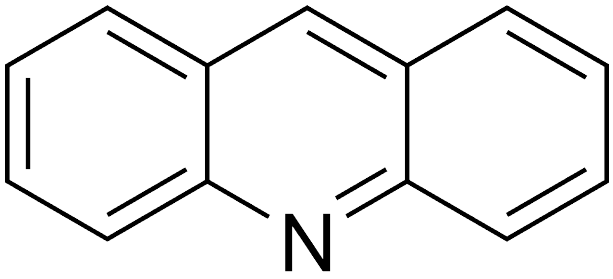
Slovenská Technická Univerzita

Fakulta Informatiky a Informačných Technológií

Tímový projekt



FOTOTOX

Vedúca tímu:
Ing. Marta Šoltésová Prnová, PhD.

Členovia tímu:
Bc. Matej Halinkovič
Bc. Jakub Knánik
Bc. Jakub Maruniak
Bc. Kateřina Mušková
Bc. Patrik Pavlačka
Bc. Tibor Sloboda

Kontakt:
tim20.tp2021@gmail.com
team20-21.studenti.fiit.stuba.sk

Obsah

Časť I:

Inžinierske dielo

1 Úvod

V tímovom projekte FOTOTOX sa venujeme predikcii fototoxicity chemických látok prostredníctvom metódy *in silico*. Tento dokument slúži ako dokumentácia k danému projektu.

V tejto časti dokumentácie sa možno dočítať o cieľoch projektu, z akých častí vyvijaný systém pozostáva a z akých modulov sú dané časti systému vytvorené. Pozrieme sa na architektúru riešenia a funkcionalitu jednotlivých častí modulov.

2 Globálne ciele projektu

Hlavným cieľom projektu je schopnosť predikovať fototoxicitu chemickej látky. Súčasťou dosiahnutia tohto cieľa je využitie modelov kvantitatívnej závislosti aktivity na štruktúre (QSAR), ktorými možno získať vzťahy medzi chemickou štruktúrou a biologickou aktivitou v súbore dát. Zo zhromaždených dát a s pomocou natrénovaného modelu budeme následne vedieť predikovať fototoxicitu látky, pričom výpočet fototoxicity bude prístupný pre používateľa prostredníctvom webovej aplikácie.

2.1 Zimný semester

Hlavné ciele zimného semestra:

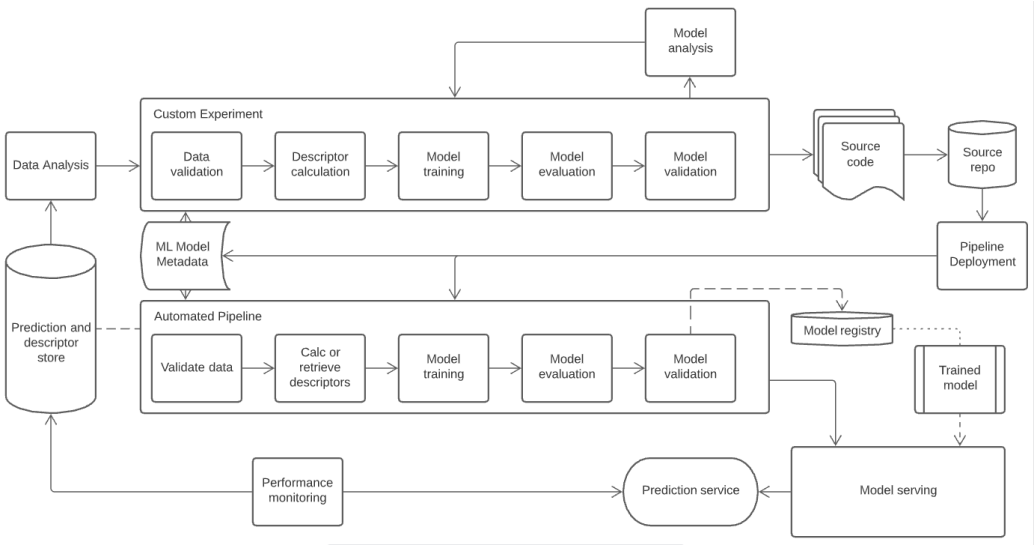
- Zadefinovanie si postupu práce a metodík.
- Získanie dostatočného množstva dát.
- Porozumenie problematike fototoxicity. Vykonanie EDA pre získané dáta.
- Identifikovanie a vypočítanie chemických deskriptorov, ktoré by mohli korelovať s fototoxicitou.
- Návrh celkovej architektúry.
- Návrh a implementácia modulu data pipe.
- Návrh a implementácia machine learning modelu.
- Návrh webovej aplikácie.
- Dodržiavanie agile postupov. Iteratívno-inkrementálny prístup k tvorbe softvéru.

3 Celkový pohľad na systém

Zo špecifikácie nášho projektu vyplynulo rozdelenie celkového systému na dve logické časti.

1. **Vytvorenie modelu**

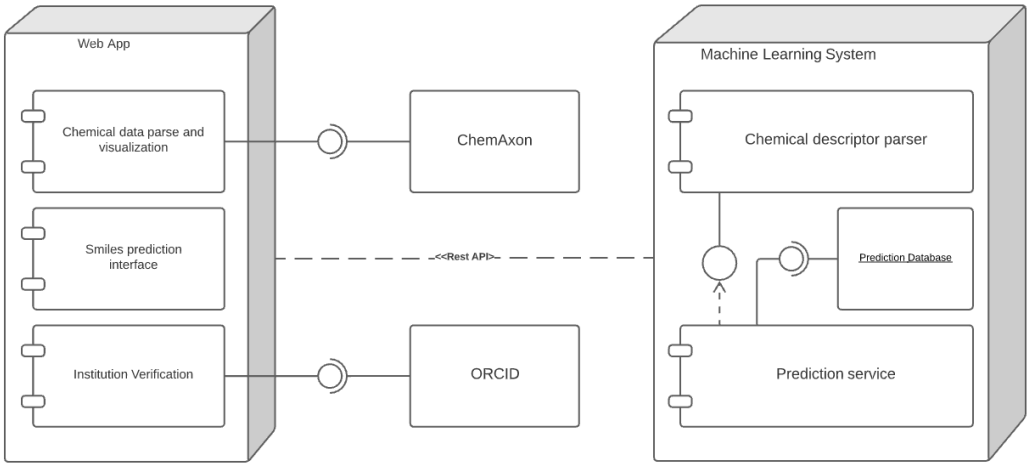
   Prvou časťou nášho projektu je vytváranie spôsobu ako predikovať fotoxicitu látky. Za týmto účelom používame prístupy strojového učenia, ktoré vyžadujú získavanie dát, predspracovanie dát, trénovanie modelov a ich validácia.



Obr. 1: Vysoko úrovňový pohľad na architektúru systému

2. **Vytvorenie web aplikácie**

   Druhou časťou nášho projektu je zdostupnenie nášho modelu pre verejnosť. Zvolili sme spôsob vývoja web aplikácie, ktorá nám pomôže dosiahnuť čo najširišie využitie.

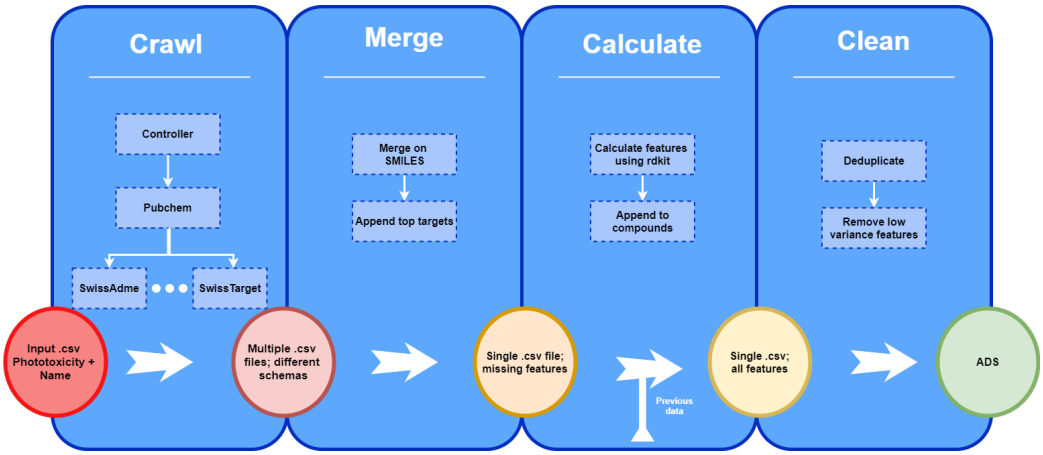


Obr. 2: Spôsob extrakcie a predspracovania dát pre modely

4 Moduly systému

4.1 Data pipe

Získavanie dát pre našu problematiku je iteratívny proces. Overovanie fototoxicity pre látky je zdĺhavé a môžeme preto spracovať zväčša len pár látok naraz a požiadavky na predspracovanie deskriptorov sa vyvíjajú. Z týchto dôvodov bolo pre nás dôležité zostaviť proces, ktorý nám umožňil automatizovať značnú čas práce a sústrediť sa tak na vývoj čo najlepšieho modelu.



Obr. 3: Spôsob extrakcie a predspracovania dát pre modely

1. **Crawl**

   Názvy látok a informácie o ich fototoxicite získavame zväčša manuálne z vedeckých publikácií. Tieto publikácie však neobsahujú hodnoty deskriptorov, ktoré potrebujeme na predikciu. Preto sme zostavili niekoľko crawlerov, ktoré zoberú zoznam látok, ktorý im zadáme a hodnoty deskriptorov vedia extrahovať z chemických databáz.

2. **Merge**

   Výstupy z našich crawlerov potrebujeme zjednotiť do jedného súboru pre jednoduchšie spracovanie v neskorších krokoch. Databázy pracujú na princípe SMILES kódov, ktoré používame ako kľúč pre zlučovanie.

3. **Calculate**

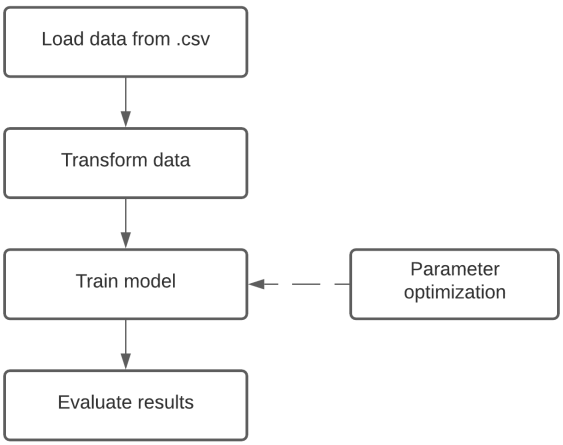
   Na základe poznatkov získaných z prednášok o fototoxicite sme identifikovali niekoľko užitočných deskriptorov, ktoré sa nezvyknú nachádzať v databázach. Pre ich vypočítanie používame knižnicu Rdkit. V tomto kroku pripájame informácie o nových látkach k už existujúcemu zoznamu látok z predchádzajúcich iterácií. Tento krok je potrebný pre zabezpečenie niektorých častí funkcionality čistenia dát.

4. **Clean**

   V poslednom kroku prípravy dát pre experimenty ich čistíme. Pre zaručenie stability našich modelov odstraňujeme premenné s nízkou variáciou, premenné ktoré sú vysoko korelované s inými a zabezpečujeme, že náš dataset neobsahuje duplikované látky.

Pre overenie správnosti funkcionality tejto pipeline, sme vytvorili aj niekoľko testov podľa metodiky testovania, bližšie popísanej v Prílohe B.

4.2 Trénovanie modelov



Obr. 4: Štruktúra experimentov

Základ nášho systému tvorí model, ktorý dokáže predikovať fototoxicitu látky. Zostavili sme experimentálny proces, ktorý nám dovolí jednoducho natrénovať optimálny model pre zadaný dataset stlačením jedného tlačidla.

Proces je implementovaný pomocou Jupyter Notebook a pre optimalizáciu modelu je využitý rámec TuneSearch. Tento proces nám umožňuje automaticky nájsť najlepšiu kombináciu modelu a jeho parametrov pre daný problém

Úspešnosť modelov je vyhodnocovaná na základe nasledujúcich metrík:

Metrika	**Výpočet**
Správnosť	$\frac{TP+TN}{TP+TN+FP+FN}$
Presnosť	$\frac{TP}{TP+FP}$
Úplnosť	$\frac{TP}{TP+FN}$
F1-skóre	$2 * \frac{presnosť * úplnosť}{presnosť + úplnosť}$

Tabuľka 1: TP - skutočná pozitivita (fototoxická látka predikovaná ako fototoxická), FP - falošná pozitivita (nefototoxická látka predikovaná ako fototoxická), TN - skutočná negativita (nefototoxická látka predikovaná ako nefototoxická), FN - falošná negativita (nefototoxická látka predikovaná ako fototoxická)

Modely vyhodnocujeme aj podľa systému SHAP, ktorý poskytuje informácie o dôležitosti jednotlivých premenných pre rozhodnutia modelu. Tento prístup nám umožňuje rozhodnúť či rozhnodutia dávajú zmysel z chemického hľadiska a pomáha nám určovať spoľahlivosť modelu. Presný popis použitých deskriptorov a ich dôležitosť pre rozhodnutia modelu je možné vidieť v Prílohe F.

Aktuálne najlepší model je postavený na optimalizovanej RandomForest architekrúre a dosahuje nasledovné výsledky:

Metrika	**Hodnota**
Správnosť	78.58%
Presnosť	89.47%
Úplnosť	80.95%
F1-skóre	85.00%

4.3 Webová aplikácia

Naša aplikácia pozostáva celkovo z 2 oddelených komunikujúcich modulov, a to samotnej web-aplikácie s ktorou priamo interaguje používateľ, a predikčnej služby ako mikroslužby nasadenej samostatne. Pre komunikáciu s predikčnou službou bude využité REST API.

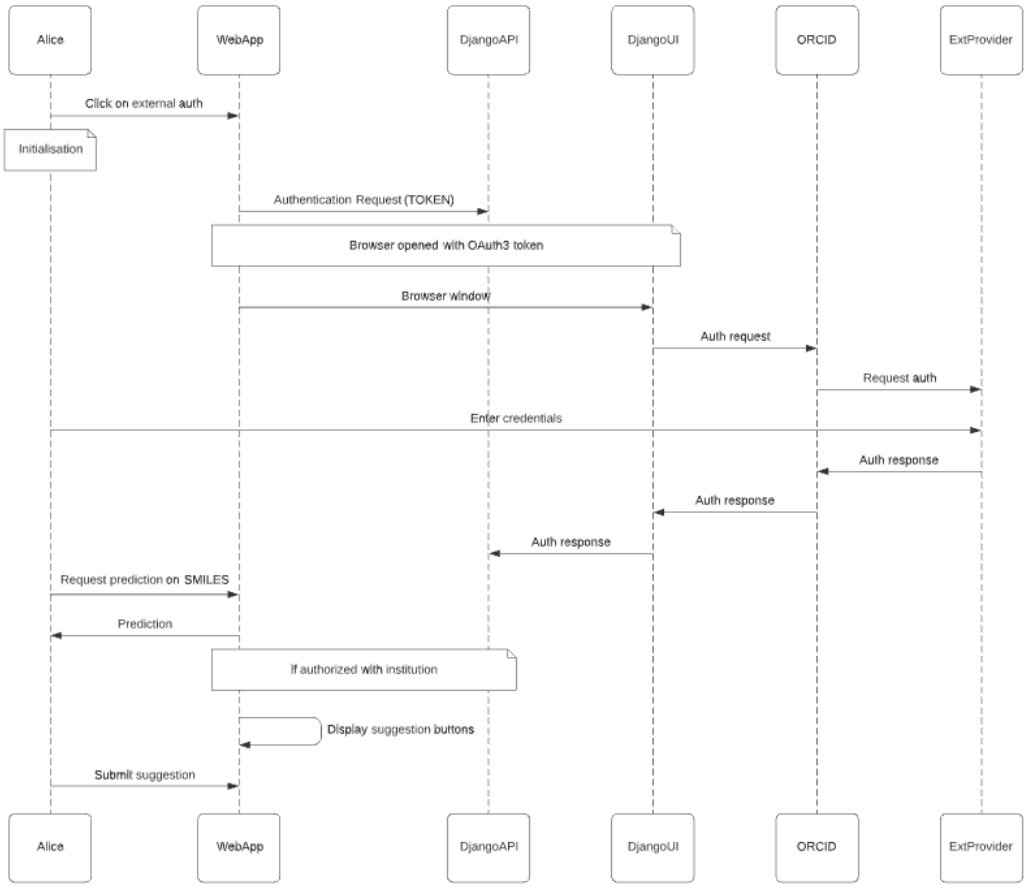
Webové rozhranie pozostáva len zo 4 hlavných častí, a to pole pre SMILES na predikciu, samotný výstup na predikciu kde autorizovaný používateľ vie validovať predikciu, systém na autorizáciu a systém pre zobrazenie molekuly a jej základných chemických deskiptorov pre daný SMILES kód.

Na vizualizáciu molekúl využijeme napojenie na službu ChemAxon, a na autentifikáciu využijeme API nášho backendu Djanga, a prihlásenie prebehne cez inštitúciu aby sme vedeli spojiť validácie predikcií s osobou z akademickej inštitúcie. Priebeh autentifikácie je znázornený na Obr 5.

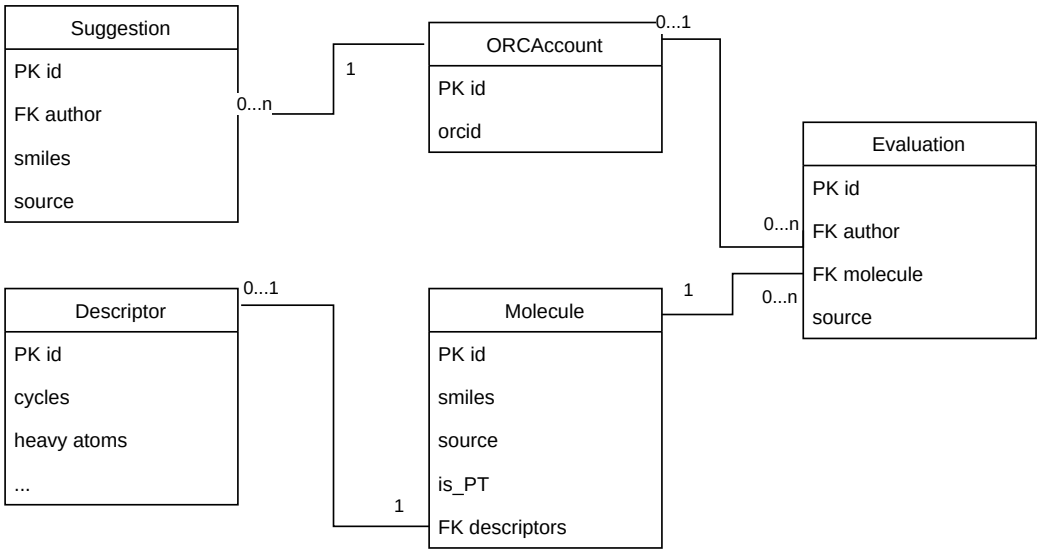
Naša predikčná služba príjma požiadavky cez REST. Ak už bola predtým hodnota predikovaná pre daný vstup, vrátime rovno predošlú predikciu ak je ID modelu rovnaká ako v predikcií. Vieme vrátiť nielen istotu predikcie ale aj koľko používateľov validovalo predikciu.

4.3.1 Metodika predikcie

Ako prvý krok skontrolujeme či nemáme látku uloženú v tabuľke Molecule (vid obrázok). Pokiaľ áno, vrátime fototoxicitu spolu so zdrojom (3T3 test, 3D test, In silico) a mierou istoty.



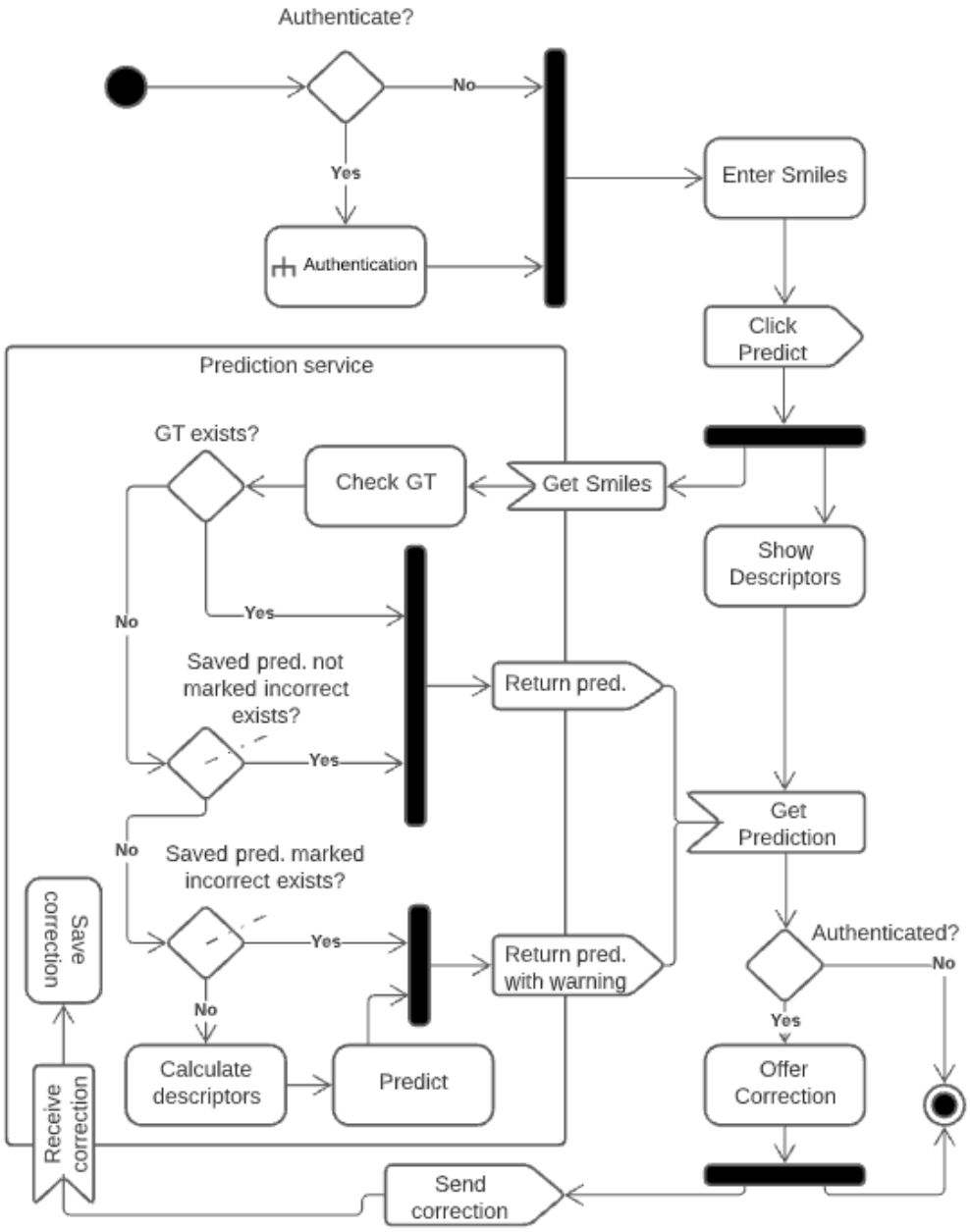
Obr. 5: Overovanie identity používateľa



Obr. 6: Štruktúra dátového úložiska pre webovú aplikáciu

V prípade chýbajúceho záznamu sa posiela SMILES kód do komponentu pre výpočet chemických deskriptorov, ktoré model využíva, a tie potom pošleme do modelu pre predikciu fototoxicity. Odpovedáme web-aplikácií so samotnou binárnou predikciou, a aj percentuálnou istotou.

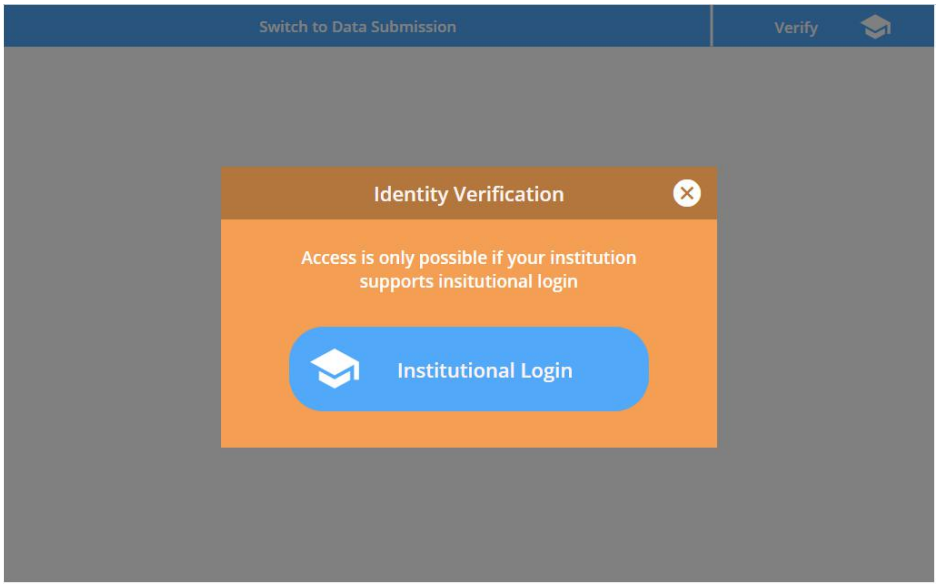
Web-aplikácia je štrukturovaná tak, aby sa automaticky nasadzovali modely v momente ak dáta o validovaných predikciách prekročia určité metriky. Vyberá sa vždy najlepší model, ale všetky modely, spolu s ich ID ukladáme.



Obr. 7: Diagram aktivít

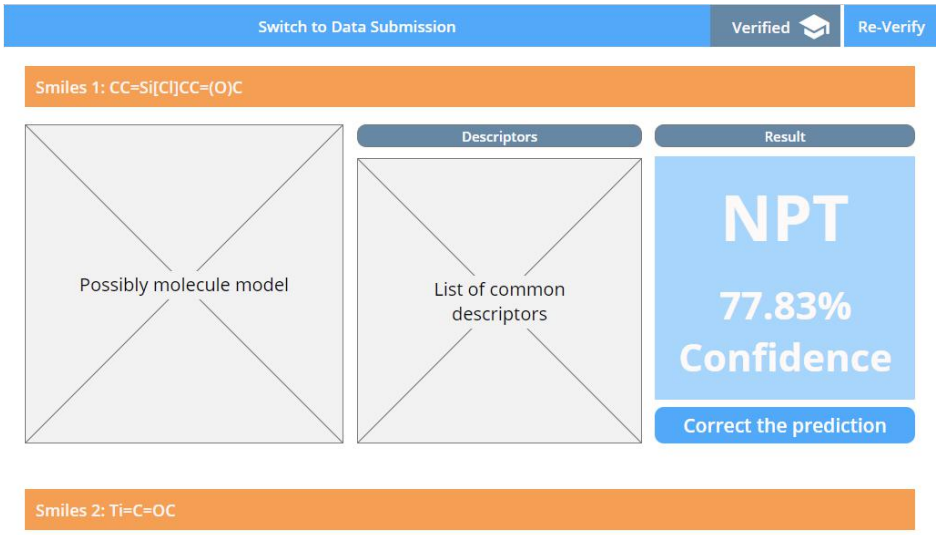
4.3.2 Obrazovky webovej aplikácie

Obr. 8: Landing page stránky, výber medzi predikciou a posielaním dát.

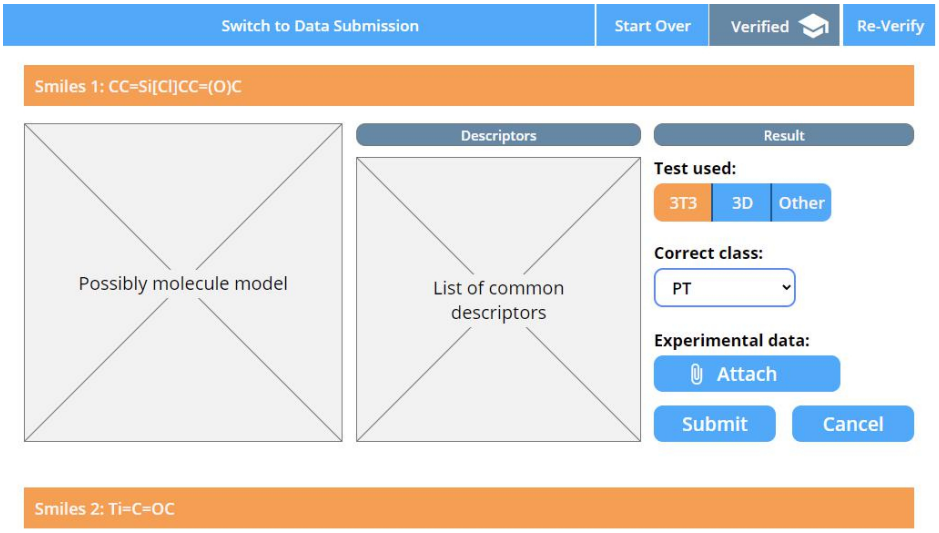


Obr. 9: Overovanie identity používateľa, pri predikcii nieje potrebná ak nechce korigovať výsledky.

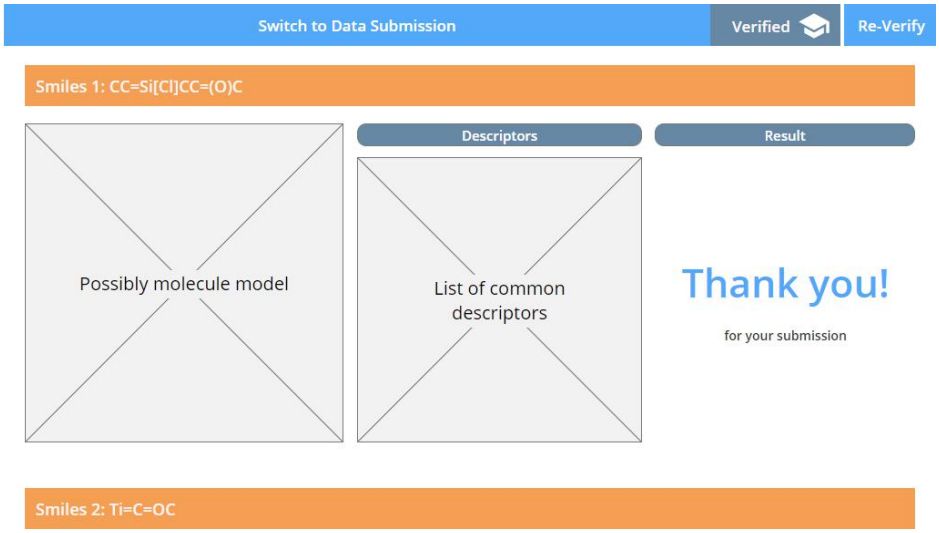
Obr. 10: Vkladanie SMILES kódov pre predikciu, možnosť vkresliť štruktúru molekuly.



Obr. 11: Viditeľný model molekuly, pár základných deskriptorov a výsledok predikcie. Možnosť opravy predikcie ak je používateľ overený.

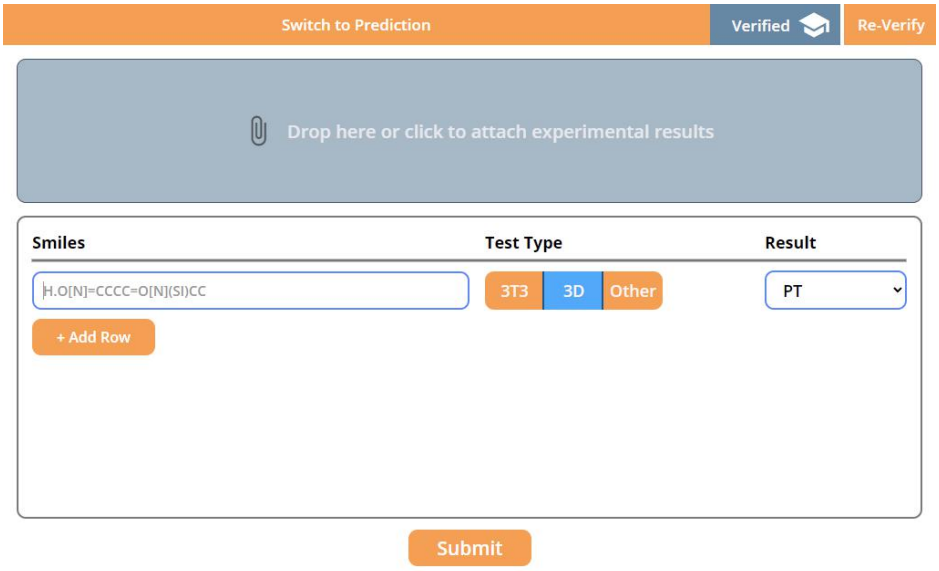


Obr. 12: Používateľ vie zadať typ testu, výsledok a pridať experimentálne dáta na opravu.

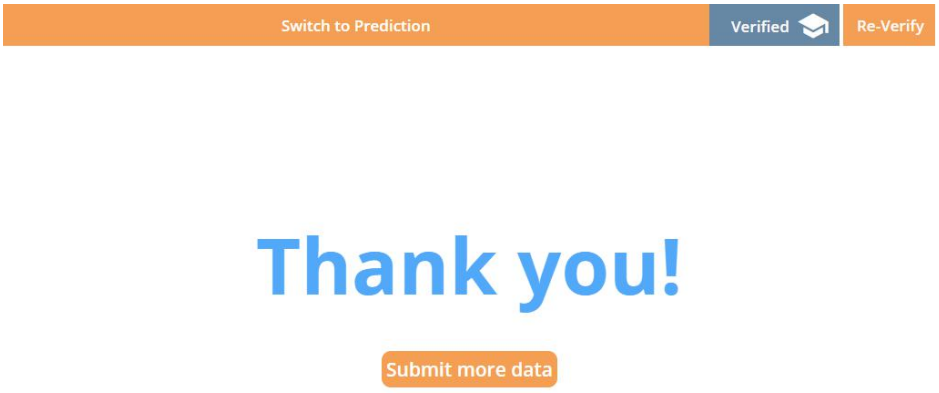


Obr. 13: Potvrdenie korekcie predikcie.

Obr. 14: Pre posielanie dát je nutné byť overený.



Obr. 15: Používateľ vie zadať ľubovoľný počet riadkov výsledkov, a pridať súbor s experimentálnymi dátami.



Obr. 16: Potvrdenie poslania dát.

Časť II:

Riadenie projektu

1 Úvod

V tejto časti si bližšie priblížime informácie o riadení projektu. Pozrieme sa na to, akým spôsobom sme organizovali riadenie úloh, aké techniky a prostriedky sme použili na riadenie a realizáciu projektu.

V tejto časti dokumentácie sa nachádzajú informácie o rolách členov tímu, aké manažérske pozície členovia tímu zastávali a aký je ich podieľ práce na vytvorenom diele, dokumentácii a metodikách. Následne sa opisujú aplikácie jednotlivých manažmentov v projekte, ktoré sa odkazujú na príslušné metodiky. Súčasťou riadenia projektu je aj globálna retrospektíva a sumarizácia šprintov, v ktorých sa uvádza pokrok tímu na projekte.

2 Role členov tímu a podieľ práce

2.1 Role členov tímu

Meno a Priezvisko	Rola
Matej Halinkovič	Architect & ML Engineer
Jakub Knánik	QA Engineer
Jakub Maruniak	Data Scientist & ML Engineer
Kateřina Mušková	Data Scientist
Patrik Pavlačka	Web & ML Engineer
Tibor Sloboda	Domain Knowledge Expert

- **Matej Halinkovič**

  Návrh ML modelov, návrh ML experimentov, návrh architektúry Data pipe, programovanie crawlerov chemických databáz

- **Jakub Knánik**

  Manažment kvality. Code review modulu Data pipe.

- **Jakub Maruniak**

  Rešerš dostupných zdrojov, analýza vhodnosti a extrakcia chemických látok z vedeckých publikácií

- **Kateřina Mušková**

  Analýza a čistenie dát (EDA), časť Cleaner

- **Patrik Pavlačka**

  Vytvorenie a nasadenie webovej stránky tímu a jej pravidelné aktualizovanie

- **Tibor Sloboda**

  Analýza, interpretácia a counsel pre doménu, výber a implementácia výpočtu chemických deskriptorov, zloženie data pipeline, spravovanie Gitu + merge, a architektúra MLOps

2.2 Podieľ práce

Šprint 1

Meno a Priezvisko	Percentuálny podiel
Matej Halinkovič	17%
Jakub Knánik	20%
Jakub Maruniak	16%
Kateřina Mušková	11%
Patrik Pavlačka	23%
Tibor Sloboda	13%

Šprint 2

Meno a Priezvisko	Percentuálny podiel
Matej Halinkovič	17%
Jakub Knánik	19%
Jakub Maruniak	17%
Kateřina Mušková	17%
Patrik Pavlačka	14%
Tibor Sloboda	16%

Šprint 3

Meno a Priezvisko	Percentuálny podiel
Matej Halinkovič	11%
Jakub Knánik	17%
Jakub Maruniak	18%
Kateřina Mušková	12%
Patrik Pavlačka	19%
Tibor Sloboda	23%

Šprint 4

Meno a Priezvisko	Percentuálny podiel
Matej Halinkovič	22%
Jakub Knánik	14%
Jakub Maruniak	15%
Kateřina Mušková	21%
Patrik Pavlačka	12%
Tibor Sloboda	16%

Šprint 5

Meno a Priezvisko	Percentuálny podiel
Matej Halinkovič	16%
Jakub Knánik	13%
Jakub Maruniak	16%
Kateřina Mušková	21%
Patrik Pavlačka	20%
Tibor Sloboda	14%

Práca na milníku

Autor	**Časti milníka**
Matej Halinkovič	Celkový pohľad na systém, Moduly systému, Role členov tímu a podiel práce, Manažment a metodológia správy úloh, Manažment a metodológia dátovej vedy
Jakub Knánik	Úvod - Inžinerske dielo, Globálne ciele projektu, Úvod - Riadenie projektu, Manažment a metodológia kvality, Manažment a metodológia verziovania projektu, Prihláška na TP Cup
Kateřina Mušková	Globálna retrospektíva, Sumarizácia šprintov, Manažment a Metodológia komunikácie, Manažment a Metodológia ukladania informácií
Tibor Sloboda	Moduly sytému, Používané chemické deskriptory

3 Aplikácie manažmentov

3.1 Manažment kvality

Súčasťou projektu je aj zabezpečovanie kvality kódu, a to prostredníctvom dodržiavania zaužívaných štandardov a nami ustanovených pravidiel. Vzhľadom na povahu projektu a možností jeho ďalšieho rozširovania si myslíme, že štandardy kvality prispejú k lepšej a rýchlejšej pochopiteľnosti kódu. Takisto udržiavaním kvality cielime na modulárnosť, ktorá pomôže k dosiahnutiu nižšej komplexnosti kódu a lepšej schopnosti rozširovania funkcionalít systému.

Na overenie dodržanej kvality slúži code review, počas ktorého sa posudzuje kvalita kódu z hľadiska chybovosti a dodržania štandardov. Konkrétne informácie o písaní kódu podľa prijatých pravidiel možno nájsť v prílohe A Metodika zabezpečovania kvality kódu.

3.2 Manažment testovania

Testovanie softvéru je jednou z nevyhnutných zložiek zabezpečovania kvality softvéru, a preto určité zaužívané spôsoby testovania aplikujeme aj pre tento projekt. Cieľom testov je nielen odhaľovanie už prítomných defektov, ale aj zabezpečovanie správnosti funkcionalít. Keď sa kód určitej funkcie pokrytej testom zmení, test by mal byť schopný odhaliť, či zmena kódu nespôsobila chybu vo funkčnosti daného kódu.

V projekte aplikujeme nasledujúce druhy testov: jednotkové testy, ktorými chceme overiť správnosť jednotlivých funkčných jednotiek kódu, integračné testy, ktorými overujeme interakcie medzi funkčnými jednotkami aplikácie, systémové testy, ktorými testujeme správnosť chodu celého vyvíjaného systému, end-to-end a aplikačné testy, ktorými validujeme vytvorené riešenie. Súčasťou projektu je tak štrukturálne (white-box) ako aj funkcionálne (black-box) testovanie.

Je vysoko odporúčané testovať inkrementálne, keďže vieme problémy zachytiť postupne a v skorších fázach vývoja. Pravidlá a konkrétnosti implementovania testovania v projekte možno nájsť v prílohe B.

3.3 Manažment verziovania projektu

Stav jednotlivých častí projektu môže byť rôznorodý, a preto je vhodné používať systém verziovania projektu, ktorý nám umožní zachovávať si rôzne verzie projektu. Takto budeme môcť pracovať paralelne na viacerých funkcionalitách a podsystémoch, čím zvýšime našu efektivitu.

Pre potreby verziovania používame distribuovaný systém riadenia revízií Git, ktorý je napojený na online nástroj pre verziovanie kódu Github. Konkrétne informácie o štruktúre projektu a jeho verziovaní možno nájsť v prílohe C Metodika verziovania kódu.

3.4 Manažment správy úloh

Pre vytváranie, sledovanie a rozdelenie úloh medzi členov tímu využívame nástroj Jira. Má rozsiahlu funkcionalitu, ktorá nám zjednodušuje udržiavanie prehľadu o tom, ktorý člen tímu pracuje na čom, sledovať ako napredujeme v jednotlivých častiach projektu. Presná metodika našej práce s týmto nástrojom je popísaná v prílohe D.

3.5 Manažment dátovej vedy

Náš projekt sa z veľkej časti zamierava na dátovú vedu, pre ktorú klasický Agile prístup nie je ideálny. Z tohto dôvodu sme naše postupy rozšírili o metodológiu CRISP-DM, ktorá je bližšie popísaná v prílohe E.

3.6 Manažment komunikácie

Keďže zadaný projekt nie je individuálny, ale skupinový, je nutné sa dohodnúť na spôsobe komunikácie. Bolo nutné zvoliť príslušné kanály, a spôsob akým budú použité. Keďže sa v novembri zhoršila pandemická situácia, všetka komunikácia sa presunula do online prostredia, čím manažment komunikácie nabral na dôležitosti. Podrobnejšie inrormácie o zvolenej metodológii možno nájsť v prílohe G

4 Globálna retrospektíva

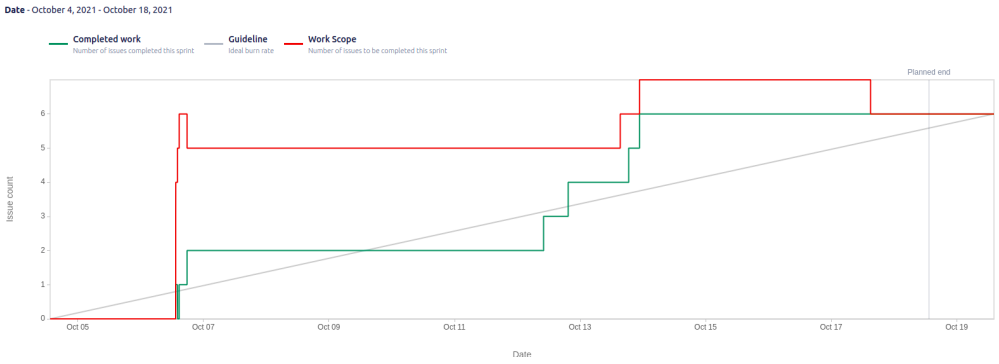
Na začiatku cesty nášho projektu sa bolo treba najskôr zorientovať v téme fototoxicity. Okolo chémie sa točila diskusia nielen v tíme, ale aj našich stretnutiach s vedúcou tímu. Po aspoň čiastočnom zorientovaní v odbore sme začali vytvárať prvé modely, ktoré sme menili v nadväznosti na ďalšie zistenia a nové dáta.

Keďže by aplikácia mala byť pohodlne použiteľná užívateľom, je súčasťou nášho projektu aj vybudovanie webovej aplikácie, ktorou sme sa začali zaoberať v treťom šprinte. Zatiaľ sme sa sústredili predovšetkým na návrh kvalitnej architektúry s možnosťou autentifikácie. Podrobnejší prehľad činností je potom spísaný v nasledujúcej kapitole 5.

5 Sumarizácie šprintov

Šprint č. 1

Cieľom nášho prvého šprintu bolo v prvom rade oboznámiť sa s problematikou fototoxicity, aby sme dobre chápali dátam s ktorými budeme pracovať, t.j. formy zápisov a reprezentácie molekúl pomocou rôznych deskriptorov. Preštudovali sme poskytnutú literatúru, a tam kde to bolo možné extrahovali dostupné chemické látky, relevantné k nášmu problému. Z nich sme pomocou webových databáz chemických látok získali deskriptory, nad ktorými sme vykonávali prieskumnú analýzu. Samozrejme, okrem práce s dátami sme spojazdnili webovú stránku s prezentáciou nášho projektu, nasetupovali GitHub, JIRA a Confluence projekty.



Obr. 17: BurnUp graf šprintu č.1

Šprint č. 2

V rámci tohto šprintu bola zavedená implementácia rozvinutých modelov pre predikciu a vyriešenie problému získavania nových dát. Podarilo sa nám implementovať nové systémy, ktoré nám do budúcnosti ušetria viac času. Pokročili sme vo všetkých smeroch, či už v rámci riadenia projektu, ale aj samotného vývoja - data engineering a modelling. Začali sme tiež plánovať štruktúru finálnej webovej aplikácie. Zároveň sme narazili na niekoľko problémov, primárne stále zápasíme s dátovou doménou, či už s množstvom dát, prístupu k nim, alebo vhodnosťou rôznych deskriptorov.

Obr. 18: BurnUp graf šprintu č.2

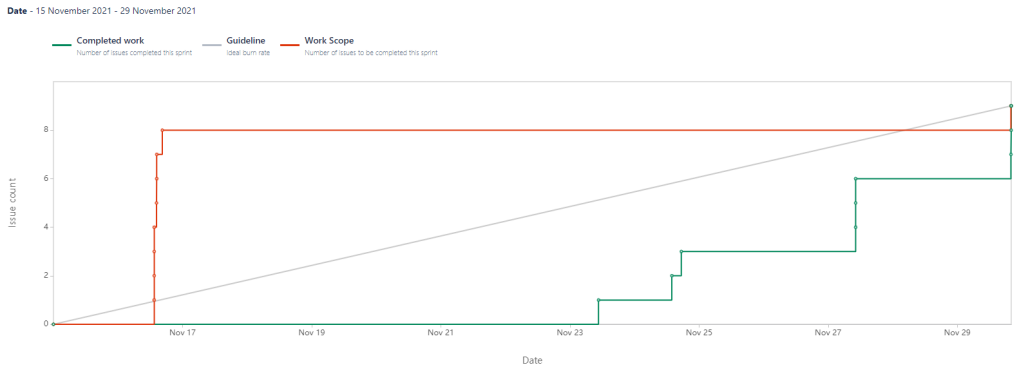
Šprint č. 3

Hlavným cieľom tohto šprintu bolo vytvoriť prototyp webovej aplikácie pre ML Model Deployment a extrahovať ďalšie dáta z článkov a databáz. Začali sme pracovať aj na dokumente Míľnik 1 (prvé 3 šprinty) a návrhu architektúry webovej aplikácie. Počas tohto šprintu sme mali aj hosťovanú prednášku s Dr. Helenou Kanďárovou. Dozvedeli sme sa nové informácie o fototoxicite, aké sú charakteristické vlastnosti fototoxických látok, kedy je potrebne testovať fototoxicitu látky, aké testy sa v súčasnosti robia na fototoxicitu a zvalidovali sme nami vybrané deskriptory.

Obr. 19: BurnUp graf šprintu č.3

Šprint č. 4

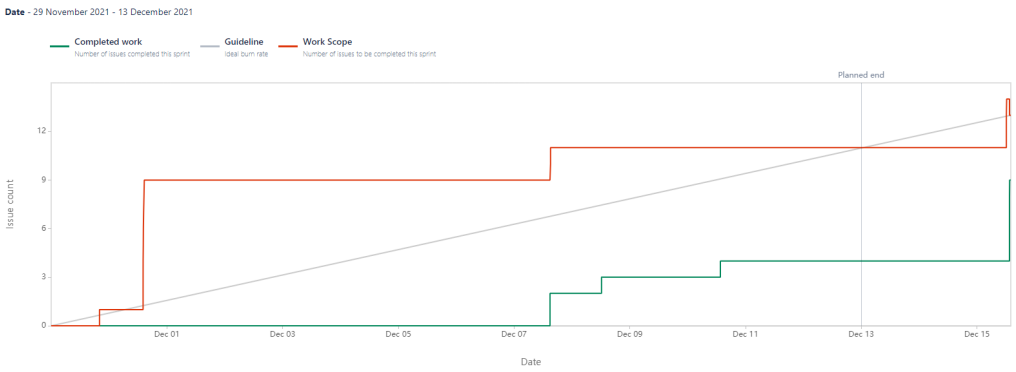
Prioritou tohto šprintu bolo spísanie prvého miľníku, čo sa odráža aj v úlohách z tohto šprintu. Okrem písania samotného dokumentu však z tvorby miľníku vyplynuli aj úlohy na dizajn architektúry pre našu budúcu webovú aplikáciu. Identifikovali sme potrebné časti ako ML pipe pre aplikáciu, databáza predikcií a rozhranie pre overovanie identity používateľa. Architektúru daných častí sme následne navrhli príslušnými diagramami. Tiež sme ďalej pokračovali v zbieraní ďalších údajov o fotoxicite látok.



Obr. 20: BurnUp graf šprintu č.4

Šprint č. 5

Riešili sme spôsob validácie používateľov, návrh backendu a používateľského rozhrania. Zároveň sme skúmali možnosti, ako prepojiť Django s JavaScriptom. V druhej časti šprintu sme navštívili chemický ústav na SAV. Pri exkurzii sme sa zoznámili s experimentami, pri ktorých sa zisťuje fototoxicity a určuje absorpčné spektrum látok, ktoré s fototoxicitou súvisia.



Obr. 21: BurnUp graf šprintu č.5

Prílohy

A Metodika zabezpečovania kvality kódu

Cieľom tejto metodiky je zadefinovanie si sady pravidiel o písaní kódu v Pythone. Pre tieto účely môžeme použiť štandardy PEP ako aj niekoľko vlastných pravidiel. Pri písaní kódu sa následne vieme riadiť uvedenými pravidlami, ktoré sa ďalej kontrolujú pri posúdení kódu.

Pravidlá písania kódu

Pre posúdenie dodržania štandardov a zachovanie kvality Python kódu vieme použiť nástroj na statickú analýzu kódu PyLint. Štandard, ktorý PyLint požaduje aby sa dodržiaval, je PEP 8. Okrem PEP 8 sa však vyžaduje dodržiavanie ďalších pravidiel uvedených v tejto metodike. V prípade, ak nejaké pravidlo nie je uvedené v tomto dokumente, ale definuje ho štandard, je vysoko odporúčané sa riadiť daným štandardom. Táto metodika slúži primárne ako návod, avšak špecifické prípady si môžu vyžadovať výnimky (ich opodstatnenie sa posúdi počas code review).

Dĺžka riadku

- Maximálna dĺžka riadku je podľa PEP 8 79 znakov. PyLint tento limit rozšíril na 100 znakov, čo považujeme za vhodnejší limit, a preto ho

použijeme aj v našej metodike.

- V prípade prekročeného limitu dĺžky riadku treba dodržať zásady odsadenia.

Odsadenie, prázdne riadky a medzery

- Na konci riadku by nemali byť zbytočné medzery. Taktiež je potrebné sa vyvarovať zbytočným znakom medzery vnútri kódu (napr. medzi názvom funkcie a zátvorkou s argumentami), obsiahlo túto časť dokumentuje PEP 8.

- Medzi funkciami treba mať 2 riadky voľné. Na konci súboru má byť práve jeden prázdny riadok. Logické skupiny importov modulov majú byť oddelené prázdnym riadkom (viď. nižšie). Ak za importami nasleduje definícia globálnych premenných, prípadne nastavenia (napr. logovanie), pridáva sa za importy jeden prázdny riadok, v opačnom prípade dva prázdne riadky.

Pomenovávanie premenných, funkcií, tried atď.

- Všetky vlastné pomenovania musia byť písané v anglickom jazyku.

- V prípade premenných treba dodržiavať snake_case. Niektoré premenné môžu pravidlá písania daným štýlom narúšať (napr. df - dataframe). Takéto prípady patria medzi výnimky a sú povolené, musí však ísť o všeobecne známe a používané názvy.

- Pre moduly (Python súbory) a balíky je vhodné používať malé písmená. Ideálne by mali byť tieto názvy čo najkratšie (jednoslovné). Ak je to nutné, je možné používať snake_case. Pridaním znaku _na rozdelenie slov vieme dosiahnuť lepšiu čitateľnosť a priamejšie pomenovanie.

- Názvy tried by mali používať konvenciu CapWords.

- Používané názvy by mali výstižne pomenovávať ich zmysel a funkcionalitu. Je lepšie názov rozpísať, než používať neznámu skratku, ktorej význam si nikto nedomyslí.

Nápoveda dátového typu

- Cieľom tohto pravidla je využitie anotácie dátových typov, čím sa Python kód stane prístupnejším pre statickú analýzu kódu a refaktoring. Aplikujú sa pravidlá smerníc PEP 484 a PEP 3107.
- Nápovedy je nutné definovať spôsobom:
  - parametre: def foo(a: expression, b: expression = 5): .
  - návratové hodnoty: def sum() -> expression: .

Import modulov

- Importy musia byť vždy vložené na začiatok modulu, ale až po komentároch pre dokumentáciu.
- Poznáme tri druhy importov. V module musia byť importy rovnakého druhu zoskupené spolu a oddelené prázdnym riadkom. Konkrétne ide o tieto typy importov (treba dodržať aj poradie typov importov):
  - importy zo štandardnej knižnice
  - importy zo súvisiacich knižníc tretích strán
  - importy z lokálneho prostredia
- Nepoužívané importy treba vymazať.

Refaktoring

- V prípade, ak sa opakuje niektorá funkcionalita v kóde, je potrebné sa zamyslieť nad možnosťou refaktorovania kódu. Medzi takéto možnosti môže patriť vypichnutie opakujúcej sa časti kódu a jej oddelenie v samostatnej funkcii či rozdelenie komplikovanej funkcie na menšie časti.
- Pomocou refaktoringu môžeme docieliť nielen lepšiu čitateľnosť, ale aj znovu-použiteľnosť. V snahe docieliť modularizáciu vieme kód rozpísať do funkcií či tried.
- Treba dodať, že nie je nutné za každú cenu refaktorovať. Dôležité je, aby jednotlivé časti kódu (funkcie, triedy) dávali zmysel a nevenovali sa príliš veľkému počtu funkcionalít.

Dokumentácia v kóde

- Štandardy PEP 8 a PEP 257 opisujú pravidlá písania dokumentácie v Python kóde.
- Pre dokumentáciu sa používajú dokumentačné reťazce (docstrings), majúce tvar troch dvojitých úvodzoviek.
- Napísanie dokumentácie je potrebné pre moduly, funkcie, triedy a ich metódy.

Ostatné pravidlá

- Textové reťazce budú definované jednotne, a to s použitím jednoduchých úvodzoviek v prípade identifikátorov a dict kľúčov a dvojitých úvodzoviek v ostatných prípadoch. V prípade, ak string obsahuje dvojitú úvodzovku, treba použiť na definovanie reťazca jednoduché úvodzovky a opačne, aby sa zabránilo zbytočnému používaniu escape znakov.
- Spustiteľný súbor musí mať definovanú časť *if __name__== "__main__":*
- Formátovanie v logovaní nemusí používať lazy formatting (%), aj keď ho odporúča PyLint. Lazy formatting sa síce v PEP 282 spomína, no nie je uvedený ako štandard.
- V prípade, ak chce programátor použiť v jazyku Python konštrukciu podobnú rozhraniu pre triedu, mal by vytvoriť abstraktnú triedu podľa smernice PEP 3119.
- Čokoľvek nepoužité (premenná, funkcia, trieda, modul ...) by malo byť vymazané, prípadne treba v komentári vysvetliť, prečo daný kód nevymazať.

B Metodika testovania

Súčasťou vyvíjaného produktu je aj jeho testovanie a validácia. Pre tento projekt boli pre účely testovania vybrané nasledujúce druhy testov: jednotkové, integračné, systémové, end-to-end/akceptačné testy. V tejto metodike možno nájsť konkrétne informácie týkajúce sa implementovania testov do projektu.

Jednotkové testy

Prostredníctvom jednotkových testov vieme zistiť správnosť fungovania určitej ucelenej časti kódu (funkcia, trieda, metóda v triede ...). Jednotkové testy by mal prednostne písať autor danej časti kódu, prípadne sa napíšu dodatočne v procese QA podľa potreby. Zoznam pravidiel pri písaní testov:

- Python: testy sa musia nachádzať v balíku tests.
- Python: konvencia názvoslovia testovacích modulov postupuje podľa prijatých pravidiel písania kódu (teda *snake_case*), každý názov testovacieho modulu ale musí začínať na *test_*, teda napr. *test_your_module.py*.
- Python: jednotkové testy sa vykonávajú pomocou knižnice *pytest*.
- Princíp AAA (arrange, act, assert). Ak je to možné, je odporúčané dodržiavať tento princíp. Najskôr si pripravíme všetko, čo k vykonaniu testu potrebujeme (arrange). To môže byť vytvorenie inštancie objektu, naplnenie premenných objektu nejakými hodnotami a podobne. Následne vykonáme funkcionalitu, ktorú chceme testovať (act), čo môže byť napr. zavolanie metódy. Nakoniec chceme porovnať očakávaný výsledok s reálnym výsledkom (assert), čím si overujeme hypotézu daného testu.
- Jeden test by mal testovať práve jednu hypotézu. Test sa tak musí venovať jednej konkrétnej funkcionalite, aby sa zabránilo nedorozumeniam v interpretácii výsledkov. Odporúčaný je jeden assert na test, nie je to však nutné, pokiaľ sa asserty venujú rovnakej hypotéze.
- Každý test by mal mať svoju krátku dokumentáciu, aby bolo jasné, čo sa testuje a aký výsledok sa očakáva.

C Metodika verziovania kódu

Na kolaboratívnu prácu na projekte sa používa nástroj na verziovanie kódu Github. V rámci organizačnej štruktúry máme pridelený priestor organizácie STU-FIIT-Team-20-2021. V danej organizácii si vytvoríme súkromné repozitáre, ktoré odzrkadľujú jednotlivé logické a organizačné celky projektu. Súčasťou projektu sú tieto Github repozitáre:

- website - súbory súvisiace so statickou verziou webového sídla tímu
- data-pipeline - kód zaoberajúci sa získavaním a čistením dát
- webapp - webová aplikácia, s ktorou bude interagovať používateľ
- phototox-analysis - venuje sa analýze získaných dát
- chem-descriptor-evaluator - vyhodnocovanie chemických deskriptorov zo SMILES kódov

Súčasťou repozitárov sú vetvy, ktoré umožňujú bezpečné vyvíjanie nových funkcionalít či opravovanie chýb. Hlavnú vetvu pomenúvame *master*, prípadne *main*. Vedľajšie vetvy sa následne pomenúvajú podľa ich významu a dôvodu vzniku. Dôležité je udržiavanie správnosti *master* vetvy - nedokončené či neoverené funkcionality je vhodné najskôr vyriešiť v samostatnej vetve a až potom urobiť zlúčenie vetiev.

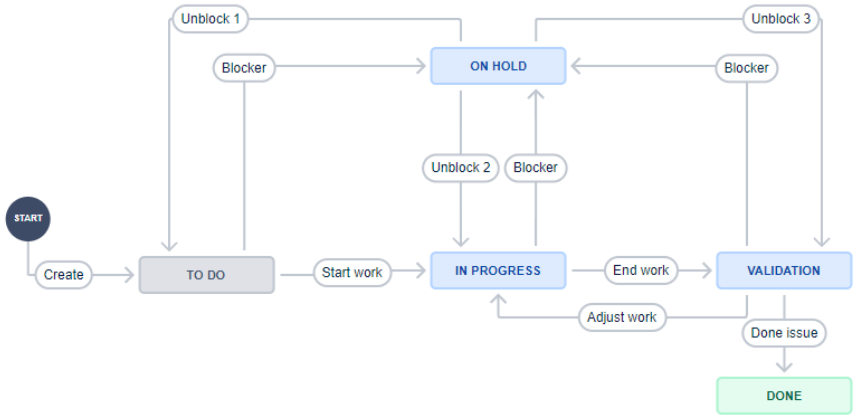
D Metodika správy úloh

Pre správu a sledovanie úloh v našom projekte používame nástroj Jira od spoločnosti Atlassian. Umožňuje nám jednoducho evidovať úlohy, ktoré sú potrebné spraviť v danom šprinte a kto je za ne zodpovedný. Tento nástroj poskytuje rozsiahlu funkcionalitu a značne nám zjednodšuje udržiavanie prehľadu nad vykonávanou prácou.

Funkcionalitu nástroja sme si upravili tak, aby vyhovovala naším potrebám a vytvorili sme nasledujúce rámce používania, ktoré zabezpečujú čo najplynulejšiu spoluprácu a umožňujú každému členovi tímu zostať v obraze.

D.1 Workflow

Predvolený workflow v Jira je pre naše potreby príliš jednoduchý. Pozostáva iba z 3 základných častí TO DO → IN PROGRESS → DONE. V našej metodike používame ešte dva kroky navyše.



Obr. 22: Workflow v Jira

- **TO DO** - Úloha je vytvorená a pridelená členovi týmu, ale ten na nej ešte nezačal pracovať.
- **IN PROGRESS** - Práca na úlohe prebieha.

- **ON HOLD** - Práca na úlohe bola zablokovaná vzniknutým problémom. Zväčša znamená čakanie na dokončenie inej úlohy.
- **VALIDATION** - Po ukončení práce na úlohe musí prejsť validáciou, ktorej cieľom je overiť či vykonaná práca skutočne splnila požiadavky.
- **DONE** - Uzavretá úloha.

D.2 Typy úloh

Jira poskytuje rôzne úrovne granularity úloh a umožňuje začleňovať úlohy do logických skupín. Jedna z možností sú napríklad Story points, užitočné pre Agile vývoj. Avšak prostredie Jiry nie je pre využitie Story points intuitívne, preto sa spoliehame primárne na Tasks, Sub-Tasks a Epics. Do Epics začleňujeme všetky úlohy a to podľa nasledujúceho kľúča:

- **Paperwork** - Administratívne úlohy ako záznamy so stretnutí, retrospektívy šprintov a písanie milníku
- **Data engineering** - Získavanie nový dát zo zdrojov, vývoj všetkých častí data pipeline
- **Modelling** - Experimentálny vývoj modelov, pretrénovanie existujúcich modelov na nových dátach a podobne
- **Web** - Úlohy týkajúce sa ako údržby tímového webu, tak aj vývoju webovej aplikácie

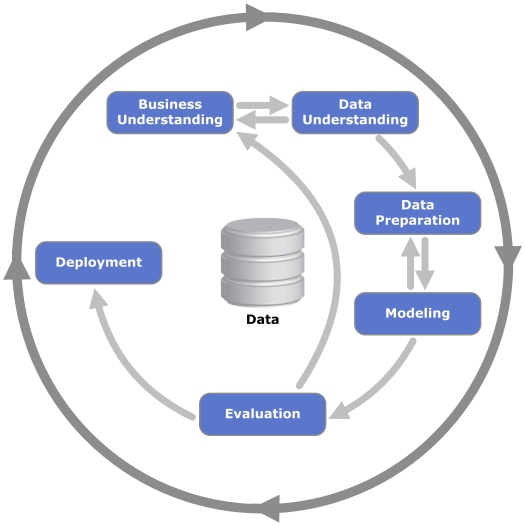
D.3 Role

Rola	**Zodpovednosti**
Majiteľ	Vytváranie úloh Písanie popisu úlohy Určí kto je za úlohu zodpovedný Validuje úlohu Môže aktualizovať status úlohy
Zodpovedný	Zodpovedný za prácu na úlohe Komunikuje s Majiteľom a Podporou Môže delegovať časť práce pre Podporu Aktualizuje stav úlohy
Podpora	Podporuje Zodpovedného programovaním, diskusiou, zosúladením s inou úlohou, ... Neaktualizuje status úlohy Komunikuje hlavne so Zodpovedným Nie je zodpovedný za úlohu

D.4 Komunikácia

Aktualizácia	**Obsah**
Povinná	Pred každým stretnutím Pri presune úlohy do stavu ON HOLD - treba špecifikovať čo úlohu blokuje Neakceptovanie úlohy pri validácií Čo najdetailnejšie popísať čo nebolo splnené.
Dobrovoľná	Inkrementálne aktualizácie keď je úloha v stave IN PROGRESS Možnosť upozorniť (@) člena tímu pri zmene stavu Kedykoľvek to člen tímu uzná za vhodné

E Metodika dátovej vedy



Obr. 23: CRISP-DM

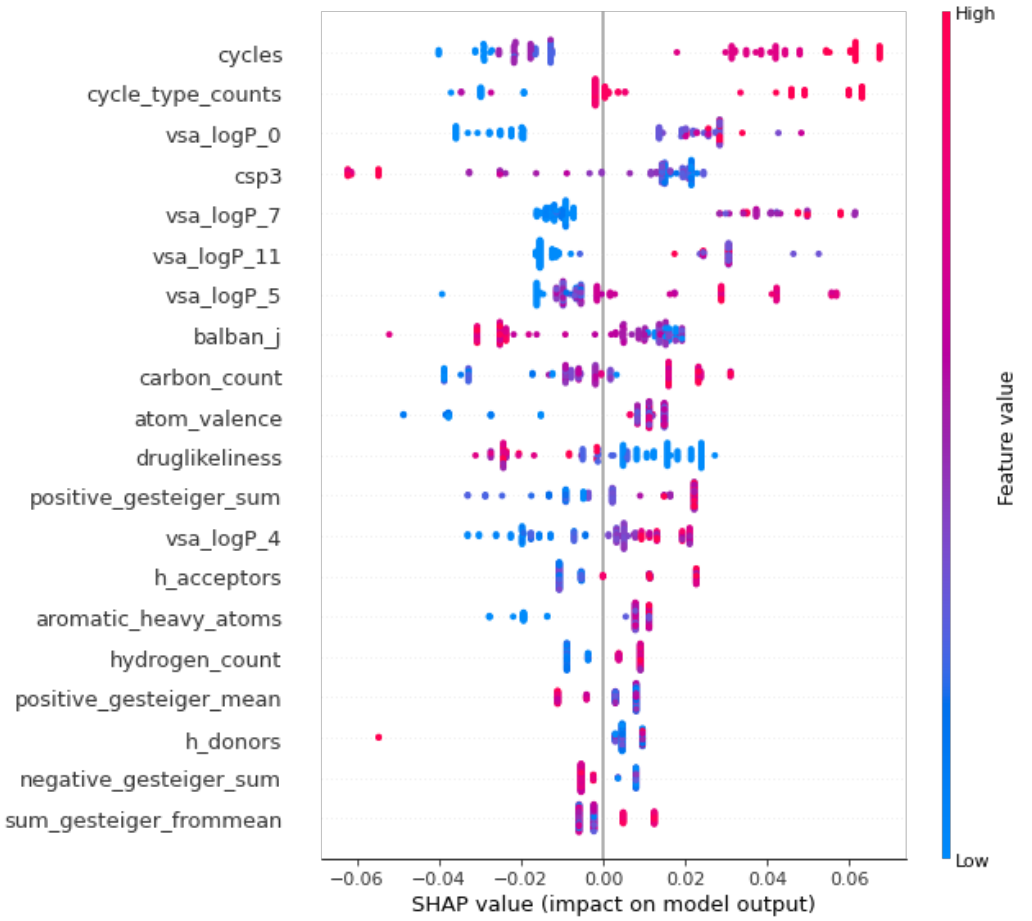
Naša práca z veľkej časti spočíva v dátovej vede a strojovom učení. Táto podskupina softvérového inžinierstva má kvôli podstate svojej činnosti špecifické požiadavky na pracovný postup. V našom tíme sa teda okrem Agile metodológie spoliehame z veľkej časti aj na CRISP-DM (Cross Industry Standard Process for Data Mining).

CRISP-DM popisuje prirodzený životný cyklus, ktorý sa vykryštalizuje vo väčšine projektov dátovej vedy. S naším prehlbujúcim sa porozumením problematike fototoxicity sa prehlbuje aj pochopenie dôležitosti rôznych deskriptorov, ktorými ju môžeme predikovať. To nám umožňuje upravovať spôsob prípravy dát a trénovania modelu tak, aby sme dosahovali čoraz lepšie výsledky. V našej webovej aplikácií nám to umožňuje stále používať najlepší dostupný model, ale zároveň sme pripravený na zmeny a implementáciu vylepšení.

F Používané chemické deskriptory

Na základe štruktúry molekúl látok a ich chemických vlastností, sme identifikovali alebo vypočítali rôzne užitočné deskriptory, ktoré by mohli byť korelované s fototoxicitou látok. Následne sme nad týmito deskriptormi trénovali modely strojového učenia, a analyzovali užitočnosť a teda dôležitosť rôznych deskriptorov pre klasifikáciu látok medzi fototoxické a nefototoxické. Samotná dôležitosť deskriptorov na základe natrénovaného modelu je viedieť vo Figure 24.

Deskriptor	**Popis deskriptoru**
cycles	Počet karbocyklov v molekule
atom_valence	Suma valencie atómov v molekule
gesteiger (rôzne)	Rôzne štatistické hodnoty odvodené z Gesteigerových nábojov atómov, napr. suma pozitívnych a negatívnych nábojov, ich priemery, a pod.
hydrogen_count	Počet vodíkov v molekule
csp3	Pomer sp3 hybridizovaných uhlíkov ku všetkým uhlíkov
aromatic_heavy_atoms	Počet ťažkých atómov v aromatických cykloch
balban_j	Topologický index komplexity molekuly
hallkier	Pomer kovalentných polomerov pre nehybridizované atómy uhlíka
labute_asa	Obsah prístupnej plochy molekuly pre solvent
h_donors/h_acceptors	Vodíkové donory a akceptory
carbon_count	Počet uhlíkov
tpsa	Obsah polárnej povrchovej plochy molekuly
druglikeness	Počet violácií deskriptorov, ktoré identifikujú molekulu ako farmaceutický cieľ
vsa_logP_N	Rôzne atribúty vypočítanej lipofolity molekúl
cycle_type_counts	Log16 hodnôt namapovaných počtov cyklov rôznch veľkostí na jednotky hexadecimálnej sústavy s ofsetom rovným veľkosti karbocyklu



Obr. 24: SHAP dôležitosť deskriptorov. Os X znázorňuje vplyv premennej na rozhodnutie. Záporné hodnoty na tejto osi predstavujú naklonenie ku klasifikovaniu látky ako 'nefototoxická', kladné hodnoty pre modelu zvyšujú šancu klasifikovať ju ako 'fototoxická'. Os Y predstavuje 20 najdôležitejších deskriptorov zoradených zostupne. Farba jednotlivých bodov v grafe predstavuje hodnotu deskriptoru. Teda pre deskriptrov 'cycles' vidíme, že nízke hodnoty tohto deskriptoru považuje model za indikátory nefototoxickej látky a vysoké hodnoty považuje za indikátor fototoxickej látky.

G Metodika komunikácie

Našim primárnym komunikačným kanálom je súkromný discord server. V rámci neho sa nachádzajú tri kanály, a to *general*, *paperwork* a *doc*. Prvý spomínaný slúži na všeobecnú komunikáciu, v druhom sa zhromažďujú materiály k písomným prácam, ktoré vznikajú v priebehu semestra ako napríklad burndown grafy. Posledný spomínaný kanál obsahuje predovšetkým odkazy na užitočné vedecké články vzťahujúce sa k riešenej problematike. Možno tu nájsť aj verejné databázy chemických látok (napr. PubChem, Swiss, ...), alebo OECD materiály. Konverzácia je prevažne štruktúrovaná pomocou vlákien.

Záložným variantom pre prípad núdze je telefonický kontakt, ktorý každý z nás na začiatku poskytol. Doteraz sme ho však nemuseli využiť.

Ďalšie kanály slúžia na komunikáciu s vedúcou nášho tímu. Na tento účel sme zriadili spoločný gmailový účet, cez ktorý sú zdieľané články, alebo komplexnejšie informácie. Pokiaľ je treba rýchlo čokoľvek odkomunikovať, je na to použitá WhatsApp skupina.

Keďže sa od polovice novembra zhoršila epidemiologická situácia v Bratislave natoľko, že sme prešli do dištančnej výučby, nebolo naďalej možné sa prezenčne schádzať. Pri tímových stretnutiach sme naďalej pokračovali na platforme Discord, pri stretnutiach s vedúcimi využívame GoogleMeet.

H Metodika ukladania dokumentov a informácií

Spolu s nástrojom Jira od spoločnosti Atlassian používame aj Confluence. Primárne slúži na ukladanie interných dát a dokumentov. Pod dátami rozumieme súbory obsahujúce fototoxicitu molekúl, vypočítané deskriptory, alebo extrahované descriptory.

Dokumenty sú trojakého druhu. V prvom z nich sú popísané metodológie, a ďalej potom popis architektúry spolu s grafmi. Najdôležitejšou časťou sú súbory so získanými informáciami súvisiacimi s fototoxicitou, ako je chemická podstata, alebo spôsob merania.

I Export úloh

Šprint č.1

Issue Type	Issue key	Issue id	Summary	Assignee	Reporter	Status	Resolution	Created	Updated
Task	PP-35	10034	Feature importances - SHAP	Matej Halinkovic	Matej Halinkovic	Done	Done	13/10/2021 15:44	14/10/2021 18:08
Task	PP-19	10018	Extract predictions from SwissTarget	Matej Halinkovic	Matej Halinkovic	Done	Done	06/10/2021 14:46	14/10/2021 18:08
Task	PP-4	10003	Setup Atlassian products	Tibor Sloboda	Matej Halinkovic	Done	Done	06/10/2021 13:27	06/10/2021 14:46
Task	PP-3	10002	Extract Data from Pubchem	Matej Halinkovic	Matej Halinkovic	Done	Done	06/10/2021 13:26	14/10/2021 18:08
Task	PP-2	10001	Create report template	Katerina Muskova	Matej Halinkovic	Done	Done	06/10/2021 13:26	14/10/2021 18:08
Task	PP-1	10000	Setup website	Patrik	Matej Halinkovic	Done	Done	06/10/2021 13:25	14/10/2021 18:10

Šprint č.2

Issue Type	Issue key	Issue id	Summary	Assignee	Reporter	Status	Resolution	Created	Resolved
Task	PP-64	10063	Jira methodology design	Matej Halinkovič	Matej Halinkovič	Done	Done	31/10/2021 12:16	31/10/2021 12:16
Task	PP-56	10055	Cleaner clustering	Kateřina Mušková	Matej Halinkovič	Done	Done	27/10/2021 14:31	31/10/2021 12:16
Task	PP-51	10050	Populate CSV with Chem. properties (pipeline step)	Tibor Sloboda	Tibor Sloboda	Done	Done	21/10/2021 21:11	26/10/2021 12:29
Task	PP-50	10049	Data pipeline architecture	Matej Halinkovič	Matej Halinkovič	Done	Done	20/10/2021 17:46	27/10/2021 14:52
Task	PP-47	10046	CSV Merger	Tibor Sloboda	Matej Halinkovič	Done	Done	19/10/2021 15:14	26/10/2021 12:29
Task	PP-46	10045	Identify cyclic and acyclic compounds	Tibor Sloboda	Matej Halinkovič	Done	Done	19/10/2021 15:14	21/10/2021 20:50
Task	PP-40	10039	Ensemble learning exploration	Matej Halinkovič	Matej Halinkovič	Done	Done	13/10/2021 22:35	27/10/2021 14:32
Task	PP-39	10038	Deeper EDA - properly identify outliers, distributions, etc.	Kateřina Mušková	Matej Halinkovič	Done	Done	13/10/2021 22:34	27/10/2021 14:52
Task	PP-38	10037	Explore different architectures: Logit, SVM, ...	Matej Halinkovič	Matej Halinkovič	Done	Done	13/10/2021 22:33	27/10/2021 12:40
Task	PP-37	10036	Iteratively improve website	Patrik	Tibor Sloboda	Done	Done	13/10/2021 18:37	26/10/2021 14:30
Task	PP-32	10031	Re-train RandomForest with new data	Matej Halinkovič	Tibor Sloboda	Done	Done	13/10/2021 15:39	27/10/2021 12:40

Task	PP-29	10028	Meeting report 4	Jakub Maruniak	Tibor Sloboda	Done	Done	13/10/2021 15:37	27/10/2021 13:15
Task	PP-27	10026	Meeting report 3	Matej Halinkovič	Tibor Sloboda	Done	Done	13/10/2021 15:35	27/10/2021 12:40
Task	PP-21	10020	Obtain additional data	Jakub Maruniak	Matej Halinkovič	Done	Done	13/10/2021 15:23	19/10/2021 15:10
Task	PP-20	10019	TP Cup application	Jakub Knánik	Matej Halinkovič	Done	Done	10/10/2021 10:10	31/10/2021 12:15
Task	PP-9	10008	Meeting report 2	Jakub Maruniak	Tibor Sloboda	Done	Done	06/10/2021 14:06	19/10/2021 15:10

Šprint č.3

Issue Type	Key	Summary	Assignee	Reporter	Status	Resolution	Created	Resolved
Task	PP-96	Create doxigen	Kateřina Mušková	Tibor Sloboda	Done	Done	13/Nov/21 6:20 PM	16/Nov/21 1:24 PM
Task	PP-54	Data extraction from newspapers	Jakub Maruniak	Matej Halinkovič	Done	Done	27/Oct/21 2:30 PM	16/Nov/21 4:28 PM
Task	PP-57	Data pipeline assembling	Tibor Sloboda	Matej Halinkovič	Done	Done	27/Oct/21 2:31 PM	10/Nov/21 2:47 PM
Task	PP-58	Data pipeline code review	Jakub Knánik	Matej Halinkovič	Done	Done	27/Oct/21 2:32 PM	16/Nov/21 1:11 PM
Task	PP-65	Expand calculated descriptors	Tibor Sloboda	Matej Halinkovič	Done	Done	02/Nov/21 4:00 PM	07/Nov/21 2:10 PM
Task	PP-93	Experiment with unsupervised approach	Matej Halinkovič	Matej Halinkovič	Done	Done	10/Nov/21 3:20 PM	16/Nov/21 1:20 PM
Task	PP-67	Find conditions of the Muegge filter	Jakub Knánik	Tibor Sloboda	Done	Done	03/Nov/21 5:12 PM	03/Nov/21 5:36 PM
Task	PP-68	Japan data extraction	Matej Halinkovič	Matej Halinkovič	Done	Done	09/Nov/21 3:29 PM	16/Nov/21 1:19 PM
Task	PP-60	Meeting report 4	Matej Halinkovič	Matej Halinkovič	Done	Done	27/Oct/21 2:43 PM	02/Nov/21 4:01 PM
Task	PP-61	Meeting report 5	Jakub Knánik	Matej Halinkovič	Done	Done	27/Oct/21 2:43 PM	03/Nov/21 2:54 PM
Task	PP-76	Meeting report 6	Tibor Sloboda	Matej Halinkovič	Done	Done	09/Nov/21 3:53 PM	15/Nov/21 3:01 PM
Task	PP-66	Modify cleaner to create masks instead of deletion	Kateřina Mušková	Tibor Sloboda	Done	Done	03/Nov/21 2:23 PM	10/Nov/21 1:28 PM
Task	PP-63	Modify cleaner to fit data pipeline	Kateřina Mušková	Tibor Sloboda	Done	Done	30/Oct/21 11:51 AM	03/Nov/21 2:25 PM
Task	PP-97	Replace seaborn in cleaner by alternativy	Kateřina Mušková	Tibor Sloboda	Done	Done	14/Nov/21 10:27 AM	14/Nov/21 1:54 PM

Task	PP-53	Setup flask website prototype	Matej Halinkovič	Matej Halinkovič	Done	Done	27/Oct/21 2:29 PM	10/Nov/21 1:02 PM
Task	PP-52	Sprint 1 retrospective	Jakub Maruniak	Matej Halinkovič	Done	Done	27/Oct/21 1:01 PM	02/Nov/21 3:50 PM
Task	PP-62	Sprint 2 retrospective	Matej Halinkovič	Matej Halinkovič	Done	Done	27/Oct/21 2:43 PM	10/Nov/21 1:21 PM
Task	PP-100	Updating website	Patrik	Patrik	Done	Done	16/Nov/21 2:54 PM	16/Nov/21 2:55 PM
Task	PP-88	Validate Data Cleaner on calculated descriptors	Kateřina Mušková	Matej Halinkovič	Done	Done	10/Nov/21 1:28 PM	13/Nov/21 5:40 PM
Task	PP-55	Viability of calculated descriptors	Matej Halinkovič	Matej Halinkovič	Done	Done	27/Oct/21 2:31 PM	07/Nov/21 2:09 PM

Šprint č.4

Issue Type	Key	Summary	Assignee	Reporter	Status	Resolution	Created	Resolved
Task	PP-101	Retrain models with Spielmann data	Matej Halinkovič	Matej Halinkovič	Done	Done	16/11/2021 16:29	27/11/2021 10:13
Task	PP-99	Create diagram for modelling experiments	Matej Halinkovič	Matej Halinkovič	Done	Done	16/11/2021 14:24	24/11/2021 13:54
Task	PP-98	Sprint 3 Retrospective	Patrik	Matej Halinkovič	Done	Done	16/11/2021 13:28	24/11/2021 17:25
Task	PP-92	Reformat existing documents to Latex	Kateřina Mušková	Kateřina Mušková	Done	Done	10/11/2021 14:37	23/11/2021 10:38
Task	PP-90	Doxigen generation	Tibor Sloboda	Kateřina Mušková	Done	Done	10/11/2021 14:31	29/11/2021 19:57
Task	PP-78	Design initial web app architecture for the Milestone	Tibor Sloboda	Matej Halinkovič	Done	Done	09/11/2021 16:01	29/11/2021 20:01
Story	PP-75	Milestone I.	Matej Halinkovič	Matej Halinkovič	Done	Done	09/11/2021 15:51	27/11/2021 10:13
Story	PP-74	Web app architecture	Tibor Sloboda	Matej Halinkovič	Done	Done	09/11/2021 15:38	29/11/2021 20:01
Task	PP-59	Milestone documentation	Matej Halinkovič	Matej Halinkovič	Done	Done	27/10/2021 14:37	27/11/2021 10:13

Šprint č.5

Issue Type	Key	Summary	Assignee	Reporter	Status	Resolution	Created	Resolved
Task	PP-116	Sprint retrospective 5	Matej Halinkovič	Matej Halinkovič	Done	Done	15/12/2021 12:13	15/12/2021 13:32
Task	PP-115	Meeting report 11	Jakub Maruniak	Matej Halinkovič	Done	Done	15/12/2021 12:13	15/12/2021 13:32
Task	PP-114	Milestone II.	Matej Halinkovič	Matej Halinkovič	Done	Done	08/12/2021 16:39	15/12/2021 13:32
Task	PP-113	Meeting report 10	Matej Halinkovič	Matej Halinkovič	Done	Done	07/12/2021 15:07	15/12/2021 13:30
Task	PP-112	Explore Django JavaScript integration	Patrik	Matej Halinkovič	Done	Done	07/12/2021 15:07	15/12/2021 13:31
Task	PP-110	Basic UX design	Tibor Sloboda	Matej Halinkovič	Done	Done	30/11/2021 14:37	15/12/2021 13:30
Task	PP-109	Create database prototype for webapp	Kateřina Mušková	Matej Halinkovič	Done	Done	30/11/2021 14:30	10/12/2021 13:34
Task	PP-108	Analyze ORCID authentication methods	Patrik	Matej Halinkovič	Done	Done	30/11/2021 14:21	15/12/2021 13:32
Task	PP-107	Port Webapp prototype to Django	Matej Halinkovič	Matej Halinkovič	Done	Done	30/11/2021 14:14	07/12/2021 14:55
Task	PP-106	Data pipe testing design	Jakub Knánik	Matej Halinkovič	Done	Done	30/11/2021 14:12	15/12/2021 13:32
Task	PP-104	Retrain model with aminoacids	Matej Halinkovič	Matej Halinkovič	Done	Done	30/11/2021 14:08	07/12/2021 14:54
Task	PP-103	Sprint retrospective 4	Matej Halinkovič	Matej Halinkovič	Done	Done	30/11/2021 14:07	15/12/2021 13:30
Task	PP-102	Meeting report 9	Patrik	Matej Halinkovič	Done	Done	30/11/2021 14:07	08/12/2021 12:16

J Prihláška na TP Cup

Kto sme?

Náš tím je tvorený ľuďmi, ktorí majú blízky vzťah k programovaniu, strojovému učeniu a dátovej analýze. Zo školských ale aj pracovných projektov sme získali nielen technické, ale aj manažérske skúsenosti. Každý člen tímu sa stretol s výskumnou prácou – vieme efektívne vyhľadávať zdroje a získavať z nich relevantné informácie, dokážeme výskumný projekt dotiahnuť do prezentovateľnej podoby a niektorí z nás už výskumnú prácu aj publikovali. Ak sa stretneme so zdanlivo neriešiteľným problémom, sme silne motivovaní ho dôkladne analyzovať a nájsť vhodné riešenie. Práve preto si myslíme, že sme tí praví pre riešenie tohto projektu.

Kontext projektu a motivácia

V posledných rokoch môžeme pozorovať zvýšenú prevalenciu kožných ochorení, Jednou z príčin je aj časté používanie rôznych liekov či kozmetiky a celková chemizácia života (potravinové aditíva, rastlinné zložky, umelé sladidlá, ...). Zvýšená citlivosť kože je priamo spôsobená interakciou žiarenia a chemickej látky v koži, ktorá je schopná pohlcovať žiarenie. Z toho dôvodu je nutné testovať všetky chemické substancie prichádzajúce na trh na fototoxicitu (FOTOtox). V minulosti sa FOTOtox testovala výlučne na zvieracích modeloch, čo je nehumánne a zároveň nepresné vzhľadom na rozdielnosť kože ľudí a zvierat. V súčasnosti sa FOTOtox testuje na bunkových kultúrach (OECD guideline 432) a na tkanivových kultúrach (OECD 498). Pre stanovenie dermatotoxicity je možné podľa guidelinov OECD využívať aj predikcie pomocou informačných technológii, ktoré skracujú čas, a šetria obrovské finančné prostriedky na "mokrý" experiment. Avšak pre stanovenie FOTOtox takáto možnosť chýba. Preto je hlavným cieľom predkladaného projektu navrhnúť riešenie pre in sillico experimenty na stanovenie FOTOtox. Myslíme si, že náš projekt by mohol pomôcť pri tvorení nových štandardov OECD. Výsledky projektu navyše budú pomáhať chrániť zdravie ako aj znižovať náklady pri vyhodnocovaní fototoxicity látok, preto túto tému považujeme za o to viac dôležitú, a zaujímavú.

Testovanie na zvieratách je žiaľ aj dnes súčasťou vývoja rôznych produktov - liekov, kozmetiky či farbív. Hoci spočiatku je možné testovať len

na odobratom tkanive, celkový efekt vyvíjaných produktov ukážu až testy na žijúcich zvieratách. Takéto testy ukážu, ako zareaguje koža po aplikovaní kozmetiky alebo akým spôsobom lieky ovplyvňujú orgány v tele. Ak testovaný produkt prejde bezpečnostnými testami a ukáže sa, že zabral na testovaných zvieratách, môže byť ďalej študovaný na ľuďoch. Tento proces je však už na dnešné pomery zastaralý a tak ako aj v iných odvetviach, tak aj v tejto oblasti je načase sa pohnúť vpred. Testovanie na zvieratách je totiž nielen nehumánne, ale aj neekologické a finančne nákladné. Každým rokom sa na testovanie použijú milióny zvierat rôznych druhov (myši, zajace, opice, psi, ...), pričom takéto testy sa často vykonávajú len preto, že to požadujú kontrolné úrady či legislatíva niektorých krajín.

Dnes už pritom existujú plnohodnotné alternatívy voči testovaniu na zvieratách. Napríklad s bunkovými kultúrami vieme vytvoriť miniatúry ľudských orgánov, na ktorých možno simulovať vplyv liekov. V prípade fototoxicity sa môžu použiť bunky z radu 3T3, ktoré sú vystavené testovanej chemikálii bez/s prítomnosťou svetla. Ďalšou alternatívou k testovaniu na zvieratách sú aj počítačové modely. Za posledné obdobie vývoj v tejto oblasti vzrástol enormným tempom, aj vďaka skúmaniu a využitiu umelej inteligencie. Jednou z možností využitia technológií je dolovanie v dátach, pomocou ktorého vieme vytvoriť predpovede o možnom nebezpečenstve chemických látok na základe už existujúcich dát. Veríme tak, že projekty ako je tento, pomôžu v procese prechodu od testovania na zvieratách k alternatívnym metódam.

Cieľ projektu

V tomto projekte sa zameriame na jednu oblasť nežiadúcich vplyvov liekov a látok, konkrétne na fototoxicitu. Fototoxická je látka, ktorá je toxická iba po pôsobení svetla. Cieľom projektu je využitie modelov kvantitatívnej závislosti aktivity na štruktúre (QSAR) pre získanie vzťahov medzi chemickou štruktúrou a biologickou aktivitou v súbore dát a následné predpovedanie FOTOtox aktivity nových chemických látok. Na základe zhromaždených dát a natrénovanému modelu tak bude vedieť naše riešenie predikovať fototoxicitu látky. Nakoniec celé riešenie nasadíme takým spôsobom, aby používateľ (toxikológ, chemik) mohol jednoducho podľa zápisu chemickej látky predikovať informácie o jej fototoxicite.

Návrh riešenia

Aby sme mohli model úspešne vytvoriť, potrebujeme najskôr zhromaždiť dostatočnú vzorku dát, z ktorej budeme môcť určiť, či niektorá konkrétna látka je alebo naopak nie je fototoxická. Analyzujeme tak veľké množstvo publikácií a relevantné informácie extrahujeme. Na identifikáciu chemickej látky vieme použiť CAS identifikačné kódy. Automatickým procesom dokážeme získať informácie o chemických látkach z verejne prístupnej databázy chemikálií, akou je napríklad PubChem.

Z týchto informácií nám vyplynie množstvo molekulových deskriptorov - potencionálnych prediktorov. Získané dáta následne analyzujeme technikou EDA, podľa ktorej budeme môcť identifikovať možné vzory v dátach a testovať hypotézy. Na základe hodnôt korelácií premenných môžeme pozorovať, ktoré vlastnosti látok spolu súvisia a akým spôsobom vplývajú na predikovanú premennú, teda na fototoxicitu. Medzi získanými dátami o chemických látkach sú aj ich SMILES kódy. Ide o chemický zápis, ktorý umožňuje reprezentáciu chemickej štruktúry spôsobom, aby ju mohol využiť počítač. Tieto kódy vieme takisto analyzovať a hľadať v nich vzory, ktoré by mohli vplývať na fototoxicitu, napr. počet aromatických cyklov. Ďalej sa zameriame na strojové učenie, preskúmame fungovanie rôznych algoritmov, ktoré by sa pre nás účel mohli hodiť (napr. v projekte pracujeme s chemickou štruktúrou, jednou z techník, ktoré by sme mohli aplikovať, je model grafovej neurónovej siete). Postupným procesom vytvoríme čo najúspešnejší a najpresnejší model, ktorý potom použijeme vo výslednom riešení. Finálna aplikácia môže mať podobu webovej aplikácie, do ktorej používateľ zadá SMILES kód chemickej látky a program vráti informácie o jej fototoxicite. Uvedené informácie môže následne používateľ ďalej využiť pre potreby svojej vlastnej práce. Týmto spôsobom tak pomáhame nielen v tvorbe alternatívy k testovaniu na zvieratách, ale aj v ďalšom výskume a pri tvorbe nových postupov OECD.

K Motivačný dokument

Tím

Náš tím je skupina vysoko motivovaných ľudí s blízkym vzťahom k programovaniu, strojovému učeniu a dátovej analytike. Pozostáva z bakalárov Mateja, Tibora, Jakuba K., Jakuba M., Patrika a Katky, ktorí sa navzájom dopĺňajú v znalostiach z rôznych oblastí informatiky. Po technickej stránke nájdeme v našej skupine znalosti zo všetkých potrebných oblastí. Skúsenosti z manažovania má každý člen tímu, nadobudnuté či už zo školských alebo aj pracovných projektov vo väčších tímoch. V rámci rôznych tímov, ktoré boli vedené agilným spôsobom, sme sa všetci stretli s verzovacími nástrojmi (Git), task manažérmi (Jira, Trello, Shortcut) alebo CI/CD nástrojmi (Jenkins).

Jakub M. má dva roky skúseností v agile tíme na pozícii Application Developer, kde na projekte v IBM pracoval na testovacích pipelinách (automatic functional, performance, API) pre webovú aplikáciu. Matej pracuje v oblasti Data Science / ML engineering pre spoločnosť Dell Technologies - pracovná náplň zahŕňa spracovanie big data, budovanie efektívnych modelov strojového učenia a ich uvedenie do produkcie. Katka má skúsenosť z výskumného tímu VeriFIT na FIT VUT, ktorý se zaoberá verifikáciou a testováním systémov.

Rovnako tak väčšina z nás ovláda základy databázových systémov. Na vyššej úrovni potom databázy ovláda Matej, ktorý má zároveň pracovné skúsenosti s analytickými OLAP databázovými technológiami Greenplum, Hadoop a Hive. S dátami je neodmysliteľne spojená aj manipulácia s nimi a ich analýza, s čím máme všetci v tíme aspoň základné skúsenosti. Väčšia časť tímu z bakalárskeho štúdia (predmet Inteligentná Analýza Údajov), a Katka z predmetu Zpracování a vizualizace dat v prostředí Python na VUT FIT.

Celý tím má skúsenosti s výskumnou prácou (prevažne z bakalárskych prác), kde viacero členov tímu pracovalo na témach z oblasti umelej inteligencie a hlbokého učenia. Traja členovia tímu boli súčasťou výskumnej orientácie, v rámci ktorej svoje výstupy verejne publikovali. Jakub K. sa venoval Analýze problematiky nepravdivých informácií. Matej sa so svojou bakalárskou prácou Analýza hudby pomocou umelých neurónových sietí zúčastnil konferencie IIT.SRC. Tibor má publikáciu v žurnáli Diagnostics vo spolupráci s doktorandom fakulty a zahraničnými lekármi v oblasti diagnos-

tiky pomocou hlbokého učenia.

Každý člen tímu má skúsenosti s technickým dokumentovaním kódu a so softvérovím inžinierstvom a architektúrou. Preferovaným jazykom tímu je Python, kde máme všetci aspoň rudimentárne schopnosti aj z testovania a dokumentácie. Dodatočne má Tibor skúsenosti s kontrolou kvality kódu, a významnú znalosť noriem PEP kvôli kontribúciam do Python codebase. Katka má taktiež rozsiahlejšie znalosti z oblasti testovania a dynamickej analýzy, okrem iného s unit-testami v Pythone a C++, ale aj s integračnými testami v Bashi. Všetci členovia tímu majú zapísané predmety na inžinierskom štúdiu z oblasti umelej inteligencie, akými sú Objavovanie znalostí (všetci), Neurónové siete (všetci), Digitálne spracovanie zvuku, obrazu a biosignálov (Matej, Jakub K., Jakub M.) a Vyhľadávanie informácií (Matej, Patrik, Tibor).

Motivácia

11. (Q)SAR analýza fototoxických látok

Ochrana životného prostredia je v posledné roky významnou a dôležitou témou. Čoraz viac firiem sa snažia urobiť svoje business procesy ekologickejšie a to ako z dôvodu ekonomického tak aj sociálneho. Tento projekt vidíme ako šancu pomôcť tejto iniciatíve a pracovať na téme, ktorú všetci považujeme za nesmierne dôležitú.

Rôzne farmaceutické, kozmetické a potravinárske firmy dodnes používajú testy na zvieratách alebo rôzne drahé experimenty na tkanivách biopsovaných zo zvierat aby overili fototoxicitu látok. Je to nielen ne-ekologické, nehumánne ale aj finančne náročné, a chceme preto revolucionizovať spôsob determinácie fototoxicity v látkach.

Nakoľko je projekt riešený v spolupráci s organizáciou OECD, jeho praktické použitie v nadväzujúcich projektoch v rozvinutých ale aj rozvojových krajinách môže mať z dlhodobého hľadiska citeľný dopad na ekológiu a zelenú budúcnosť. Náš tím je zručný v oblasti spracovania signálu, analýzy dát a strojového učenia, a má skúsenosti vo výskume v oblasti bioinformatiky. Dodatočne, v našom tíme máme členov so zapísaným predmetom Digitálne spracovanie zvuku, obrazu a biosignálov, ktorý je obsahom priamo relevantný k zadanej téme.

15. ION Mobility Spectrometry for Rapid HEMP potency testing

Vo využití CBD pre medicínske účely vidíme veľký potenciál a v rámci tímu s ním máme aj osobné skúsenosti. Radi by sme pomohli rozvoju tejto potencionálne život-meniacej oblasti a zároveň prehĺbili naše vedomosti nielen v rámci súvisiacich technológií, ale aj v problémoch, s ktorými sa musia pestovatelia vysporiadať.

V dnešnej dobe sa v poľnohospodárstve už bežne využívajú podobné prístroje meracie napríklad obsah cukru v hrozne, obsah lepku v obilí, alebo kvalita medu. Týmto projektom by sme teda radi rozšírili už existujúce riešenia, ktoré sú v dnešnej dobe nepostrádateľné.

Matej, Tibor, a Jakub majú skúsenosti so spracovaním obrazu a signálov z bakalárskych prác. Počas inžinierskeho štúdia absolvujeme všetky predmety spojené s danou témou, niektorí členovia tímu majú tiež skúsenosti aj s budovaním experimentov pre porovnávanie vhodnosti rôznych prístupov strojového učenia a ich použitia v praxi.

Spracovanie signálu má bohaté využitie, preto by sme ocenili príležitosť nadobudnúť počas Tímového projektu relevantné vedomosti, ktoré by nám v budúcnosti mohli otvoriť dvere k množstvu ďalších príležitostí.

4. Adverse Media Screening

V modernej dobe IOT zariadení a prepojenosti je takmer nemožné vyhnúť sa zanechaniu rôznych digitálnych stôp na webe. Minulosť jednotlivcov sa čoraz jednoduchšie preveruje a skrytie spojenia s nelegálnymi aktivitami je čoraz náročnejšie.

Takáto previerka je však pre spoločnosti veľmi nákladná, nakoľko z veľkej časti pozostáva z manuálneho prehľadávania online zdrojov a digitálnych záznamov za účelom vytvárania správ o histórii osoby alebo spoločnosti. Tieto správy sú však nevyhnutné pre identifikovanie potencionálnych ukazovateľov podvodov, prania peňazí a iných nepriaznivých aktivít.

Našim cieľom je využiť nadobudnuté skúsenosti v oblasti strojového učenia pre automatizáciu procesu prehľadávania zdrojov a preveriť zapojenie osôb a spoločností do nelegálnych aktivít. Nami vytvorené riešenie by mohlo výrazne znížiť množstvo času a peňažných prostriedkov potrebných na vykonávanie tejto dôležitej činnosti.

Člen tímu Jakub K. má skúsenosti s technológiami relevantnými pre túto tému, a to konkrétne so spracovaním prirodzeného jazyka, web scraping a web

crawling. Zvyšok tímu je skúsený v oblasti analýzy dát a strojového učenia. Všetci si chceme prehĺbiť naše vedomosti v oblasti spracovania prirodzeného jazyka, KYC noriem a webových technológií.

Poradie tém

1 - najpreferovanejšie

1. 11. (Q)SAR analýza fototoxických látok
2. 15. Ion Mobility Spectrometry for Rapid HEMP Potency Testing
3. 4. Adverse Media Screening
4. 3. DataHub pre rôzne typy zariadení, ich spracovanie / analýzu / vizualizáciu
5. 12. Spektrometrické rozpoznávanie túh do pera
6. 19. Automatizácia procesov KYC (Know your client) a AML (Anti-money laundering)
7. 14. IoT platforma na priemyselnú automatizáciu - malý pivovar
8. 5. Vytvorenie inteligentného model-based agenta (umelá inteligencia na báze Knowledge grafov) pre tvorbu komplexných dátových štruktúr a vzťahov pre aplikovaný výskum v klinickej onkológii
9. 8. Educational Content Engineering Hub - Databáza otázok, odpovedí, úloh a riešení [ECEH-DU]
10. 13. Navigácia v smartfóne pomocou rozšírenej reality
11. 9. Monitorovanie a správa systému pre výrobný areál [LOMON]
12. 7. Vizualiza´cia softve´ru vo virtua´lnej a rozs˘i´renej realite [VizReal]

L Generovaná technická dokumentácia

DataPipeline

Generated by Doxygen 1.8.17

Chapter 1

Usage:

Run the file run_pipeline.py after installing python using conda from environment.yml

run_pipeline.py supports the following arguments:

- **-inputfile "path"** to specify .csv containing [phototoxic, name] pairs, where phototoxic is either 0 or 1
- **-outputfile "path"** to specify final output for .csv file
- **-noscrape** to skip the scraping using crawlers

Chapter 2

Namespace Index

2.1 Packages

Here are the packages with brief descriptions (if available):

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Namespace Documentation

6.1 run_pipeline Namespace Reference

Functions

- def run_pipe ()

6.1.1 Detailed Description

```
Manages the execution of the pipeline.
```

6.1.2 Function Documentation

6.1.2.1 run_pipe()

```
def run_pipeline.run_pipe ( )
```

```
Runs the pipeline in sequence of steps: Crawler, merger, chem populator, cleaner.

The chem populator also merges existing data with new data before it's sent to the cleaner.

Multiple arguments are supported, as described in README.
```

6.2 scripts Namespace Reference

Namespaces

- cleaner
- crawlers
- merger
- populate_chem

6.3 scripts.cleaner Namespace Reference

Functions

- pd.DataFrame load_data (str csv_file_name)
- pd.DataFrame set_proper_data_type (pd.DataFrame df)
- [str, int] unique_count (pd.Series column)
- pd.DataFrame remove_useless_columns (pd.DataFrame df)
- list get_duplicit_correlated_descriptors (nx.MultiGraph graphs, exclude)
- def plot_correlation_heatmap (pd.DataFrame correlations, str fig_location, int threshold)
- def get_correlated_descriptors (pd.DataFrame df, int threshold, str fig_location)
- list create_correlation_graphs (pd.DataFrame correlations, str fig_location, float threshold)
- None save_mask (pd.DataFrame df, str mask_name, [None, list] columns=None, [None, list] rows=None, str data_dir="data")
- pd.DataFrame check_correlated_column (pd.DataFrame df, float threshold=0.9, bool remove=False, list preserve_columns=None, str plot_dir="plot", str data_dir="data")
- pd.DataFrame remove_duplicits (pd.DataFrame df, str subset, keep="first")
- def check_outliers (pd.DataFrame df, float threshold=4.2, bool remove=False)
- None main (str input_file="data/chem_output/chem_populated.csv", str output_file="data/final/phototox.↩ csv", float correlation_threshold=0.95, float outliers_threshold=4.2, list preserve_columns=None, bool remove=False)
- dict process_config (str config_file="conf/cleaner.ini")

Variables

- script_dir = os.path.dirname(os.path.realpath(__file__))
- project_dir = os.path.dirname(script_dir)
- dict args = process_config(os.path.join(project_dir, "conf/cleaner.ini"))
- input_data = os.path.join(project_dir, "data/chem_output/chem_populated.csv")
- output_data = os.path.join(project_dir, "data/chem_output/phototox.csv")

6.3.1 Detailed Description

```
Clean input data. Deletes unuseful columns. Correlations columns and outliers are ether deleted or a mask is c
```

6.3.2 Function Documentation

6.3.2.1 check_correlated_column()

```
pd.DataFrame scripts.cleaner.check_correlated_column (
            pd.DataFrame df,
            float  threshold = 0.9,
            bool  remove = False,
            list  preserve_columns = None,
            str  plot_dir = "plot",
            str  data_dir = "data" )
```

```
Check whether some columns are not correlated over given threshold.
If yes, remove them or create mask.
:param df: input DataFrame
:param threshold: correlation threshold
:param remove: if True,remove these columns, otherwise make mask
:param preserve_columns: columns, that are always preserved
:param plot_dir: dir to save generated correlation plot
:param data_dir:dir to store masks
:return: output DataFrame
```

6.3.2.2 check_outliers()

```
def scripts.cleaner.check_outliers (
            pd.DataFrame df,
            float  threshold = 4.2,
            bool  remove = False )
```

```
Check whether some outliers are over given threshold (standard deviation).
If yes, remove them, otherwise create mask.
:param df: input DtaFrame
:param threshold: standard deviation
:param remove: if True, remove these columns, otherwise make mask
:return: output DataFrame
```

6.3.2.3 create_correlation_graphs()

```
list scripts.cleaner.create_correlation_graphs (
            pd.DataFrame correlations,
            str fig_location,
            float threshold )
```

```
Plots correlation graphs.
:param correlations: correlation values.
:param fig_location: graph image file.
:param threshold: correlation threshold.
:return: list of subgraphs.
```

6.3.2.4 get_correlated_descriptors()

```
def scripts.cleaner.get_correlated_descriptors (
            pd.DataFrame df,
            int threshold,
            str fig_location )

This function creates correlation between descriptors
:param df: input DataFrame
:param threshold: threshold for minimal correlation
:param fig_location: location of correlation figure
:return: created correlations
```

6.3.2.5 get_duplicit_correlated_descriptors()

```
list scripts.cleaner.get_duplicit_correlated_descriptors (
            nx.MultiGraph graphs,
             exclude )

Fnc extract corelated  descriptors (column names) from graphs. From each correlation group is excluded one ite
or if some items from exclude are in correlation, these items are excluded instead.
:param graphs: input graphs
:param exclude: list of descriptors to exclude from final result
:return: list of correlated descriptors without excluded
```

6.3.2.6 load_data()

```
pd.DataFrame scripts.cleaner.load_data (
            str csv_file_name )

Load data from csv ta pandas DataFrame
:param csv_file_name: csv file name
:return: loaded DataFrame
```

6.3.2.7 main()

```
None scripts.cleaner.main (
            str  input_file = "data/chem_output/chem_populated.csv",
            str  output_file = "data/final/phototox.csv",
            float  correlation_threshold = 0.95,
            float  outliers_threshold = 4.2,
            list  preserve_columns = None,
            bool  remove = False )

Runs a series of cleaner functions and writes the result to the output file.
:param input_file: input file location
:param output_file: output file location
:param correlation_threshold: float
:param outliers_threshold: float
:param preserve_columns: list
:param remove: bool
:return: None
```

6.3.2.8 plot_correlation_heatmap()

```
def scripts.cleaner.plot_correlation_heatmap (
            pd.DataFrame correlations,
            str fig_location,
            int threshold )
```

```
Plot heatmap of correlated descriptors
:param correlations:input DataFrame with correlations
:param fig_location: path to store plot
:param threshold: used threshold to generate correlations
```

6.3.2.9 process_config()

```
dict scripts.cleaner.process_config (
            str  config_file = "conf/cleaner.ini" )
```

```
Process input config and extract
:param config_file: config file name location
:return: configuration
```

6.3.2.10 remove_duplicits()

```
pd.DataFrame scripts.cleaner.remove_duplicits (
            pd.DataFrame df,
            str subset,
             keep = "first" )
```

```
Removes duplicit rows
:param df: input DataFrame
:param subset: columns where to check duplicity
:param keep: keep argument
:return: output DataFrame
```

6.3.2.11 remove_useless_columns()

```
pd.DataFrame scripts.cleaner.remove_useless_columns (
            pd.DataFrame df )
```

```
Remove columns from DataFrame, that are duplicated from multiple sources or otherwise unuseful
:param df: input DataFrame
:return: reduced DataFrame
```

6.3.2.12 save_mask()

```
None scripts.cleaner.save_mask (
            pd.DataFrame df,
            str mask_name,
            [None, list]  columns = None,
            [None, list]  rows = None,
            str  data_dir = "data" )
```

```
Save mask
:param df: data.
:param mask_name: mask string.
:param columns: cols.
:param rows: rows.
:param data_dir: data directory.
:return: None
```

6.3.2.13 set_proper_data_type()

```
pd.DataFrame scripts.cleaner.set_proper_data_type (
            pd.DataFrame df )
```

```
Set data types to df columns.
:param df: input DataFrame
:return: df with changed types
```

6.3.2.14 unique_count()

```
[str, int] scripts.cleaner.unique_count (
            pd.Series column )
```

```
Get number of unique values in Series
:param column: input Series
:return: columns name and number of unique values
```

6.3.3 Variable Documentation

6.3.3.1 args

```
dict scripts.cleaner.args = process_config(os.path.join(project_dir, "conf/cleaner.ini"))
```

6.3.3.2 input_data

```
scripts.cleaner.input_data = os.path.join(project_dir, "data/chem_output/chem_populated.csv")
```

6.3.3.3 output_data

```
scripts.cleaner.output_data = os.path.join(project_dir, "data/chem_output/phototox.csv")
```

6.3.3.4 project_dir

```
scripts.cleaner.project_dir = os.path.dirname(script_dir)
```

6.3.3.5 script_dir

```
scripts.cleaner.script_dir = os.path.dirname(os.path.realpath(__file__))
```

6.4 scripts.crawlers Namespace Reference

Namespaces

- abstract_crawler
- crawler_orchestrator
- dependencies

6.5 scripts.crawlers.abstract_crawler Namespace Reference

Classes

- class AbstractCrawler

6.5.1 Detailed Description

```
Defines an abstract class for crawlers.
```

6.6 scripts.crawlers.crawler_orchestrator Namespace Reference

Functions

- def main (file)

6.6.1 Detailed Description

```
Manages the execution of web crawling.
```

6.6.2 Function Documentation

6.6.2.1 main()

```
def scripts.crawlers.crawler_orchestrator.main (
              file )
```

```
Accesses SwissADME and PubChem to retrieve smile codes and descriptors.
Uses two separate crawlers for each executed in squence, each producing a separate output.

:param file: The input file for the crawler, by default data/input_data/chem.csv
:return None
```

6.7 scripts.crawlers.dependencies Namespace Reference

Namespaces

- pubchem
- swissadme
- swisstarget

6.8 scripts.crawlers.dependencies.pubchem Namespace Reference

Classes

- class PubchemCrawler

6.8.1 Detailed Description

```
Crawler of the public chemical database PubChem.
```

6.9 scripts.crawlers.dependencies.swissadme Namespace Reference

Classes

- class SwissadmeCrawler

6.9.1 Detailed Description

```
Crawler of the Swiss Institute of Bioinformatics website for ADME parameters (SwissADME).
```

6.10 scripts.crawlers.dependencies.swisstarget Namespace Reference

Classes

- class SwissTargetCrawler

6.10.1 Detailed Description

```
Crawler of the Swiss Institute of Bioinformatics website for target predictions
(SwissTargetPrediction).
```

6.11 scripts.merger Namespace Reference

Functions

- None merge ()

6.11.1 Detailed Description

```
Manages the merging function of the data pipeline.
```

6.11.2 Function Documentation

6.11.2.1 merge()

```
None scripts.merger.merge ( )
```

```
Combines the output from PubChem and SwissADME outputs from the crawler, from their default locations.

Default input files:
file_pubchem = 'data/crawler_output/pubchem.csv'
file_swiss = 'data/crawler_output/swiss.csv'

Default output:
out = 'data/merger_output/merged.csv'
```

6.12 scripts.populate_chem Namespace Reference

Functions

- pd.DataFrame populate (file=None)

6.12.1 Detailed Description

```
Calculating features using rdkit.
```

6.12.2 Function Documentation

6.12.2.1 populate()

```
pd.DataFrame scripts.populate_chem.populate (
              file = None )
```

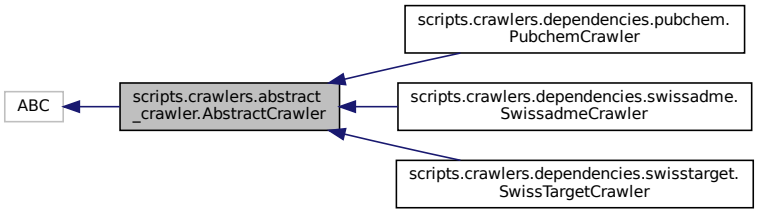
```
Takes smiles from the output .csv of merger, and populates the csv with chemical descriptors based on them.

The default input path is: "data/merger_output/merged.csv"
the default output path is: "data/chem_output/chem_populated.csv

The calculated descriptors currently are:
cycles
atom_valence
gesteiger sum and mean of positives, negatives, then total mean and total sum
hydrogen_count
csp3
atomatic_heavy_atoms
balban_j
hallkier
labute_asa
h_donors and h_acceptors
carbon_count
tpsa
druglikeness
vsa_logP
cycle_type_counts

:param file: Input file for the function for testing, otherwise leave empty for pipeline automation
:return The modified dataframe, but it's also edited inplace
```

Chapter 7

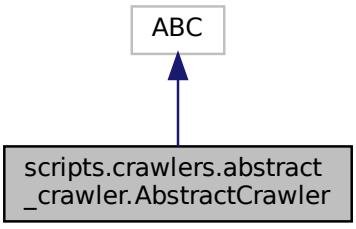
Class Documentation

7.1 scripts.crawlers.abstract_crawler.AbstractCrawler Class Reference

Inheritance diagram for scripts.crawlers.abstract_crawler.AbstractCrawler:



Collaboration diagram for scripts.crawlers.abstract_crawler.AbstractCrawler:



Public Member Functions

- bool is_file_downloaded (cls, str filename, int timeout=5)
- None crawl (cls, dict prefs)

7.1.1 Detailed Description

```
Abstract class that is used as an interface for crawler classes.
```

7.1.2 Member Function Documentation

7.1.2.1 crawl()

```
None scripts.crawlers.abstract_crawler.AbstractCrawler.crawl (
             cls,
            dict prefs )
```

```
Abstract method, has to be implemented. It should contain the core functionality
of the crawler.
:param prefs: preferences for the webdriver.
:return: None
```

Reimplemented in scripts.crawlers.dependencies.pubchem.PubchemCrawler, scripts.crawlers.dependencies.swisstarget.SwissTarget and scripts.crawlers.dependencies.swissadme.SwissadmeCrawler.

7.1.2.2 is_file_downloaded()

```
bool scripts.crawlers.abstract_crawler.AbstractCrawler.is_file_downloaded (
             cls,
            str filename,
            int  timeout = 5 )
```

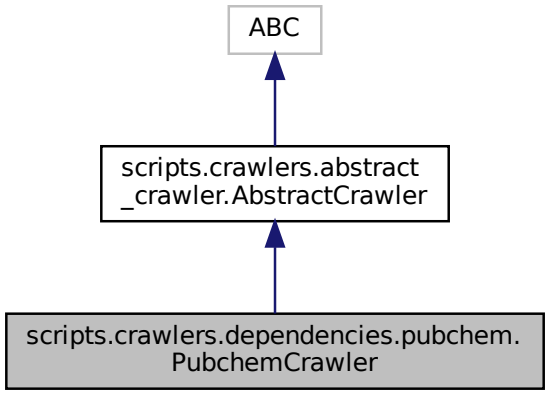
```
Checks whether the required file is present.
:param filename: path to the file.
:param timeout: time until file search stops.
:return: boolean stating whether the file was found or not.
```

The documentation for this class was generated from the following file:

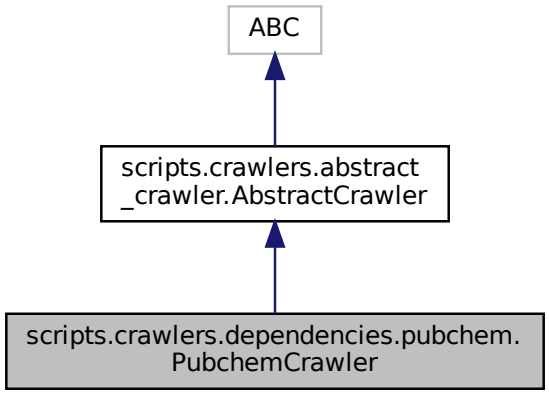
- scripts/crawlers/abstract_crawler.py

7.2 scripts.crawlers.dependencies.pubchem.PubchemCrawler Class Reference

Inheritance diagram for scripts.crawlers.dependencies.pubchem.PubchemCrawler:



Collaboration diagram for scripts.crawlers.dependencies.pubchem.PubchemCrawler:



Public Member Functions

- str get_classname (cls)
- None crawl (cls, dict prefs)
- bool is_file_downloaded (cls, str filename, int timeout=5)

Static Public Attributes

- list COLUMNS

7.2.1 Detailed Description

```
Class for the Pubchem crawler. The class uses Selenium for information extraction.
The crawler uses webdriver for Chrome. The class Follows the :class:`AbstractCrawler` interface.
```

7.2.2 Member Function Documentation

7.2.2.1 crawl()

```
None scripts.crawlers.dependencies.pubchem.PubchemCrawler.crawl (
              cls,
             dict prefs )
```

```
Extracts relevant information from the Pubchem database. Results are saved into a .csv file.
:param prefs: preferences for the Chrome webdriver.
:return: None
```

Reimplemented from scripts.crawlers.abstract_crawler.AbstractCrawler.

7.2.2.2 get_classname()

```
str scripts.crawlers.dependencies.pubchem.PubchemCrawler.get_classname (
              cls )
```

```
Get the name of the class.
:return: name of the class as a string.
```

7.2.2.3 is_file_downloaded()

```
bool scripts.crawlers.abstract_crawler.AbstractCrawler.is_file_downloaded (
              cls,
             str filename,
             int  timeout = 5 )  [inherited]
```

```
Checks whether the required file is present.
:param filename: path to the file.
:param timeout: time until file search stops.
:return: boolean stating whether the file was found or not.
```

7.2.3 Member Data Documentation

7.2.3.1 COLUMNS

```
list scripts.crawlers.dependencies.pubchem.PubchemCrawler.COLUMNS  [static]
```

Initial value:

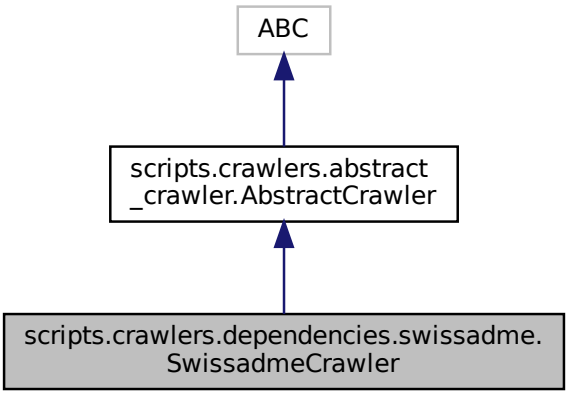
```
=  ["Phototoxic", "Name", "Smiles", "Molecular Weight", "Hydrogen Bond Donor Count",
               "Hydrogen Bond Acceptor Count", "Rotatable Bond Count", "Exact Mass",
               "Monoisotopic Mass", "Topological Polar Surface Area", "Heavy Atom Count",
               "Formal Charge", "Complexity", "Isotope Atom Count",
               "Defined Atom Stereocenter Count", "Undefined Atom Stereocenter Count",
               "Defined Bond Stereocenter Count", "Undefined Bond Stereocenter Count",
               "Covalently-Bonded Unit Count", "Compound Is Canonicalized"]
```

The documentation for this class was generated from the following file:

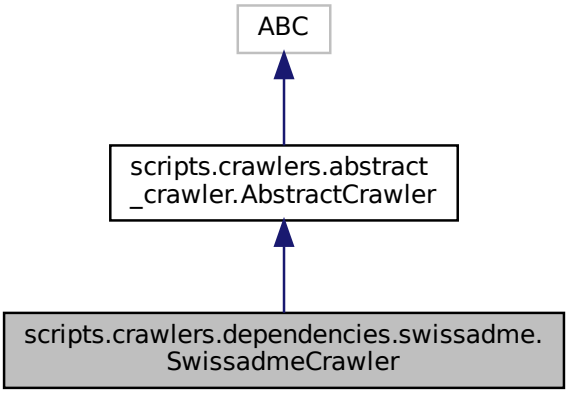
- scripts/crawlers/dependencies/pubchem.py

7.3 scripts.crawlers.dependencies.swissadme.SwissadmeCrawler Class Reference

Inheritance diagram for scripts.crawlers.dependencies.swissadme.SwissadmeCrawler:



Collaboration diagram for scripts.crawlers.dependencies.swissadme.SwissadmeCrawler:



Public Member Functions

- str get_classname (cls)
- None crawl (cls, dict prefs)
- bool is_file_downloaded (cls, str filename, int timeout=5)

7.3.1 Detailed Description

```
Class for the SwissADME crawler. The class uses Selenium for information extraction.
The crawler uses webdriver for Chrome. The class Follows the :class:`AbstractCrawler` interface.
```

7.3.2 Member Function Documentation

7.3.2.1 crawl()

```
None scripts.crawlers.dependencies.swissadme.SwissadmeCrawler.crawl (
             cls,
            dict prefs )
```

```
Extracts information from the SwissADME website. Results are saved into a .csv file.
:param prefs: preferences for the Chrome webdriver.
:return: None
```

Reimplemented from scripts.crawlers.abstract_crawler.AbstractCrawler.

7.3.2.2 get_classname()

```
str scripts.crawlers.dependencies.swissadme.SwissadmeCrawler.get_classname (
              cls )
```

Get the name of the class.
:return: name of the class as a string.

7.3.2.3 is_file_downloaded()

```
bool scripts.crawlers.abstract_crawler.AbstractCrawler.is_file_downloaded (
              cls,
             str filename,
             int  timeout = 5 )  [inherited]
```

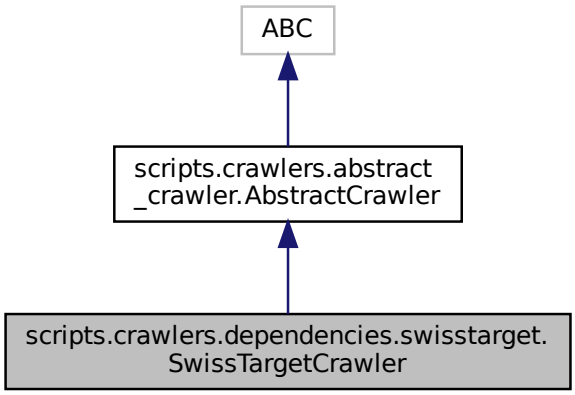
Checks whether the required file is present.
:param filename: path to the file.
:param timeout: time until file search stops.
:return: boolean stating whether the file was found or not.

The documentation for this class was generated from the following file:

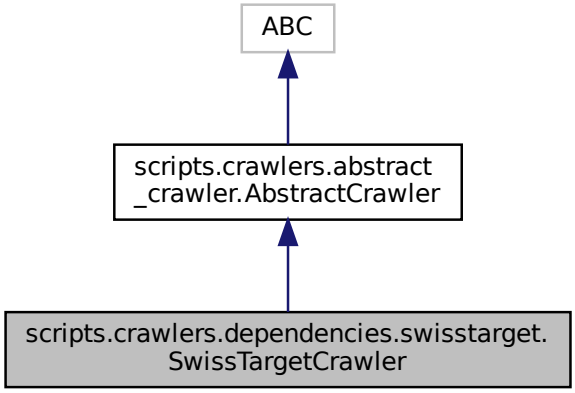
- scripts/crawlers/dependencies/swissadme.py

7.4 scripts.crawlers.dependencies.swisstarget.SwissTargetCrawler Class Reference

Inheritance diagram for scripts.crawlers.dependencies.swisstarget.SwissTargetCrawler:



Collaboration diagram for scripts.crawlers.dependencies.swisstarget.SwissTargetCrawler:



Public Member Functions

- str get_classname (cls)
- None crawl (cls, dict prefs)
- bool is_file_downloaded (cls, str filename, int timeout=5)

Static Public Attributes

- list COLUMNS

7.4.1 Detailed Description

```
Class for the SwissTarget crawler. The class uses Selenium for information extraction.
The crawler uses webdriver for Chrome. The class Follows the :class:`AbstractCrawler` interface.
```

7.4.2 Member Function Documentation

7.4.2.1 crawl()

```
None scripts.crawlers.dependencies.swisstarget.SwissTargetCrawler.crawl (
              cls,
             dict prefs )

Extracts information from the SwissTargetPrediction website.
Results are saved into a .csv file.
:param prefs: preferences for the Chrome webdriver.
:return: None
```

Reimplemented from scripts.crawlers.abstract_crawler.AbstractCrawler.

7.4.2.2 get_classname()

```
str scripts.crawlers.dependencies.swisstarget.SwissTargetCrawler.get_classname (
              cls )
```

Get the name of the class.
:return: name of the class as a string.

7.4.2.3 is_file_downloaded()

```
bool scripts.crawlers.abstract_crawler.AbstractCrawler.is_file_downloaded (
              cls,
             str filename,
             int  timeout = 5 )  [inherited]
```

Checks whether the required file is present.
:param filename: path to the file.
:param timeout: time until file search stops.
:return: boolean stating whether the file was found or not.

7.4.3 Member Data Documentation

7.4.3.1 COLUMNS

```
list scripts.crawlers.dependencies.swisstarget.SwissTargetCrawler.COLUMNS  [static]
```

Initial value:

```
=  ["Compound", "Target", "Common name", "Uniprot ID", "ChEMBL ID",
              "Target Class", "Probability*", "Known actives (3D/2D)"]
```

The documentation for this class was generated from the following file:

- scripts/crawlers/dependencies/swisstarget.py

Chapter 8

File Documentation

8.1 README.md File Reference

8.2 run_pipeline.py File Reference

Namespaces

- run_pipeline

Functions

- def run_pipeline.run_pipe ()

8.3 scripts/__init__.py File Reference

Namespaces

- scripts

8.4 scripts/crawlers/__init__.py File Reference

Namespaces

- scripts.crawlers

8.5 scripts/crawlers/dependencies/__init__.py File Reference

Namespaces

- scripts.crawlers.dependencies

8.6 scripts/cleaner.py File Reference

Namespaces

- scripts.cleaner

Functions

- pd.DataFrame scripts.cleaner.load_data (str csv_file_name)
- pd.DataFrame scripts.cleaner.set_proper_data_type (pd.DataFrame df)
- [str, int] scripts.cleaner.unique_count (pd.Series column)
- pd.DataFrame scripts.cleaner.remove_useless_columns (pd.DataFrame df)
- list scripts.cleaner.get_duplicit_correlated_descriptors (nx.MultiGraph graphs, exclude)
- def scripts.cleaner.plot_correlation_heatmap (pd.DataFrame correlations, str fig_location, int threshold)
- def scripts.cleaner.get_correlated_descriptors (pd.DataFrame df, int threshold, str fig_location)
- list scripts.cleaner.create_correlation_graphs (pd.DataFrame correlations, str fig_location, float threshold)
- None scripts.cleaner.save_mask (pd.DataFrame df, str mask_name, [None, list] columns=None, [None, list] rows=None, str data_dir="data")
- pd.DataFrame scripts.cleaner.check_correlated_column (pd.DataFrame df, float threshold=0.9, bool remove=False, list preserve_columns=None, str plot_dir="plot", str data_dir="data")
- pd.DataFrame scripts.cleaner.remove_duplicits (pd.DataFrame df, str subset, keep="first")
- def scripts.cleaner.check_outliers (pd.DataFrame df, float threshold=4.2, bool remove=False)
- None scripts.cleaner.main (str input_file="data/chem_output/chem_populated.csv", str output_file="data/final/phototox.↩ csv", float correlation_threshold=0.95, float outliers_threshold=4.2, list preserve_columns=None, bool remove=False)
- dict scripts.cleaner.process_config (str config_file="conf/cleaner.ini")

Variables

- scripts.cleaner.script_dir = os.path.dirname(os.path.realpath(__file__))
- scripts.cleaner.project_dir = os.path.dirname(script_dir)
- dict scripts.cleaner.args = process_config(os.path.join(project_dir, "conf/cleaner.ini"))
- scripts.cleaner.input_data = os.path.join(project_dir, "data/chem_output/chem_populated.csv")
- scripts.cleaner.output_data = os.path.join(project_dir, "data/chem_output/phototox.csv")

8.7 scripts/crawlers/abstract_crawler.py File Reference

Classes

- class scripts.crawlers.abstract_crawler.AbstractCrawler

Namespaces

- scripts.crawlers.abstract_crawler

8.8 scripts/crawlers/crawler_orchestrator.py File Reference

Namespaces

- scripts.crawlers.crawler_orchestrator

Functions

- def scripts.crawlers.crawler_orchestrator.main (file)

8.9 scripts/crawlers/dependencies/pubchem.py File Reference

Classes

- class scripts.crawlers.dependencies.pubchem.PubchemCrawler

Namespaces

- scripts.crawlers.dependencies.pubchem

8.10 scripts/crawlers/dependencies/swissadme.py File Reference

Classes

- class scripts.crawlers.dependencies.swissadme.SwissadmeCrawler

Namespaces

- scripts.crawlers.dependencies.swissadme

8.11 scripts/crawlers/dependencies/swisstarget.py File Reference

Classes

- class scripts.crawlers.dependencies.swisstarget.SwissTargetCrawler

Namespaces

- scripts.crawlers.dependencies.swisstarget

8.12 scripts/merger.py File Reference

Namespaces

- scripts.merger

Functions

- None scripts.merger.merge ()

8.13 scripts/populate_chem.py File Reference

Namespaces

- scripts.populate_chem

Functions

- pd.DataFrame scripts.populate_chem.populate (file=None)

Index