

Metadata reranking Twitter

David Omrai

12. prosince 2021

1 POPIS PROJEKTU

Tento projekt je nadstavbou nad sociální sítí Twitter, která svým uživatelům umožňuje vytváření a sdílení zpráv, taktéž známých jako "tweety". Tyto zprávy mohou obsahovat text, pro který byly prvně zamýšlené, ale nyní taktéž i multimédia jakož jsou obrázky, videa a ve speciálních případech i zvuk (iOS). Vzájemná interakce probíhá pomocí tzv. srdíček, kterými si uživatel zvolí, které tweety se mu líbí a na základě toho mu jsou doporučovány další. Dále lze příspěvky komentovat a sdílet tzv. retweetnout. Toto všechno umožňuje uživatelům si vytvářet prostředí, ve kterém se cítí dobře a objevují pouze ty věci, které se jim budou nejspíše dle "doručovacího algoritmu" líbit.

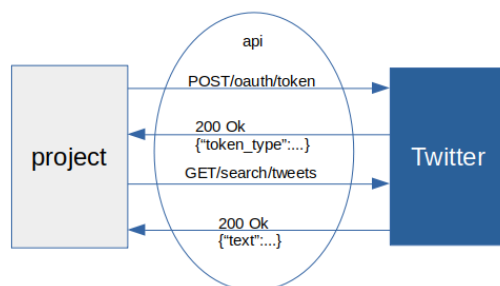
1.1 Náplň projektu

Problém však stále přetrvává ve vyhledávání tweetů pomocí vlastně definovaných preferencí, neboli není zde možnost tweety seřadit například pomocí toho kolik daný tweet má srdíček, kdy a kým byl vytvořen a co se v něm objevuje. Z důvodu absence této vlastnosti jsem se v tomto projektu rozhodl vytvořit jednoduchý Twitter metadata reranking, který umožní jednoduché přeuspořádání tweetů pomocí uživatelem zadaných parametrů. Uživatel si na vstupu vybere které tweety chce vyhledat a pomocí množiny možných vyhledávacích parametrů, společně s váhami, ovlivní uspořádání tweetů. Každý zadaný vyhledávací parametr bude využit v příslušné podobnostní funkci s daty obsaženými v jednotlivých tweetech. Množina všech takto získaných podobností je pak pomocí váženého průměru spočtena a ta reprezentuje výslednou podobnost jednotlivých tweetů uživatelskému požadavku.

1.2 Twitter API

Twitter API představuje programovatelného prostředníka mezi dvěma aplikacemi, které spolu chtějí komunikovat. Základními kameny této komunikace jsou HTTP síťové požadavky GET k získání dat a POST k nahrání dat. Díky tomuto je umožněno komukoliv, kdo si zařídil potřebné oprávnění komunikace s Twitter serverem. Toto oprávnění se dá získat pomocí vyplnění krátkého dotazníku, která zjišťuje, kdo je autorem a co od aplikace chce. V mém případě stačilo uvést, že se jedná o aplikaci pro studijní potřeby a v krátkém popisu uvést cíle této práce. Do několika dní od tohoto vyplnění přichází rozhodnutí. V případě kladné reakce je umožněno

autorovi si vygenerovat přístupové, uživatelské klíče. Pomocí těchto klíčů se prokazuje identita dané aplikace při komunikaci se serverem.



Obrázek 1.1: Využití API

Na obrázku je zobrazena prvotní komunikace aplikace s Twitter serverem, která nejprve sděluje svůj klíč a poté se dotazuje na získání určitých tweetů. Server Pokaždé požadavek zpracuje, prozkoumá, zdali obsahuje veškeré potřebné části a odpoví.

Tato komunikace, ač by byla v projektu možná, tak lepším přístupem je využití knihovny. Ta tyto veškeré přímé HTTP požadavky obaluje metodami, a všechny potřebné požadavky sama sestavuje. Touto knihovnou je Tweepy a dále v textu se o ní zmíním.

1.3 Komplikace s Twitter API

Při práci s Twitter API jsem narazil na několik drobných problémů. Ty spočívají v omezení časového horizontu a počtu stáhnutelných tweetů. Tento limit je dán typem účtu, které je možné si vytvořit. Během 15 minut je možný pouze omezený počet požadavků, kterým je předcházeno před vytěžováním serverů. V rámci jednoho měsíce je taktéž omezený počet stáhnutelných tweetů. Tento limit je pro neplacené služby omezen na 500,000 tweetů/měsíc.

I přes tyto prvotní problémy, které mě nutily vymýšlet možné způsoby jak tento limit obejít, pokud by bylo třeba, se nakonec ukázalo, že poskytnutý počet bohatě stačí pro účely tohoto projektu.

Klasický Twitter Developer účet spadá do kategorie Elevated. V této kategorii je možné mít jeden projekt, přístup k v2 i v1 API a omezené objekty představující tweety. Co v nich chybí jsou například komentáře, které jsou přístupné ve vyšších a placených kategoriích. Z tohoto důvodu jsem musel určité plány lehce upravit a nakonec přišel na základní sadu parametrů, kterými lze vyhledávat podobné tweety.

1.4 Vstupy projektu

Tento projekt lze rozdělit na dvě logické části, kde první je frontend vytvořen s využitím frameworku REACT, který se stará o interakci s uživateli. Nejprve jsou uživatelé dotázáni k uvedení pomocí čeho si přejí vyhledávat tweety a kolik jich má na každý dotaz být. Dotazem se zde myslí uživatelské jméno a hashtag, kterými se tweety vyhledávají.

Vstupy projektu se dělí na dvě části. V první se nastaví co chce uživatel hledat, tedy hashtagy a jména twitterových účtů. A ve druhé si zvolí parametry a jejich váhy, které se využijí k získání podobnosti jednotlivých tweetů se vstupním nastavením. Do této části se řadí čas (hodiny a minuty), datum (den, měsíc a rok), počet srdíček, počet retweetů, délka textu a obsah tweetu.

Doplňujícími vstupy jsou pak váhy, počet tweetů na vyhledání a podobnostní funkce. V rámci projektu jsou vytvořeny dvě. První je naivní přístup pomocí "words similarity", kde se pouze počítá počet výskytů jednotlivých slov. Druhá využívá přetrénovaný model, díky kterému získá vektor reprezentující danou větu a pomocí kosinové podobnosti se získá jejich podobnost. Toto je provedeno mezi každou větou z tweetu a uživatelského požadavku. Více o tomto dále v textu.

1.5 Výstupy projektu

Druhou částí tohoto projektu je backend, který má na starosti veškerou práci s daty. Po získání množiny uživatelem nadefinovaných vstupních parametrů provede řadu operací, kterými získá chtěné tweety a ty ohodnotí podobnostmi pro každý parametr. Po získání podobnosti se všemi parametry provede finální podobnost za pomoci vah jednotlivých parametrů. S využitím této hodnoty se provede přeuspořádání tweetů a výsledek je předán k vizuální prezentaci frontendu.

Frontend tyto syrová data zpracuje a zobrazí ve formátu nejvíce připomínající tweet z oficiální stránky. Uspořádání je on nejpodobnějšího od po nejméně podobný. Uživatel si tento seznam může prohlédnout stejným způsobem, který by využil na Twitteru.

2 ZPŮSOB ŘEŠENÍ

V předešlé části tohoto dokumentu jsem lehce nakoukl, jak je tato aplikace organizovaná. Že zde nalezneme frontend starající se o komunikaci s uživatelem a backend starající se o veškerou práci na datech. V této sekci tyto věci dále rozvedu a popíši, jak fungují, a co v jaké lze najít. Při zpracovávání tweetů k získání jejich podobnosti s uživatelem nadefinovaným vstupem jsou využity algoritmy probírané na přednáškách. Jelikož tweety už z jejich vlastního konceptu jsou balíčky dat, tak většina těchto funkcí bude zaměřena právě na ně.

2.1 Zpracování uživatelských dat

Zpracování dat zadaných uživatelem probíhá v několika krocích. V připraveném formuláři si uživatel nejdříve zvolí podle čeho chce tweety vyhledávat. Zde zadává jednotlivá klíčová slova a názvy profilů do patřičných políček. Limit není nastaven a pro každý požadavek aplikace stáhne definovaný počet tweetů. Tento počet si v této části uživatel také volí, předem je definováno desítka.

Druhá část uživatelského vstupu je formulářem připravena jakožto "custom search", kde lze navolit parametry tweetů a jejich důležitost. Váhy nejsou předem z žádného omezeného rozsahu. Výsledné váhy se dále v programu sečtou a váha jednoho parametru bude závislá ku všem. Pro vyhledávání lze definovat datum vytvoření tweetu a čas. Původně byla podobnost nastavena na číselné podobě dat. Tedy každá část data a času se porovnávala s tou skrytou v tweetu a počet položek, třeba hodiny, minuty atd., rozhodovaly o celkové podobě. To však nebylo příliš intuitivní a proto to bylo změněno, ale o tom dále v této sekci.

Jednou z nejdůležitějších částí vyhledávání dle mého názoru je obsah tweetu. Tedy uživatel si z toho množství tweetů zvolí dle zadaných témat ty, které chce upřednostnit. Do políčka zadá v případě "words similarity" slova, kterými chce tweety prioritizovat, nebo věty, které drží myšlenku v případě kosinové podobnosti. Opět o tomto se více rozepíši dále v textu. Když už je uživatel spokojený se svým parametrizovaným požadavkem k vyhledání stiskne "SUBMIT" tlačítko.

Nyní se tedy podíváme, co se děje v pozadí frontendu aplikace. Tato část je vytvořena ve frameworku React a tedy je napsána v javascriptu. Jádrem této strany aplikace se schovává v App.jsx, která spojuje jednotlivé části stránky dohromady. Dokud frontend nemá data, uspořádané tweety k zobrazení, tak se na stránce zobrazí pouze formulář. Formulář se zobrazuje po celou dobu operace s aplikací a to kvůli tomu, aby mohl uživatel opětovně zasílat nové požadavky k vyhledávání.

Stránka je složena ze čtveřice celků, komponentů. Těmi jsou vyhledávací formulář v SearchForm.jsx, výsledek vyhledávání v Results.jsx, jednotlivý tweet v Tweet.jsx a zápatí stránky s mým jménem v Footer.jsx. Dokud není výsledek vyhledávání není část s výsledkem zobrazena. Po odeslání požadavku k vyhledávání zavolá SearchForm onSubmit funkci, kterou mu dala App. S daty z formuláře poté App vyšle POST požadavek backendu a vyčkává na jeho zpracování. Délka zpracování se odvíjí od vybrané podobnosti metody a počtu zadaných parametrů. Po dokončení backend zašle kladnou zprávu a App se opět dotáže nyní již s GET požadavkem. Data se natáhnou všechna a dojde k jejich uložení do proměnné state. Uložení pomocí setState() funkci vyvolá aktualizaci stránky a po ní se získané tweety předají Results.

Po získání tweetů Results pomocí smyčky všechny tweety vybere a pošle je Tweet aby z nich vytvořila vizuální reprezentaci tweetů. Toto postupně vytvoří vizuální výsledek, který si může uživatel prohlédnout tak, jak by to udělal i na Twitteru.

2.2 Komunikace frontendu s backendem

Komunikace, jak jsem již zmiňoval, se odehrává v App. Pomocí před-připraveného backendu, u kterého jsem využil framework Flask nebylo obtížné zajistit propojení backendu s frontendem. K tomu stačilo pouze v package.json nastavit proxy, která odkazuje na adresu serveru, běžícím na localhostu.

Vlastní komunikace pak na straně javascriptu je realizovaná pomocí Fetch API. Tedy api které umožňuje manipulaci s HTTP pipelínou pro požadavky a odpovědi. Metoda fetch je využita pro asynchronní získání dat ze serveru a to za pomoci promísy. Pokud během komunikace nastane problém, tak je frontend o tomto informován, včetně dodatečných informací o kódu chyby (404, 500, atd.). Na tento promise, který je vytvořen pomocí fetch dále napojuji pomocí then() metody další fetch(), který po úspěšném zpracování prvního natáhne data ze serveru. Celý proces tedy je kontaktování serveru/backendu s parametrizovaným dotazem, vyčkání na pozitivní odpověď a nakonec požadavek serveru/backendu k zaslání dat.

2.3 Zpracování uživatelského vstupu

Jádrum backendu, se kterým probíhá komunikace se nachází v app.py a je zde lehká implementace Flasku pro dva možné HTTP požadavky, které slouží k namapování specifických URL na asociované funkce. Celý backend je napsán v pythonu a tedy i tyto funkce. Po POST požadavku se spustí funkce search_tweets(), která nejdříve definuje globální proměnnou data, do které se budou ukládat získané tweety z Twitteru.

Nejprve se načtou uživatelská data získaná z formuláře pomocí metody get_json(). Pro další práci s těmito daty jsou z nich do separátní proměnné vyextrahovány váhy. První částí vlastního získání tweetů je vytvoření instance třídy Tweets. Tato třída v sobě implementuje řadu metod pro práci s i na tweetech. Nejdříve se metodou fetch_tweets(hashtags, people, num) načtou tweety pomocí Twitter API, které je implementováno v stejné pojmenované třídě. Pro každý vyhledávací parametr je načten definovaný počet tweetů. Tweety jsou nejdříve ve formátu použité knihovny Tweepy a v kódu probíhá jejich překlad do pandas dataframu. Ten je volen proto,

aby co nejvíce usnadnil budoucí práci na nich. Nově získané tweety se uloží do vnitřní proměnné Tweets třídy. Po získání dat se nad každou feature pro kterou uživatel poskytl data provede podobnostní operace. Výsledky jsou opět uloženy do pandas dataframů. Ze spočtených podobností jednotlivých parametrů je ještě před finálním přeuspořádáním nutno získat jednu hodnotu, která bude využívat i uživatelem definovaných vah. Tedy prioritizace jedné podobnostní před jinými. Toto se provede pomocí metody `count_weighted_similarities(weights, data)`, která si bere dva vstupní parametry a počítá vážený průměr všech spočtených podobností. Ten se opět uloží do pandas dataframů pro finální uspořádání.

Uspořádání všech tweetů se provede pomocí metody `sort_tweets()`. Zde se provede lehké přeuspořádání dat pomocí získaného váženého průměru. V poslední části funkce `search_tweets()` se do globální proměnné `data` uloží výsledná data ve formátu json, o který se stará metoda `tweets_to_json()`. Po požadavku o navrácení dat se data ve formátu json odešlou frontendu k finálnímu vizuálnímu zpracování.

2.4 Komunikace backendu s Twitterem

Komunikace aplikace s Twitterem je nejdůležitější částí projektu a k jeho dosažení bylo potřeba zajistit několik věcí. Dříve v textu jsem zmiňoval, že bylo potřeba si založit Twitter Developer účet. Abych jej nezakládal na mém vlastním účtu, tak jsem si vytvořil účet nový a podal žádost. K žádosti samotné vedla cesta přes několik otázek co a jak plánuji s tímto účtem dělat a nakonec bylo třeba vyplnit krátkou slohovou práci o tom, co přesně je mým cílem. Jelikož jsem ve všem uvedl, že se jedná o školní práci netrvalo uznání developer účtu až tak dlouho. Každý účet si může zakládat jeden až několik projektů, v rámci kterých lze Twitter API využívat.

Ten svůj jsem pojmenoval podobně jako semestrální práci Twimetaking (Twitter meta reranking). A tímto projektem jsem byl umožněn si vygenerovat vlastní klíče, kterými probíhá komunikace s Twitter serverem. Tato aplikace má povoleno číst a psát tweety stejně tak i posílat zprávy, ale v této semestrální práci využívám pouze čtení tweetů.

Komunikace s Twitter servery by byla možná i pomocí GET a FETCH HTTP požadavků, ale značně by to znepřehlednilo kód. Z tohoto důvodu jsem se rozhodl využít knihovnu Tweepy, která všechny tyto operace obaluje. Její použití spočívá nejdříve ve vytvoření instance s využitím autentizace. Ta si bere pěti `consumer key`, `consumer secret`, `access token`, `access token secret` a `bearer token`. Tyto tokeny a klíče se starají o přihlášení k twitter účtu, přes který se následně provádějí veškeré akce.

Veškeré tyto části ne nachází v TwitterAPI. Při vytváření této třídy se v metodě `__init__()` vytvoří autentifikační objekt s využitím klíčů a tokenů. Ten se následně využije k získání tweepy API, přes které bude probíhat veškerá komunikace s Twitterem. Pro možné budoucí rozšíření jsem zde taktéž vytvořil klienta a uživatele, obojí objekty tweepy knihovny, které lze například využít při vytváření tweetů či posílání zpráv.

V této třídě se nachází trojice metod, z nichž dvě se starají o stahování tweetů z Twitteru a jedna, která je těmito dvěma využívána k formátování získaných dat. Toto formátování spočívá v získání z tweetu, množiny dat, jen toho co je třeba. Každý tweety objekt je přeplněn různými informacemi, jako je text tweetu, datum vytvoření, počet srdíček atd. Jelikož ne všechny tyto data se v této aplikaci využijí, tak jsem zavedl lehký způsob jak si je definovat. V `init` metodě se krom jiného nachází i proměnná `tweet_features`, do které se vloží v definovaném formátu, které části tweetu chceme získat. Tato informace se využívá při vytváření pandas dataframů a taktéž při extrakci samotné.

Dvě metody které slouží k získávání dat se jmenují `get_tweets_by_hashtags(hashtags, length)` a `get_tweets_by_usernames(usernames, length)`. Jak jistě jejich název sám popisuje, tak každá z nich se stará o stažení jiným způsobem tweetů. V případě `usernames` toto stáhne určitý počet tweetů z profilů daných uživatelů a v případě `hashtags` to nalezne ty nejnovější tweety, které dané slovo obsahují. Původní plán byl využít `hashtags`, ale zde se objevily problémy s jejich získáváním. Proto jsem přemýšlel zdali toto nebude velký problém, ale přece jen ne všichni lidi označují tweety `hashtags` a samotné `hashtags` patří do podmnožiny slov, tedy pokud se budou hledat podle slov, tak se naleznou i `hashtags`.

Poslední metodou této třídy je `get_extracted_features(tweet)`. Tato metoda vezme tweet, který reprezentuje balík dat v určitém formátu a z parametrů definovaných v `tweet_features` vybere jen tyto. Je zde použita lehká logika, která si hraje s formátem slova a případně je seká aby získala ty správné data.

2.5 Podobnostní funkce

Důležitou částí tohoto projektu jsou samotné podobnostní funkce implementované v `Metrics.py`, které zajišťují podobnostní ohodnocení jednotlivých tweetů s využitím uživatelského vstupu. Tuto část proberu rychleji s popisem všech podobnostních funkcí, které se v projektu nacházejí.

2.5.1 Number similarity

Nejzákladnější podobnostní, která se v projektu objevuje je takzvaně číselná podobnost a určuje, jak jsou si dvě daná čísla podobná. Tedy také možno chápat jejich vzdálenost zobrazená na jednotkový interval, kde 1 znamená stejné a 0 absolutně nepodobné. Nepodobnost nastává tehdy, pokud srovnáváme jakékoliv číslo s nulou, jelikož sama nula neдрží žádný počet. Jednička pro nulu nastane jen v případě srovnání se sebou samou.

$$num_sim(a, b) = \frac{\min(|a|, |b|)}{\max(|a|, |b|)} \quad (2.1)$$

2.5.2 Time similarity

K podobnosti času lze přistupovat více přístupy. Prvním, který jsem implementoval byl průměr součtu podobností jednotlivých částí. To však není moc intuitivní a tak jsem do-implementoval klasickou, neboli převedení na minuty a z těch pak podobnost.

$$time_sim_{old}(a, b) = \frac{num_sim(a.hour, b.hour) + num_sim(a.minutes, b.minutes)}{2} \quad (2.2)$$

$$time_sim_{new}(a, b) = num_sim(a.hour * 60 + a.minutes, b.hour * 60 + b.minutes) \quad (2.3)$$

2.5.3 Date similarity

Podobně jak tomu bylo u času, tak i u data jsem prvně počítal průměr podobností jednotlivých částí. Tedy dne, měsíce a roku. Výhodu v tomto formátu vidím leda tak, když se daný uživatel nepamatuje všechny části ale jen určité z nich. Problém je, že ty ostatní pak mohou a ovlivňují zbytek. V tomto případě jsem vše převedl na timestamp, tedy sekundy od počátku epochy.

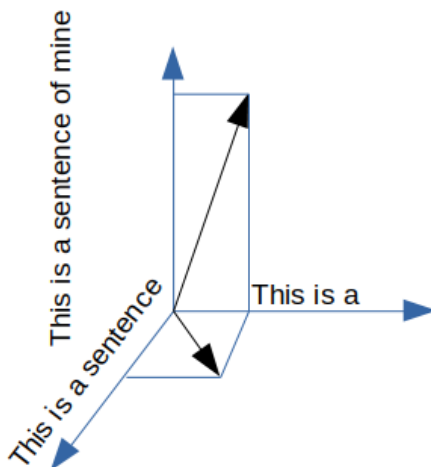
$$date_sim_{old}(a, b) = \frac{num_sim(a.d, b.d) + num_sim(a.m, b.m) + num_sim(a.y, b.y)}{3} \quad (2.4)$$

$$date_sim_{new}(a, b) = num_sim(epoch_sec(a), epoch_sec(b)) \quad (2.5)$$

2.5.4 Words similarity

Prvním a nejzákladnějším přístupem k měření podobnosti textů, které mě napadlo využít je pomocí vzdálenosti slov v nich obsažených. Z těchto dvou textů se vyextrahují vektory, které jsou založené na umístění jednotlivých slov ve větách. Pro přiblížení tohoto přístupu zmíním pár částí z článku [geeksforgeeks](#). [1]

Každý dokument, v tom to případě text je reprezentován posloupností slov, která si lze vhodně představit jako vektory slov. Pokud si vezmeme skalární součin těchto slov, a součin dvou stejných slov zastoupíme 1 a nestejných pomocí 0 dostaneme výsledek reprezentující počet slov, ve kterých se dané dva texty shodovaly.



Obrázek 2.1: Podobnosti vět

Výpočet skalárního součinu je první částí k vypočítání úhlu mezi dvěma větami. Úhel nám pak prozradí, jak moc si jsou dané věty podobné, pokud je úhel 0 stupňů jsou věty identické, pokud 90, tak velmi odlišné.

$$\cos(w) = \frac{a \cdot b}{|a||b|} \quad (2.6)$$

V Metrics je tento kód rozdělen do několika částí, kde každá část se stará o jednu část tohoto výpočtu. V první části každá věta rozseká do jednotlivých slov. Tyto slova jsou následně použity a z nich vypočítána frekvence jejich výskytu. Tím se množina všech slov jednotlivých vět značně zmenší a bude se s ní lépe pracovat. Následně jsou tyto dvě množiny slov použity k výpočtu jejich skalárního součinu. Nakonec je z nich vypočten úhel a ten dále vhodně namapován na jednotkový interval ke změření podobnosti.

$$numer(a, b) = dot_product(word_freq(a), word_freq(b)) \quad (2.7)$$

$$denom(a, b) = dot_prod(word_freq(a), word_freq(a)) * dot_prod(word_freq(b), word_freq(b)) \quad (2.8)$$

$$words_sim(a, b) = 1 - \frac{arccos(\frac{numer}{denom})}{\pi} \quad (2.9)$$

2.5.5 Cosine similarity with pretrained model

Poslední podobnostní funkcí je kombinace dvou sofistikovaných metod k hledání podobnosti po významové stránce jednotlivých vět. Touto kombinací je kosinová podobnost společně s předučným BERT modelem (transformátorem) k převodu vět na vektory reprezentující dané věty. Díky této reprezentaci vět lze následně využít nejnižší vzdálenost (Euclidean) nebo právě nejmenší úhel (Kosinova podobnost). Tento přístup je nejobtížnější z výpočetního hlediska, trvá poměrně dlouho jej spouštět na lokálu, ale za věnovaný čas získáme nejlepší match po významové stránce tweetů.

3 IMPLEMENTACE

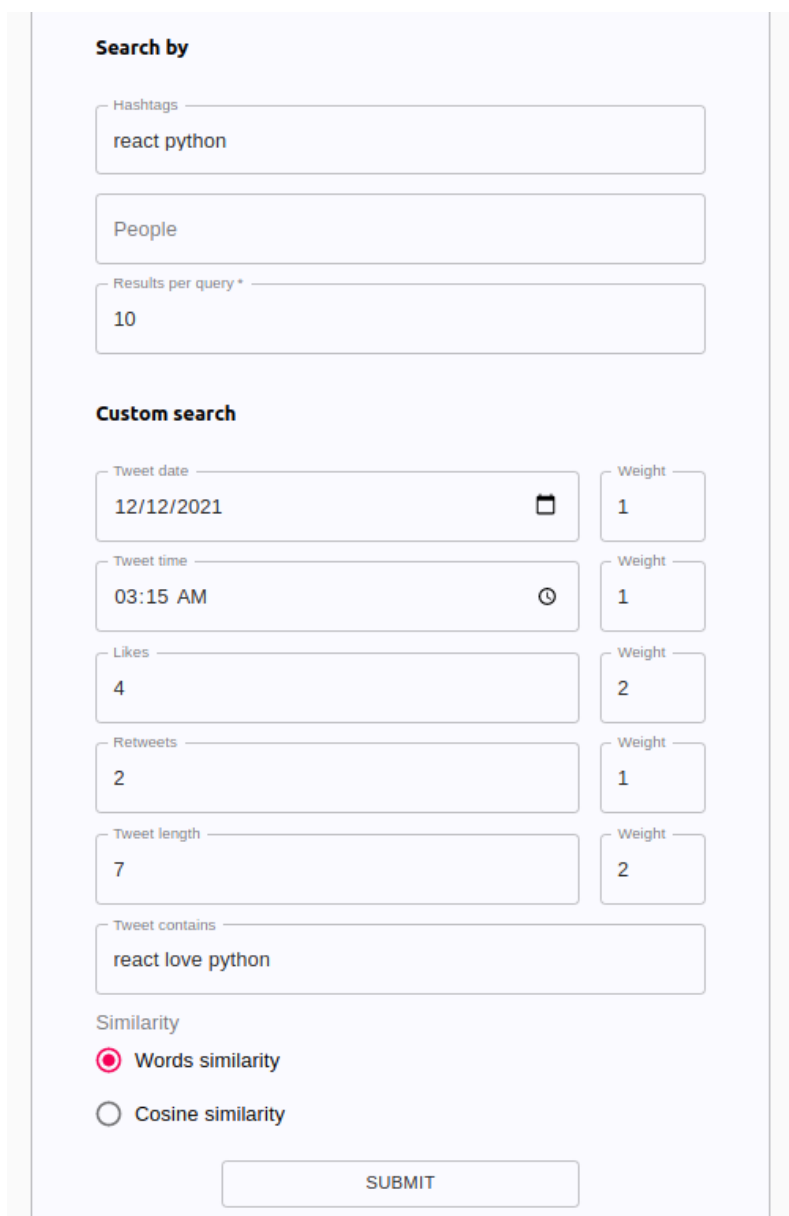
Použité prostředky a knihovny

- Frontend
 - Framework - React
 - Programovací jazyk - Javascript
 - Knihovny
 - * React - framework
 - * PropTypes - kontrola typů parametrů
 - * Material UI - komponenty pro design stránek
 - * clsx - definice stylů
- Backend
 - Framework - Flask
 - Programovací jazyk - Python
 - Knihovny
 - * Flask - framework
 - * tweepy - Twitter API
 - * numpy - matematické operace
 - * datetime - pro zpracování času
 - * re - regulární výrazy
 - * math - matematické operace
 - * pandas - práce s daty
 - * sentence-transformers - před-trénovaný model pro nlp

4 PŘÍKLADY VÝSTUPU

Pro prezentaci funkčnosti této semestrální práce jsem si vybral vyhledání tweetů obsahující slova react a python. Množinu těchto tweetů poté chci seřadit podle dnešního data, času během brzkého rána, počtu srdíček a sdílení a aby obsahovali určitá slova.

4.1 Vstup



The image shows a web form titled "Search by" and "Custom search". The "Search by" section has three input fields: "Hashtags" with the text "react python", "People" (empty), and "Results per query *" with the value "10". The "Custom search" section has several fields for filtering tweets: "Tweet date" (12/12/2021), "Tweet time" (03:15 AM), "Likes" (4), "Retweets" (2), and "Tweet length" (7). Each of these fields has a corresponding "Weight" field with values 1, 1, 2, 1, and 2 respectively. There is also a "Tweet contains" field with the text "react love python". At the bottom, there is a "Similarity" section with two radio buttons: "Words similarity" (selected) and "Cosine similarity". A "SUBMIT" button is at the very bottom.

Field	Value	Weight
Hashtags	react python	
People		
Results per query *	10	
Tweet date	12/12/2021	1
Tweet time	03:15 AM	1
Likes	4	2
Retweets	2	1
Tweet length	7	2
Tweet contains	react love python	

Similarity

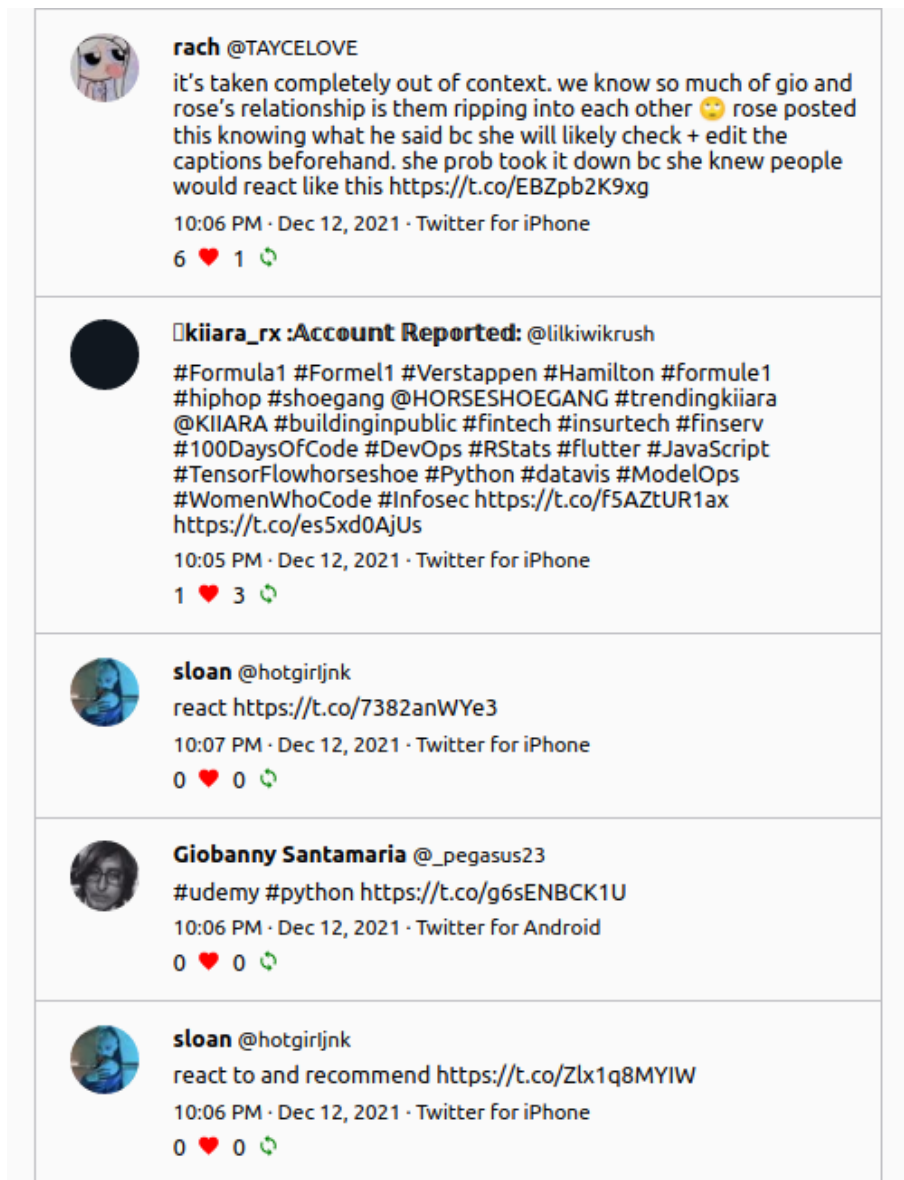
☒ Words similarity

☐ Cosine similarity

SUBMIT

Obrázek 4.1: Vstup programu

4.2 Výstup

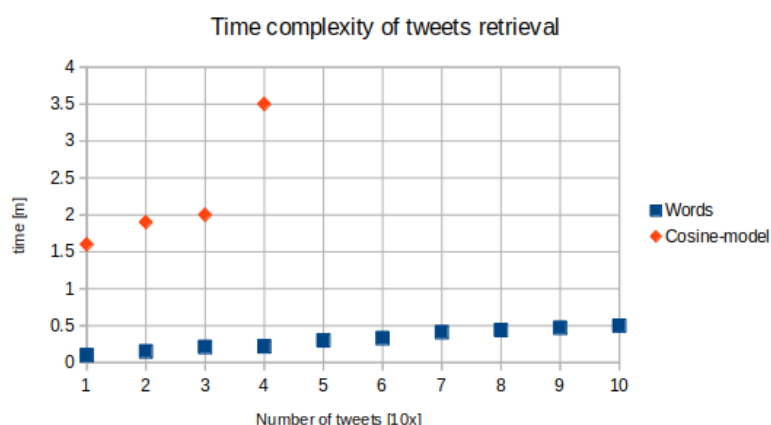


Obrázek 4.2: Výstup programu

5 EXPERIMENTÁLNÍ SEKCE

Jelikož není zrovna žádné měřítko, kterým by se dala určit přesnost výsledku, až na pořadí navrácených tweetů z hlediska čísel, která mohou být počet srdíček, sdílení nebo času a data. Tak se v této sekci zaměřím na to, jak počet tweetů s operací aplikace podobnosti textu liší se zvyšujícím se počtem tweetů. Tyto data zaznamenám do grafu a ten pak zhodnotím.

5.1 Měření



Obrázek 5.1: Časová náročnost běhu programu

Z časového důvodu jsem nebyl schopen zcela změřit průběh podobnostního ohodnocování modelem. Nicméně co se týká podobnosti podle výskytu slov lze pozorovat lineární růst časové náročnosti. Jedinými vstupy byl profil a text pro výpočet podobnosti.

6 DISKUZE

V rámci tohoto projektu jsem využil několik podobnostních funkcí, které šly využít na datech poskytnutými pomocí Twitter API. Nejzajímavějším bylo hledání podobnosti v textu samotném. Jelikož nejčastěji tuto stránku tweetů si bude chtít uživatel definovat, aby nelezl ty, které mu vyhovují. Lze na to přistoupit dvěma cestami, buď podobnostní slov, které jsem implementoval a funguje dobře, nebo významovou podobností, která je o něco výpočetně složitější. Jelikož u obou podobnostních přístupů využívám úhly k měření podobnosti, tak v možné rozšiřující práci bych se zaměřil i na ostatní přístupy. Například pomocí Jaccardovo indexů, k měření podobnosti slov ve dvou dokumentech (tweetech).

V rámci tweetu samotného se nachází více informací, které by šlo dále využít pro parametrizované vyhledávání. Avšak například lokalizace tweetu, či jiné nejsou často ukládány. Lidé je mohou mít vypnuty a z tohoto důvodu jsem je nevyužil v tomto projektu. V navazujícím by bylo zajímavé brát v potaz i je. V neposlední řadě bych se také více změřil na reranking. V tomto projektu jsem k němu přistupoval poměrně přímo. Nejdříve spočítat podobnosti jednotlivých parametrů zadáných uživatelem a poté jejich využití při váženém průměru. Tímto přístupem se ovšem ztrácí určitá informace, kterou uživatel chtěl zachovat zadáním vah na určité parametry.

7 ZÁVĚR

V tomto projektu jsem měl za úkol poskytnout uživatelům možnost rozšířeného vyhledávání tweetů. Toho se dosáhlo takovým způsobem, že v závislosti na uživatelském vstupu se spočte podobnost jednotlivých parametrů obsažených v tweetech. Uživatel tyto parametry může dále ohodnocovat váhami, a tak upřednostnit části svého vstupu před ostatními. Společně s váhami se spočetla hodnota odpovídající podobnosti jednotlivých tweetů se vstupem. Tato nová informace se následně využila k jejich preuspořádání tak, aby ty nejpodobnější uživateli požadavku předcházely ostatní méně podobné.

Značná část projektu je věnována příznivému uživatelskému prostředí, které má připomínat Twitter. V případě podobnosti vstupu na úrovni slov je aplikace svižná a vrací takové tweety, které by si uživatel přál. Stinnou stránkou projektu je využití modelu pro hledání podobného kontextu ve větách tweetů. Projekt je vyvíjen tak, aby bylo možné na něm dále pokračovat. Zhodnotil bych projekt jako úspěch, neboť mi přiblížil práci s Twitter API a co obnáší doporučování výsledku v závislosti na vstupu.

ODKAZY

1. AGAR, Walkeshav. *Measuring the Document Similarity in Python* [online]. 2020 [cit. 2021-12-12]. Dostupné z: <https://www.geeksforgeeks.org/measuring-the-document-similarity-in-python>.