budou dávat dobré výsledky. Pro každý model jsme si nechali vypsat 15 indexů (pokud jich alespoň 15 bylo), které se ve 100 nejlepších výsledcích v každé iteraci 2-9 vyskytovaly celkově nejvícekrát. Všechny takové indexy jsou k nalezení v Příloze A.4. V Tabulce 6.4 uvádíme indexy, které se vyskytovaly alespoň ve 2 modelech.

Celkově bylo 148 indexů. V Tabulce 6.4 je uvedeno 31 indexů, které si vedly nejlépe. Pozorujeme, že žádný index se nevyskytuje v této tabulce ve všech modelech. Nejvíce se vyskytuje maximálně v 5 modelech. Modely *AS* a *VES* mají dost podobné indexy, stejně tak to platí i pro modely *PI* a *OSI*, *O_0* a *O_40*, *OP_0* a *OP_40*.

Jelikož jsou tyto hodnoty pro 100 nejlepších indexů (pokud jich alespoň 100 bylo) z iterací 2-9, tak maximální počet skupin, ve kterých se může určitý index vyskytovat, je 800. Tedy vidíme, že např. v modelu *OP_40* se vyskytuje index *882399112* celkem 772-krát, což znamená, že je v podstatě v každé skupině. Dále můžeme pozorovat, že model *AS* v 15 nejvíce se vyskytujících indexech obsahuje index, který se vyskytl ve výsledných skupinách pouze 11-krát. To tedy znamená, že u vyšších iterací budeme dostávat velice málo skupin, protože budeme mít málo

Index	AS	PI	OSI	VES	O_0	O_40	OP_0	OP_40
847961216	526	-	-	688	455	730	-	-
864942730	500	-	-	638	413	720	-	-
2246699815	483	-	-	540	-	626	-	-
3217380708	464	-	-	631	-	-	-	-
3218693969	462	-	-	610	-	-	-	-
951226070	446	-	-	442	-	-	-	-
847867887	402	-	-	-	231	471	-	-
3824050433	400	-	-	-	-	446	-	-
3106190496	275	-	-	-	-	255	-	-
3498475363	294	397	-	-	-	-	-	-
1510328189	26	-	-	249	-	406	-	-
2246728737	11	-	-	501	-	-	-	-
882399112	-	681	501	436	-	-	4	772
98513984	-	655	540	421	-	-	-	-
3983062349	-	608	-	-	-	-	7	665
3351556771	-	604	-	-	-	-	-	642
369205567	-	565	-	-	-	-	4	-
4121755354	-	468	534	-	-	-	4	-
2297887526	-	434	398	-	-	-	-	172
2132511834	-	392	464	-	-	505	-	-
2720313463	-	314	357	-	-	-	-	-
2752034647	-	310	395	-	-	-	-	-
2331740049	-	289	371	-	-	-	-	326
1530134232	-	267	-	-	-	-	-	489
493191496	-	-	281	-	-	-	-	323
1530636652	-	-	84	-	-	-	-	539
1135286194	-	-	-	497	209	-	-	-
864674487	-	-	-	400	382	615	-	-
2041434490	-	-	-	-	373	632	-	-
422715066	-	-	-	-	366	635	-	-
3189457552	-	-	-	-	307	557	-	-

Tabulka 6.4: Ke každému indexu je uveden počet, kolikrát se daný index vyskytoval ve 100 nejlepších výsledcích v iteracích 2-9 v daném modelu.

indexů. To koresponduje s Tabulkou 6.3, kde pro model *AS* dostaneme v 9. iteraci již pouze 1 skupinu.

Dále bychom se chtěli podívat, jak vypadají skupiny, které dávají nejlepší výsledky. Výše jsme určili, že nejlepší iterace je 7. iterace, tak bychom se chtěli podívat, jestli skupiny, které vznikly pro různé modely, jsou ve svých velikostech stejné nebo alespoň podobné. Toto neplatí, jelikož když se podíváme na nejlepší skupiny 7. iterace pro jednotlivé modely, tak jsou velice odlišné, např. model *VES* obsahuje 7 dvojic, zatímco model *O_0* obsahuje jednu dvanáctici.

Celkově můžeme říci, že pouze z indexů aktivních molekul nedokážeme předem určit, jestli daná skupina v určitém modelu bude mít lepší nebo horší výsledek, než je základní. Tedy pouze na základě indexů aktivních molekul nedokážeme určit skupiny, které budou dávat dobré výsledky. Ovšem pro modely existují indexy, které při tvorbě skupin, mají větší pravděpodobnost pro vykazování dobrých výsledků. Ale nemůžeme říci, že při použití jiných indexů, které zde uvedeny nejsou, budou výsledky špatné, protože jsme v každé iteraci vybírali 100 nejlepších výsledků a tedy výsledky indexů, které neměly zastoupení ve 100 nejlepších výsledcích pro skupiny 2. iterace jsme již dále nepozorovali.

6.5 Všechny aktivní molekuly

V posledním experimentu nás zajímalo, jestli můžeme dosáhnout ještě lepších výsledků, pokud bychom do trénovací sady aktivních molekul přidali i všechny aktivní molekuly z testovací sady (budeme značit jako *all*). Což samozřejmě můžeme pro modely, které nechávají aktivní molekuly rozdělené, tedy: *AS*, *PI*, *OSI*, *VES*. Pro tyto modely budou hodnoty AUC, EF 1% a EF 5% nejvyšší možné, protože všechny aktivní molekuly z testovací sady budou mít skóre rovné jedné, jelikož podobnost 2 stejných molekul je 1. Co by mohlo snížit toto vyhodnocení je, kdyby nějaká neaktivní testovací molekula byla stejná s aktivní testovací molekulou. Takové molekuly se ale v testovací sadě nevyskytují.

Tedy jsme tento experiment pustili pro model, který dává všechny indexy aktivních molekul dohromady. Použili jsme model, který dosáhl nejvyššího AUC, tedy model *OP_40*. Opět jsme v každé iteraci vybírali 100 nejlepších výsledků. Výsledek v porovnání s přístupy stejného modelu, ale když jsme používali klasickou sadu aktivních molekul a pro výběr 100 nejlepších v každé iteraci a pro výběr 200 nejlepších v každé iteraci je znázorněn na Obrázku 6.8.

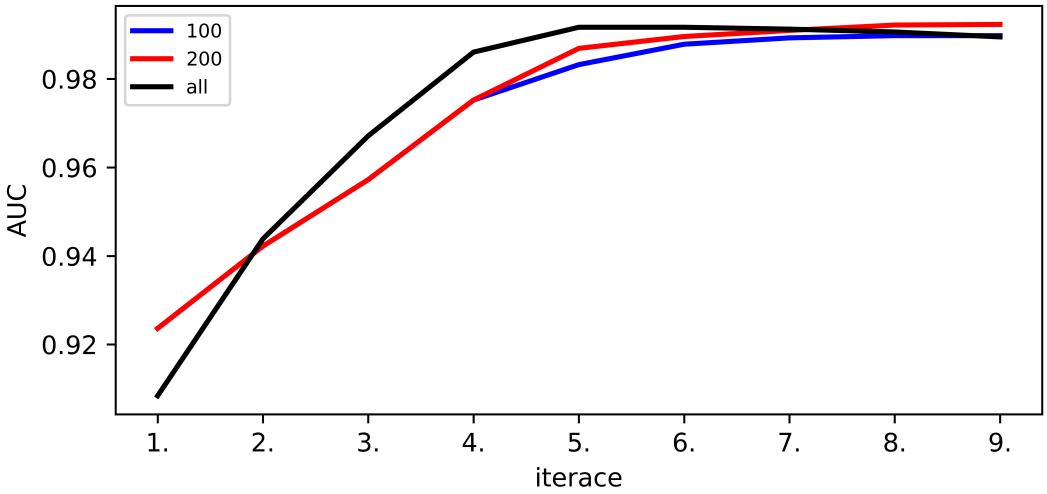
Z Obrázku 6.8 je vidět, že nejvyšší dosažené hodnoty AUC se moc neliší. Rozdíl je v tom, že *all* dosahuje velice dobrých výsledků rychleji než *100* a *200* a poté jeho výkonnost začíná trochu klesat. Nejvyšší dosažené hodnoty AUC jsou následující (budeme zaokrouhlovat na 5 desetinných míst): *all* má AUC $\doteq 0.99168$; *200* má AUC $\doteq 0.99232$; *100* má AUC $\doteq 0.98978$. Je tedy vidět, že rozdíly jsou minimální, z čehož vyplývá, že všechny důležité informace pro tvorbu modelu již známe z trénovací sady aktivních molekul.

Poté jsme se podívali na indexy těchto 3 přístupů, abychom mohli porovnat, jestli indexy, které dávají dobré výsledky pro *all* jsou obsaženy v trénovací sadě aktivních molekul. Do Tabulky 6.5 jsme opět umístili indexy s počty, kolikrát se daný index vyskytl ve výsledcích v iteracích 2-9.

Z Tabulky 6.5 vidíme, že indexy pro *100* a *200* jsou poměrně podobné, což se dalo očekávat. Indexy pro *all* se ale poměrně liší od *100* a *200*. To, že se

Index	**100**	**200**	**all**
882399112	772	1491	688
3692055567	709	1477	529
3983062349	665	1345	-
3351556771	642	1252	-
1530636652	539	1091	-
1510656476	493	948	-
1530134232	489	926	-
300633889	455	976	-
3482873808	420	751	-
2331740049	326	475	-
493191496	323	-	-
3586270004	250	418	-
2050059121	218	-	-
1627192235	214	408	-
2297887526	172	-	-
4121755354	-	900	-
2117068077	-	863	-
1343371647	-	432	-
3624155	-	-	742
74537039	-	-	634
3980805843	-	-	585
271891986	-	-	524
349847563	-	-	441
888801799	-	-	400
18429960	-	-	397
1096334908	-	-	389
3532542749	-	-	322
2070976245	-	-	212
3176806076	-	-	176
1879358772	-	-	146
2990306066	-	-	145

Tabulka 6.5: Ke každému indexu je uveden počet, kolikrát se daný index vyskytoval v iteracích 2-9 ve 100 (nebo 200 pro *200*) nejlepších výsledcích daného modelu.



Obrázek 6.8: Porovnání *OP_40*.

nevyskytují některé indexy *all* v prvních dvou sloupcích může být způsobeno 3 faktory:

1. daný index se nevyskytuje v sadě aktivních molekul
2. daný index se vyskytuje v sadě aktivních molekul, ale neprošel oříznutím
3. daný index se vyskytuje ve výsledcích pro *100* a *200*, ale ne tak často aby se vešel do 15 nejčastějších indexů

2. bod nesplňuje žádný index z Tabulky 6.5; třetí bod splňují indexy *3624155*, *3176806076*, *3532542749*, *1879358772*, *2990306066*; ostatní splňují bod 1. Je tedy vidět, že i když neznáme všechny indexy aktivních molekul, tak jsme schopni dostat srovnatelné výsledky, jako když je známe.

Závěr

Zjistili jsme, že v námi implementovaných modelech mohou kolize vést k velice dobrým výsledkům, kterých jsme, v běžně používaných přístupech či přístupech strojového učení, nedosáhli. U většiny modelů platilo, že většina námi provedených kolizí je prospěšná. Nicméně u určité kolize nedokážeme předem s jistotou říci, jestli díky ní budeme dostávat lepší či horší výsledky. Pro každý model jsme zjistili indexy fragmentů, které když se vyskytnou v kolizi, tak s velkou pravděpodobností budou dávat dobré výsledky. Pokud máme 2 kolize, které zlepšily výkonnost, a tyto kolize bychom spojili dohromady, tak při zvolení správného modelu lze očekávat, že výsledná výkonnost nebude horší. Celkově můžeme říci, že správné kolize mohou vést ke zlepšení nacházení biologicky aktivních molekul.

V budoucí práci bychom se chtěli zaměřit na podrobnější analýzu jednotlivých skupin, které zlepšily výkonnost. Nyní totiž víme, že existuje velké množství skupin, které zlepšují výsledek, ale nevíme, jak je nalézt pouze ze znalosti indexů aktivních molekul.

Seznam použité literatury

[1] the University of Wisconsin-Madison UW Health. Uwccc small molecule screening facility. Dostupné z: https://cancer.wisc.edu/research/resources/ddc/.

[2] David Hoksza Petr Škoda. Benchmarking platform for ligand-based virtual screening. *2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, 2016. Dostupné z: https://doi.org/10.1109/BIBM.2016.7822693,DOI: 10.1109/BIBM.2016.7822693.

[3] Hoksza D Krivák R. Improving protein-ligand binding site prediction accuracy by classification of inner pocket points using local features. *Journal of cheminformatics*, 2015 Dec. DOI: 10.1186/s13321-015-0059-5.

[4] Demetrios K. Vassilatis Zoe Cournia Evanthia Lionta, George Spyrou. Structure-based virtual screening for drug discovery: Principles, applications and recent advances. *Current Topics in Medicinal Chemistry*, 2014 Aug. DOI: 10.2174/1568026614666140929124445.

[5] Dennis H. Smith Raymond E. Carhart and R. Venkataraghavan. Atom pairs as molecular features in structure-activity studies: definition and applications. *Journal of Chemical Information and Computer Sciences*, pages 64–73, 1985. Dostupné z: https://doi.org/10.1021/ci00046a002, DOI: 10.1021/ci00046a002.

[6] J. Scott Dixon Ramaswamy Nilakantan, Norman Bauman and R. Venkataraghavan. Topological torsion: a new molecular descriptor for sar applications. comparison with other descriptors. *Journal of Chemical Information and Computer Sciences*, pages 82–85, 1987. Dostupné z: https://doi.org/10.1021/ci00054a008, DOI: 10.1021/ci00054a008.

[7] David Rogers and Mathew Hahn. Extended-connectivity fingerprints. *Journal of Chemical Information and Modeling 2*, pages 742–754, 2010. Dostupné z: https://doi.org/10.1021/ci100050t, DOI: 10.1021/ci100050t.

[8] Yiqun Cao, Tao Jiang, and Thomas Girke. A maximum common substructure-based algorithm for searching and predicting drug-like compounds. *Bioinformatics*, 24(13):i366–i374, 07 2008. DOI: 10.1093/bioinformatics/btn186.

[9] Petr Škoda David Hoksza. Using bayesian modeling on molecular fragments features for virtual screening. *2016 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)*, 2016. Dostupné z: https://doi.org/10.1109/CIBCB.2016.7758111,DOI: 10.1109/CIBCB.2016.7758111.

[10] Sereina Riniker and Gregory A Landrum. Open-source platform to benchmark fingerprints for ligand-based virtual screening. *Journal of Cheminformatics*, 2013. Dostupné z: http://www.jcheminf.com/content/5/1/26.

[11] *Python*. Dostupné z: https://www.python.org/.

[12] *RDKit: Open-Source Cheminformatics Software*. Dostupné z: https://www.rdkit.org/.

[13] *scikit-learn*. Dostupné z: https://scikit-learn.org/stable/.

[14] *Matplotlib*. Dostupné z: https://matplotlib.org/.

A. Přílohy

A.1 Seznam skriptů a jejich argumentů

Zde uvedeme všechny skripty a jejich argumenty. Pokud neuvedeme jinak, tak vstupní a výstupní soubory z jednotlivých skriptů jsou ve formátu json.

A.1.1 Benchmarkovací program

Skript *extract_fragments.py* pro sady molekul extrahuje fragmenty molekul a jejich indexy. Spouští se následujícím způsobem s následujícími argumenty:

python extract_fragments.py

- -i (čárkou oddělené 3 vstupní soubory obsahující molekuly, 1. soubor obsahuje aktivní molekuly, 2. soubor obsahuje neaktivní molekuly, 3. soubor obsahuje testovací molekuly)
- -o (3 výstupní soubory oddělené čárkou - kam se mají výsledky vypsat, 1. soubor bude obsahovat fragmenty s indexy aktivních molekul, 2. soubor bude obsahovat fragmenty s indexy neaktivních molekul, 3. bude obsahovat fragmenty s indexy testovacích molekul)
- -f (typ molekulového otisku, jehož fragmenty chceme získat)
- -p (typ vstupních souborů molekul - povolené jsou 2 typy: smi, sdf; implicitně je smi)
- --kekule (vypíše kekulovou formu fragmentů, volitelné)
- --isomeric (přidá 3D informaci fragmentů, volitelné)

Typy molekulových otisků musí mít následující tvar:

- tt.{velikost}
- ecfp.{velikost}
- fcfp.{velikost}
- ap

kde velikost pro tt je počet na sebe navazujících atomů, pro ecfp a fcfp je to průměr fragmentu.

Výstupní soubory z *extract_fragments.py* poté použijeme při výpočtu deskriptorů, jako jsou váha molekuly, počet valenčních elektronů atd. Ten se spouští tímto způsobem:

python compute_descriptors.py

- -i (3 vstupní soubory z *extract_fragments.py* oddělené čárkou, opět v pořadí: fragmenty s indexy aktivních molekul, fragmenty s indexy neaktivních molekul, fragmenty s indexy testovacích molekul)

- -o (jména 3 výstupních csv souborů oddělené čárkou: 1. bude obsahovat deskriptory aktivních molekul, 2. deskriptory neaktivních molekul, 3. deskriptory testovacích molekul)
- --fragments (pokud tento argument napíšeme, tak se budou počítat deskriptory pro molekuly, jinak pro fragmenty)

Dále budeme vytvářet model ve skriptu *create_model.py*. Spouští se následovně:
python create_model.py

- -c (konfigurační soubor modelu)
- -a (vstupní soubor s fragmenty a indexy aktivních molekul - dostali jsme z *extract_fragments.py*)
- -f (vstupní soubor s fragmenty a indexy neaktivních molekul - dostali jsme z *extract_fragments.py*)
- -d (vstupní soubor s deskriptory aktivních molekul - výstup z *compute_descriptors.py*)
- -i (vstupní soubor s deskriptory neaktivních molekul - výstup z *compute_descriptors.py*)
- -o (výstupní soubor, který bude obsahovat model)

Vytvořený model použijeme na výpočet skóre molekul z testovací sady:
python score_molecules.py

- -m (vstupní soubor obsahující model)
- -f (vstupní soubor obsahující fragmenty s indexy testovacích molekul - výstupní soubor z *extract_fragments.py*)
- -d (vstupní soubor obsahující deskriptory testovacích molekul - výstupní soubor z *compute_descriptors.py*)
- -o (výstupní soubor, který bude obsahovat molekuly z testovací sady a k nim přiřazená skóre)

K přiřazeným skóre molekul potřebujeme dodat jejich skutečnou aktivitu:
python add_activity.py

- -s (vstupní soubor, který jsme dostali ze *score_molecules.py*)
- -a (vstupní soubor s aktivitou testovacích molekul)
- -o (výstupní soubor, který pro každou molekulu bude obsahovat její skóre a její skutečnou aktivitu)

Nakonec ještě potřebujeme vyhodnotit celý model - spočítat AUC, EF:
python compute_evaluation.py

- -i (vstupní soubor, který je výstupním souborem z *add_activity.py*)

- -o (výstupní soubor, který bude obsahovat vyhodnocení modelu)

Jak jsme již ukázali v uživatelské dokumentaci, tak celý tento výše uvedený proces se dá spustit pomocí skriptu *run_all_scripts.py*:
python run_all_scripts.py

- -a (volitelný vstupní soubor obsahující aktivní molekuly, implicitně: *data/actives.smi*)
- -i (volitelný vstupní soubor obsahující neaktivní molekuly, implicitně: *data/inactives.smi*)
- -t (volitelný vstupní soubor obsahující testovací molekuly, implicitně: *data/test.smi*)
- -c (volitelný vstupní soubor obsahující aktivitu testovacích molekul, implicitně: *data/test_activity.json*)
- -m (vstupní soubor obsahující konfiguraci modelu)
- -o (výstupní soubor - kam se vypíše vyhodnocení modelu)
- -p (volitelný argument určující typ vstupních souborů obsahující molekuly, jsou 2 možnosti: smi, sdf; implicitně: smi)
- -d (volitelný argument, pokud nás zajímají i jiné výstupní soubory než vyhodnocení, tak zadáme do tohoto argumentu cestu do adresáře, kam se mají vypsat)

A.1.2 Skupiny

run_all_pair.py má následující argumenty:

- -i (vstupní soubor, který obsahuje fragmenty s indexy aktivních molekul)
- -t (vstupní soubor, který obsahuje fragmenty s indexy testovacích molekul)
- -a (vstupní soubor, který obsahuje aktivitu testovacích molekul)
- -m (jméno modelu)
- -n (číslo, které určuje, na kolika jádrech počítače proběhne výpočet)
- -c (volitelný argument, který se zadává u modelů, které používají oříznutí, udává se jako číslo od 0 do 100 (reprezentace procent))
- -o (cesta k výstupnímu adresáři, kam se vypíší výsledky)

pair_analysis.py má tyto argumenty:

- -f (vstupní soubor obsahující fragmenty s indexy aktivních molekul)
- -b (vstupní soubor, který obsahuje výsledky základního modelu - jedná se o soubor, který dostaneme z *run_all_scripts.py* nebo z *compute_evaluation.py*)

- -d (adresář obsahující všechny výsledky modelů - jedná se o adresář, který jsme zadali jako výstupní v *run_all_pair.py* + přidáme */evaluations*)
- -o (adresář, kam se vypíší soubory obsahující skupiny 2. iterace roztříděné do souborů podle toho, v jakých aspektech měly lepší vyhodnocení než základní model)

select_best_results.py:

- -f (vstupní soubor, ze kterého chceme vybrat nejlepší výsledky)
- -b (číslo, kolik nejlepších výsledků chceme vybrat)
- -t (podle kterého kritéria budeme vybírat, možnosti: AUC, EF1, EF5)
- -o (výstupní soubor)

run_all_groups.py:

- -i (vstupní soubor obsahující fragmenty s indexy aktivních molekul)
- -g (vstupní soubor obsahující skupiny, které chceme dávat dohromady - jeden z výstupních souborů kromě *baseline.json* z *pair_analysis.py* nebo *group_analysis.py* nebo *select_best_results.py*)
- -t (vstupní soubor s fragmenty a indexy testovacích molekul)
- -a (vstupní soubor s aktivitou testovacích molekul)
- -m (jméno modelu)
- -n (počet jader počítače, na kterých má běžet výpočet)
- -c (volitelný argument, který se zadává u modelů, které používají oříznutí - velikost oříznutí, udává se jako číslo od 0 do 100 (reprezentace procent))
- -o (adresář kam se uloží 4 výstupní adresáře)

group_analysis.py:

- -b (vstupní soubor, ze kterého budeme počítat základní hodnoty)
- -c (adresář *configurationfiles* z *run_all_groups.py*)
- -e (adresář *evaluations* z *run_all_groups.py*)
- -o (adresář, kam se uloží výsledky roztříděné do stejných souborů jako z *pair_analysis.py*)

A.1.3 Kreslení obrázků

Argumenty pro *print_group_graphs.py*:

- -b (soubor obsahující základní hodnoty - soubor *baseline.json* z *pair_analysis.py* nebo později z *group_analysis.py*)
- -i (soubor, který jsme dostali z *pair_analysis.py* nebo *select_best_results.py* nebo *group_analysis.py* (kromě souboru *baseline.json*))
- -o (adresář, kam uložit výstupní png soubory)

print_all_analyses_graphs.py:

- -i (vstupní soubory oddělené čárkou, pro každou iteraci 1 soubor, např. můžeme vzít pro každou iteraci *auc.json*)
- -n (pojmenování souborů z argumentu -i; pod těmito pojmenováními se budou jednotlivé soubory znázorňovat v obrázcích)
- -o (adresář kam se vykreslí výstupní png soubory)

A.2 Názvy skriptů k modelům

Český název	**Název skriptu**
základní indexový model	control_model.py
lineární regrese	linear_regression_model.py
rozhodovací stromy	decision_tree_model.py
deskriptor model	descriptors_model.py
model tvořený všemi aktivními indexy	active_index_model.py
hashovací model (pro AP)	nbit_ap_model.py
hashovací model (pro ECFP)	nbit_ecfp_model_vector.py
hashovací model (pro FCFP)	nbit_fcfp_model_vector.py
hashovací model (pro TT)	nbit_tt_model.py
aktivní skupiny	active_group_model.py
přidej index	add_group_model.py
odstraň stejné indexy	delete_index_group_model.py
velikost skupiny	group_and_add_model.py
ořízni indexy	cutoff_active_group_model.py
ořízni a přidej indexy	cutoff_and_group_model.py

Tabulka A.1: Názvy skriptů k modelům.

A.3 Porovnání základních modelů

AUC zaokrouhlujeme na 3 desetinná místa. *model model tvořený všemi aktivními indexy* budeme zkracovat jako *via*. *Model tvořený všemi aktivními indexy* a

deskriptor model jsme pouštěli pro $k_1 = 1$ a $k_2 = 1$. Uvedené výsledky skupinových modelů jsou výsledky pro 1. iteraci. U modelů, kde jsme používali oříznutí je u názvu modelu v procentech uvedeno, jak velké toto oříznutí bylo.

Model	**Typ**	**AUC**	**EF 1%**	**EF 5%**
ECFP	ECFP2	0.916	49	16
ECFP	ECFP4	0.927	63.7	17
ECFP	ECFP6	0.925	63.7	17
ECFP	ECFP8	0.926	68.6	17
ECFP	ECFP10	0.927	68.6	17
ECFP	ECFP12	0.928	63.7	17
FCFP	FCFP2	0.9	34.3	14
FCFP	FCFP4	0.92	53.9	15
FCFP	FCFP6	0.926	58.8	16
FCFP	FCFP8	0.922	58.8	16
FCFP	FCFP10	0.922	58.8	16
FCFP	FCFP12	0.924	63.7	16
AP	AP	0.913	63.7	16
TT	TT2	0.907	44.1	14
TT	TT3	0.913	63.7	16
TT	TT4	0.914	58.8	16
TT	TT5	0.924	63.7	17
TT	TT6	0.937	68.6	17
via	AP	0.839	9.8	9
via	ECFP4	0.93	58.8	17
via	ECFP6	0.938	58.8	17
via	ECFP10	0.942	49	16
via	FCFP4	0.911	49	13
via	FCFP6	0.934	39.2	14
via	FCFP10	0.93	29.4	15
via	TT	0.892	24.5	10
rozhodovací strom	ECFP4	0.962	39.2	8
lineární regrese	ECFP4	0.819	4.9	6
deskriptor model	AP	0.891	83.8	17
deskriptor model	ECFP4	0.919	49.0	15
deskriptor model	ECFP6	0.935	53.9	16
deskriptor model	ECFP10	0.932	39.2	15
deskriptor model	FCFP4	0.919	49	15
deskriptor model	FCFP6	0.935	53.9	16
deskriptor model	FCFP10	0.932	39.2	15
deskriptor model	TT	0.91	39.2	12
aktivní skupiny	ECFP4	0.928	68.6	17
přidej indexy	ECFP4	0.928	68.6	17
odstraň stejné indexy	ECFP4	0.928	68.6	17
velikost skupiny	ECFP4	0.928	68.6	17
ořízni indexy_0	ECFP4	0.918	58.8	15
ořízni a přidej indexy_0	ECFP4	0.918	58.8	15

ořízni indexy_40	ECFP4	0.924	58.8	15
ořízni a přidej indexy_40	ECFP4	0.924	58.8	15
ořízni indexy_60	ECFP4	0.776	9.8	3
ořízni a přidej indexy_60	ECFP4	0.776	9.8	3

Tabulka A.2: Porovnání běžně používaných metod, strojového učení a námi definovaných přístupů.

A.4 Porovnání indexů

Index	AS	PI	OSI	VES	O_0	O_40	OP_0	OP_40
847961216	526	-	-	688	455	730	-	-
864942730	500	-	-	638	413	720	-	-
2968968094	493	-	-	-	-	-	-	-
2246699815	483	-	-	540	-	626	-	-
3217380708	464	-	-	631	-	-	-	-
3218693969	462	-	-	610	-	-	-	-
951226070	446	-	-	442	-	-	-	-
847867887	402	-	-	-	231	471	-	-
3824050433	400	-	-	-	-	446	-	-
784740959	319	-	-	-	-	-	-	-
3106190496	275	-	-	-	-	255	-	-
3498475363	294	397	-	-	-	-	-	-
3532542749	259	-	-	-	-	-	-	-
1510328189	26	-	-	249	-	406	-	-
2246728737	11	-	-	501	-	-	-	-
882399112	-	681	501	436	-	-	4	772
98513984	-	655	540	421	-	-	-	-
3248686995	-	614	-	-	-	-	-	-
3983062349	-	608	-	-	-	-	7	665
3351556771	-	604	-	-	-	-	-	642
369205567	-	565	-	-	-	-	4	-
4121755354	-	468	534	-	-	-	4	-
2297887526	-	434	398	-	-	-	-	172
2132511834	-	392	464	-	-	505	-	-
2720313463	-	314	357	-	-	-	-	-
2752034647	-	310	395	-	-	-	-	-
2331740049	-	289	371	-	-	-	-	326
1885379073	-	287	-	-	-	-	-	-
1530134232	-	267	-	-	-	-	-	489
3162837314	-	-	361	-	-	-	-	-
1352399629	-	-	316	-	-	-	-	-
493191496	-	-	281	-	-	-	-	323
1840994228	-	-	217	-	-	-	-	-
1316507708	-	-	120	-	-	-	-	-
1530636652	-	-	84	-	-	-	-	539
2245384272	-	-	43	-	-	-	-	-
994485099	-	-	-	692	-	-	-	-
1135286194	-	-	-	497	209	-	-	-
864674487	-	-	-	400	382	615	-	-
1412710081	-	-	-	371	-	-	-	-
2117068077	-	-	-	274	-	-	-	-
2041434490	-	-	-	-	373	632	-	-
422715066	-	-	-	-	366	635	-	-

3189457552	-	-	-	-	307	557	-	-
3337745080	-	-	-	-	264	-	-	-
3975275337	-	-	-	-	236	-	-	-
2076190208	-	-	-	-	223	-	-	-
1637836422	-	-	-	-	213	-	-	-
419746180	-	-	-	-	-	332	-	-
4086265842	-	-	-	-	-	227	-	-
3643586416	-	-	-	-	-	299	-	-
3692055567	-	-	-	-	-	-	-	709
1510656476	-	-	-	-	-	-	-	493
300633889	-	-	-	-	-	-	-	455
3482873808	-	-	-	-	-	-	-	420
3586270004	-	-	-	-	-	-	-	250
2050059121	-	-	-	-	-	-	-	218
1627192235	-	-	-	-	-	-	-	214

Tabulka A.3: Ke každému indexu je uveden počet, kolikrát se daný index vyskytoval ve 100 nejlepších výsledcích v iteracích 2-9 v daném modelu.