Analýza sentimentu na sociálnych sieťach

Samuel Fanči

Jún 2022

1 Motivácia

Začiatkom roku 2021 sa na finančných trhoch odohrala neočakávaná udalosť. Firma GameStop [6], reťazec predajní v USA zameraných na videohry, bola už niekoľko rokov v recesii. Dôvodom bola hlavne digitalizácia predaja videohier a popularizácia konkurentných firiem, ktoré v tejto časti trhu začali dominovať.

Cena firmy GameStop klesala skoro lineárne už od roku 2014, pričom z ceny okolo $54 USD v 2014 padala postupne až trochu viacej ako $2.57 USD v apríli 2020. No v januári 2021 sa vďaka malým investorom z internetu vyšplhala najprv na $65.01, a o týždeň až na $483 (najvyššia vnútrodňová cena).

Je všeobecne známe, že komunita r/wallstreetbets [12] na sieti Reddit bola jedným z najdôležitejších vplyvov na nárast ceny, pretože práve užívatelia v tejto komunite si všimli podmienky na trhu, ktoré nakoniec vyústili v astronomický nárast cenz jej akcií. Pozdejšie sa celá udalosť rozšírila aj na iné siete, hlavne Twitter, a taktiež aj do významých TV staníc a novín v USA.

V mojom projekte sa zaujímam o spracovanie dát a sentimentu z r/wallstreetbest a Twitteru. Využívam rôzne knižnice a rozhrania na prácu s ich API a na trénovanie klasifikátorov, ku ktorým je dostupné aj grafické rozhranie. Cieľom je vytvoriť modely ktoré sa dajú použiť na analýzu sentimentu komunity. Toto sa môže ďalej použiť napr. na skúmanie vplyvu sentimentu na pohyb ceny GameStop(ďalej GME) v danom dni.

2 Zdroje dát

Pre Reddit aj Twitter existujú oficiálne API, ktoré dajú identifikovaným užívateľom prístup k rôznym informáciám zo siete, či už o príspevkoch alebo užívateľoch.

2.1 Reddit

Pre Reddit je ich vlastné API [13] obsiahle, ale nie je najlepšie na prehľadávanie histórie príspevkov. Preto som sa rozhodol použiť knižnicu PSAW [10], ktorá je súčasťou služby Pushshift.io, ktorá ukladá príspevky z celého Redditu a je možné v nich hľadať podľa kľúčových slov. PSAW je wrapper pre PRAW [9], čo je oficiálny Python API pre Reddit, takže sa s ním pracuje celkom jednoducho.

2.2 Twitter

Pre Twitter API [17] je to trochu komplikovanejšie. Taktiež existuje officiálna knižnica pre Python, Tweepy [16], ale pre moje potreby je nedostačujúca. Ak užívateľ nemá dostatočné vysoký level prístupu, tak nie je možné zistiť informácie o príspevkoch starších ako jeden mesiac.

Naštastie existuje niekoľkoverejne dostupných služieb, ktoré archivujú príspevky z Twitteru, a jednou z nich je snscrape. Táto služba je dostupná pre viaceré sociálne siete, ale pre mňa je podstatný modul sntwitter. Pomocou neho vieme hľadať v archivovaných príspevkoch podľa rôznych kľúčových slov, a aj pre špecifické časové rozmedzie. Taktiež vieme získať veľa informácií o jednotlivých príspevkoch.

3 Dostupné dáta

Obidve API boli nakoniec dosť jednoduché spracovať. V obidvoch som zadal do vyhľadávania kľúče "GME" a "Gamestop", a obidve služby mi vrátili príspevky, ktoré obsahujú aspoň jeden z nich, a to bez ohľadu na veľké a malé písmená. Dáta sú z časového rozmedzia od 07. 01. 2021 do 08. 02. 2022. Vrchol ceny GME nastal 27. 01. 2021, takže som chcel vybrať počiatočný dátum s dostatočným predstihom.

3.1 Reddit

Pre Reddit príspevky sú dostupné nasledovné informácie:

- id - unikátne ID príspevku.
- score - počet pozitívnych hlasov (ďalej 'upvotes') - počet negatívnych hlasov (ďalej 'downvotes'.
- selftext - text daného príspevku
- title - nadpis daného príspevku
- created_utc - čas a dátum vytvorenia príspevku, vo forme UTC timestamp
- upvote_ratio - pomer upvotes oproti downvotes.
- num_comments - počet komentárov daného príspevku.

Nakoniec štruktúra Reddit datasetu v .json formáte vypadá takto:

```
{
    '<ID príspevku>' : {
        'score' : int,
        'selftext' : '<text>',
        'title' : '<title>',
        'created_utc' : float,
        'upvote_ratio' : float,
        'num_comments' : int,
        'author' : {
            'name' : '<name>',
            'id' : '<id>'

        } // author môže byť None
    }
}
```

Figure 1: Štruktúra reddit.json

- **Title**: What superstitions/rituals do you guys do when holding?
- **Text**: What superstitions or rituals do you guys do when holding, except just hoping? Do you pray? I do.

  What do you do when you can't do anything else? I have lucky underwear and I like to refresh 3 times... Stuff like that. I'm wondering how far it can go? I like to imagine some of us rocking back and forth in our desk chairs, counting to 100 and clutching a sweat-stained GameStop t shirt.

  I mean, I don't know if I believe in any of this actually working, but at some point it's like - is it worth the risk not, once it worked more than once?

Figure 2: Príklad typického príspevku z Redditu

3.2 Twitter

Pre Twitter príspevky sú dostupné tieto informácie:

- Date - čas a dátum vytvorenia príspevku, vo forme pandas[8] DateTime objektu.
- UserID - ID užívateľa, ktorý tento príspevok vytvoril
- Content - text príspevku
- ID - unikátne ID príspevku.
- LikeCount - počet 'Likes' daného príspevku
- ReplyCount - počet odpovedí daného príspevku
- QuotedTweet - Tweet objekt citovaného príspevku, ak nejaký existuje. Uložíme si jeho ID
- RetweetedTweet - Tweet objekt 'retweeted' príspevku, ak existuje. Tiež si uložíme jeho ID.
- **Text:** Investing in 2021 rule number 1:

  You don't bet against 3.2m degens #crypto #wallstreetbets #investing $GME #gamestop

Figure 3: Príklad typického Tweetu

4 Spracovanie dát

Táto časť prebieha hlavne pomocou skriptov scraper.py a dataset_processor.py a annotator.py. Všetky výsledné .json súbory sú uložené v priečinku Data.

4.1 scraper.py

V scraper.py je hlavná úloha získať dáta priamo z API a uložiť ich v .json formáte. V obidvoch prípadoch prichádzajú príspevky ako objekty Submission a Tweet. Tieto objekty obsahujú nejaké informácie hneď, ale veľa z nich je načítaných až po zavolaní nejakého atribútu v kóde. Tým sa spraví HTTP request na dané API a odpoveď sa vráti až po chvíli.

Čiže trvá nejaký čas, kým sa spracujú všetky príspevky, pretože je potrebné čakať na odpovede. Taktiež je pri niektorých atribútoch potrebné zabaliť celú časť kódu do try/except bloku, pretože ak sa nejaký request nepodarí, vyhodí sa výnimka.

Po načítaní príspevkov v scraper.py ich uložíme do .json formátu. Tu je spravená chyba dizajnu programu, pretože som začal spracovať Reddit skôr ako Twitter, a štruktúra .json súboru pre Reddit je iná ako pre Twitter. Toto je ošetrené v
dataset_processor.py. Reddit príspevky sú najprv uložené do súboru reddit.json, v ktorom sú príspevky uložené v slovníku indexovaného pomocou ID príspevkov (Figure 1), a sú tam všetky dátové položky spomínané predošle. Twitter príspevky sú tiež serializované v podobe .json, a to v súbore twitter.json. No v tomto prípade som sa rozhodol, že lepší nápad je ukladať ich do pandas[8] DataFrame [7], pretože je jednoduchšie pracovať s DataFrames pri trénovaní modelov. Tento DataFrame je indexovaný len číslami, a jeho stĺpce obsahujú predošle spomínané informácie o danom príspevku.

4.2 annotator.py

Na trénovanie modelov potrebujeme mať nejaký počiatočný dataset, a na to máme annotator.py. Tento skript nám do terminálu predhadzuje príspevok za príspevkom a vieme pomocou neho uložiť informáciu o tom, či sentiment daného príspevku je pozitívny (anotácia 1), negatívny (anotácia 0) alebo irelevantný voči GME (uložené do časti 'irrelevant' v slovníku). Tento skript vytvorí slovník, ktorý je indexovaný pomocou ID príspevku, a hodnota je len daná anotácia. Čiže štruktúra je:

```
{
    '<ID príspevku>' : <anotácia>,
    '<ID príspevku>' : <anotácia>,
    '<ID príspevku>' : <anotácia>,
    ...
}
```

Figure 4: Štruktúra anotácií

alebo

```
{
    'irrelevant':
    {
      '<ID príspevku>',
      '<ID príspevku>',
      ...
    },

    '<ID príspevku>' : <anotácia>,
    '<ID príspevku>' : <anotácia>,
    '<ID príspevku>' : <anotácia>,
    ...
}
```

Figure 5: Štruktúra anotácií s irelevantnými príspevkami

Z historických dôvodov sa anotujú príspevky z Redditu a Twitteru samostatne, a potom sa to dokopy spojí v dataset_processor.py

Výstupom tohoto skriptu sú súbory:

- reddit_annotated.json - pozitívne a negatívne anotácie pre Reddit
- reddit_annotated_with_irrelevant.json - obsahuje aj irelevenatné príspevky
- twitter_annotated.json - rovnako ako pre Reddit
- twitter_annotated_with_irrelevant.json - rovnako ako pre Reddit

4.3 dataset_processor.py

Tento skript má za úlohu zjednotiť dáta z jednotlivých .json súborov do jednotného trénovacieho datasetu, ktorý obsahuje anotované príspevky, a taktiež totálneho datasetu, kde sú uložené relevantné informácie z obidvoch zdrojov. Pre trénovacie potreby nateraz stačí uložiť text daného príspevku, jeho typ (či pochádza z Redditu alebo Twitteru) a jeho ID.

Výstupom tohoto skriptu sú tri súbory: total_dataset.json - obsahuje všetky príspevky, train_dataset.json - obsahuje len pozitívne alebo negatívne anotované príspevky, a train_dataset_alt.json, ktorý obsahuje aj irelevantné príspevky. Irelevantné príspevky sú uložené s anotáciami 2.
Tu je príklad pre train_dataset_alt.json

	text	type	id	target
0	I've been working on a project to recreate DFV...	reddit_post	qicrti	2
1	I am an engineer that works in the tech indust...	reddit_post	qia0tt	2
2	Long time lurker (full disclosure, mostly for ...	reddit_post	qhvtqi	2
3	This DD is short, because the logic is straigh...	reddit_post	qhnq2y	2
4	In my opinion, now is the time to hedge long e...	reddit_post	qh7yld	2

Figure 6: Štruktúra "alternatívneho" datasetu. Hodnota target rovná 2 znamená, že príspevok je irelevantný. Počas tréningu sú tieto numerické hodnoty transformované na one-hot encoding (čiže ak máme 3 triedy, tak one-hot encoding pre číslo 2 je [0, 0, 1], pre 1 je to [0, 1, 0] atď.).

5 Modely

Cieľom bolo natrénovať klasifikátory, ktoré sa naučia rozdeliť príspevky na triedy podľa vstupného datasetu. Najprv som trénoval len pre pozitívne a negatívne, ale je lepšie využiť všetky anotácie, a taktiež vďaka tomu môžu byť modely robustnejšie, čiže trénujeme aj rozpoznávanie irelevantných príspevkov.

O trénovanie modelov sa starajú skripty neural_network.py, rnn.py a other_models.py, ktoré spolu tvoria modul Models.

5.1 other_models.py

Tento skript má nastarosti všetky modely ktoré niesú súčasťou TensorFlow [15], čo je knižnica zameraná na neurónové siete. Silne tu využívame knižnicu scikit-learn [14], ktorá obsahuje veľa implementovaných modelov na strojové učenie, a taktiež ďalšie nástroje na prácu s dátami a iné.

Najprv v main funkcii načítame dataset a súbor s výsledkami trénovania results.json, potom vyberieme model ktorý dáme funkcii trénovanie, a tá sa postará o predspracovanie dát, rozdelenie na trénovaciu a testovaciu množinu, krížovú validáciu, a vráti nám natrénovaný model spolu s inštanciou transformátoru ktorý bol použitý na predspracovanie.

Podstatná je funkcia train_model(), ktorá dostane ako jeden z argumentov inštanciu modelu, ktorý má natrénovať, a taktiež argumenty pre funkciu preprocess(), ktorá sa stará o predspracovanie textu. train_model potom tento predspracovaný text rozdelí na časti pre k-násobnú krížovú validáciu, zakaždým natrénuje model a zapamätá si rôzne metriky o ňom, a nakoniec vypíše priemernú presnosť, F_1 makro skóre, $F_{0.5}$ makro skóre a váženené F_1 skóre. Tieto informácie taktiež uloží do results DataFrame a vráti nám model s transformátorom.

F_β makro skóre sa vypočíta pre nejaké reálne číslo beta ako

$$F_\beta = (1+\beta)^2 \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \text{precision}) + \text{recall}}$$

kde

$$\text{precision} = \frac{\text{true positive}}{\text{true positive} + \text{false positive}}$$

$$\text{recall} = \frac{\text{true positive}}{\text{true positive} + \text{false negative}}$$

Keďže máme viac ako 2 triedy, potrebujeme urobiť nejaké priemerovanie. V prípade makro skóre zoberieme jednotlivé F_β skóre pre každú triedu a spriemerujeme ich.Pre vážené skóre zoberieme do úvahy proporciu každej triedy v našich dátach a podľa toho spravíme vážený priemer týchto skóre.

Okrem toho máme funkciu preprocess(), ktorá premení vstupný text na tfidf vektor, pričom je možné nastaviť niektoré parametre ako minimálnu povolenú dokumentovú frekvenciu pre nejaký term, alebo aké n-gramy máme brať do úvahy. N-gram je nejaká súvislá postupnosť n znakov v texte, napr. pre text 'neurónová sieť' máme napr. 4-gramy ako 'neur', 'euró', 'urón', ale aj 'vá s'.

Súčasťou tohoto je aj zmena všetkých znakov na malé písmená. Taktiež pred transformáciou na tf-idf[3] odstráni všetky znaky pre nový riadok, zátvorky a taktiež nahradí každý HTTP odkaz za reťazec "[link]", čo pomáha pri tréningu. Tf-idf je štatistika, ktorá sa snaží numericky reprezentovať ako dôležitý je daný term pre nejaký dokument v korpuse dokumentov. Matematicky to súvisí s podmienenou entropiou náhodne vybraného dokumentu, ak obsahuje term t. Počíta sa nasledovne

$$\text{tfidf}(t, d, D) = \text{tf}(t, d) \cdot \text{idf}(t, D)$$

kde

- D - korpus dokumentov
- d - dokument $\in D$
- t - term vyskytujúci sa v aspoň jednot dokumente v D

Skladá sa z dvoch častí: term-frequency(tf) a inverse document-frequency(idf).

$$\text{tf}(t, d) = \frac{\#\text{ výskytov slova t v d}}{\#\text{ slov v d}}$$

$$\text{idf}(t, D) = \log \frac{\#\text{ dokumentov v D}}{\#\text{ dokumentov v D obsahujúcich t} + 1}$$

Pre každý dokument vytvoríme vektor s týmito hodnotami, pričom na i-tej pozícii vo vektore pre dokument j je $\text{tfidf}(t_i, d_j, D)$. Nakoniec tieto vektory znormalizujeme euklidovskou normou.

Nakoniec máme funkcie s názvami modelov, ktoré nám vygenerujú objekt daného modelu a nastavia v ňom hyperparametre, ktoré chceme, a funkciu training_wrapper(), ktorá slúží ako unifikované rozhranie pre GUI, o ktorom bude reč pozdejšie.

Po tréningu sa model uloží v súbore <name>.MODEL, kde <name> je meno funkcie ktorá vygenerovala daný model.

5.2 neural_network.py

Tento skript sa stará o vytvorenie a natrénovanie doprednej neurónovej siete [1] pomocou TensorFlow. Funguje to veľmi podobne ako pri predošlom skripte, len v tomto prípade máme kvôli GUI najprv funkciu neural_network_ensemble. Táto slúži ako modelová funkcia podobne ako v module other_models, len pre

jednoduchosť sa v nej načíta aj dataset aj results. Tieto sa predajú funkcii train_model spolu s ostatnými parametrami.

train_model je tiež podobný ako v other_models, len teraz tu máme manuálne spravený ensembling. Pri každej iterácii krížovej validácie natrénujeme niekoľko modelov, pričom využívame subsampling aby každý model mal trochu inú trénovaciu množinu. Potom použijeme hlasovanie pri predikcií anotácií v testovacích dátach. Hlasovanie znamená že každému modelu predložíme vstup, necháme ho predikovať anotáciu a zoberieme tú anotáciu, ktorá má najviac hlasov, alebo pri rovnakom počte zoberieme prvú takú.

Taktiež súčasťou paramterov pre neural_network_ensemble a train_model je tuple layers_shape, ktorá drží informáciu o architektúre sietí, ktorá sa predá funkcii make_model. Táto funkcia vytvorí sieť pomocou vrstiev Dense z tensorflow.keras, čo je len husto prepojená vrstva neurónov s danou aktiváciu. Napr. ak layers_shape = (100, 200, 200), tak na prvej vrstve je 100 neurónov, na nich je napojených ďalších 200 atď. Každá sieť má potom na výstupe vektor veľkosti počtu tried, ktorý zodpovedá pravdepodobnostnej distribúcií tried pre daný vstup.

Funkcia make_model sieť vytvorí, zkompiluje a natrénuje ju na dátach, pomocou metódy Adam[5]. Je to optimizácia klasického stochastického gradientného zostupu používaného na trénovanie neurónových sietí, ktorá dynamicky počíta learning rate pre každý parameter v sieti počas tréningu, na základe prvého a druhého momentu gradientov.

Funkcia vráti objekt siete a jej presnosť. Po konci trénovania sa sieť serializuje a uloží ako neural_network.MODEL.

5.3 rnn.py

Tento skript má nastarosti trénovanie rekurentných sietí pomocou TensorFlow. Môžeme si vybrať, či chceme aby sme trénovali LSTM(Long Short-Term Memory) [2] alebo GRU (Gated Recurrent Unit) [4] jednotky. V tomto prípade nepotrebujeme spraviť také veľké predspracovanie, pretože súčasťou trénovania je aj natrénovanie Word Embeddings, čiže nepoužívame tfidf. Word Embedding je vektorová reprezentácia slova, kde sa snažíme aby slová s podobným významom mali malú vzdialenosť. Napriek to mu, že nepoužívame tfidf rovnako prečistíme text ako v ostatných prípadoch, pretože to pomáha s presnosťou.

Pred trénovaním ale musíme predspracovaný dataset prekonvertovať na Ragged Tensory, čo znamená že vytvoríme slovník so všetkými slovami, každému priradíme nejaký index, a vstupné reťazce premeníme na tenzory obsahujúce len zodpovedajúce čísla namiesto slov. Výhoda rekurentných sietí je, že dokážu pra-

covať so vstupom rôznej dĺžky, takže nemusím mať príliš dlhé vstupné vektory narozdiel od tfidf. Keď už máme text zkonvertovaný, tak ho ešte musíme rozdeliť na batche skladajúce sa z niekoľkých tenzorov, pretože RNN v Tensorflow sa trénujú len na takýchto vstupoch.

Toto všetko má nastarosti funkcia create_datasets(), ktorej okrem argumentov z terminálu predáme veľkosť pre jeden batch. Vráti nám tuple pozostávajúci z trénovacieho datasetu, testovacieho datasetu a validačného datasetu.

Samotné vytvorenie modelu zabezpečuje rnn_word_embed_simple(), kde vieme podobne zadať architektúru sieťe, typ rekurentnej bunky a learning_rate. Architektúra siete sa predáva pomocou tuplu layers_shape. Prvé číslo hovorí, akú veľkosť má mať výsledný Word Embedding vektor, ktorý vytvára vrstva tensorflow.keras.layers.Embedding. Ďalej nasleduje niekoľko obojsmerných rekurentných vrstiev, kde číslo na danej pozícii reprezentuje veľkosť vektoru na výstupe jednej tejto vrstvy. Úplne na konci máme podobne ako predtým vrstvu Dense so softmax aktiváciou, ktorá na vráti pravdepodobnostné rozloženie na triedach.

Čiže pre tuple (64, 32, 32) by sme mali embedding vektory veľkosti 64, a potom dve obojsmerné vrstvy s výstupnými vektormi veľkosti 32.

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

$$i = 1..K, z = (z_1, ..., z_K) \in \mathbb{R}^k$$

Figure 7: Vypočet funkcie softmax pre K-zložkový vektor

Obidva tieto moduly obsahujú funkciu training_wrapper(), ktorá bude zavolaná z gui a proste začne trénovanie, podobne ako keby sme zavolali skript sám o sebe.

6 Užívateľské rozhranie

Jednotlivé skripty na trénovanie modelov sa dajú spustiť aj sami o sebe, ale taktiež je k dispozícií aj GUI, kde je možné zadať hodnoty pre hyperparametre, stlačiť Train, a model sa natrénuje a uloží.

Toto GUI je napísané v demo.py. Využívam pre to knižnicu PySimpleGUI [11], čo je knižnica pomocou ktorej je skutočne ľahké vytvoriť nejaké GUI. Veľa pri tom využívam retrospection, čo je vlastnosť Pythonu, vďaka ktorej vieme zistiť veľa vecí o objektoch a funkciách počas behu programu.

```
# Intro window layout
layout = [
    [
        sg.Push(),
        sg.Text('Model training interface'),
        sg.Push()
    ],
    [sg.VPush()],
    [sg.Push(), sg.Text('Model:'), sg.Combo(
            model_names, size=(40, len(model_names)
        ),
        key = 'MODEL', enable_events=True), sg.Push()],
    [sg.VPush()],
]
```

Figure 8: Príklad na layout. sg.Text() je element obsahujúci nejaký text, sg.Combo() sa používa na zobrazenie listu možností, sg.Push() a sg.VPush() sú elementy slúžiace ako padding, vďaka nim vieme posúvať ostatné elementy a vytvoriť medzi nimi priestor. sg.Push() slúži na horizontálne a sg.VPush() na vertikálne usporiadanie.

Hlavná časť je funkcia main(), ktorá sa stará o vytvorenie okna a všetkých elementov v ňom, a taktiež je v nej nekonečný while loop, kde sa čítajú udalosti z okna. PySimpleGUI (ďalej sg) obsahuje triedu Window, ktorej vieme predať list listov layout, ktorý obsahuje jednotlivé elementy UI ako riadky. Môžeme tam mať napríklad nejaký textový element pomocou sg.Text, pole na užívateľský vstup pomocou sg.Input, rôzne tlačítka atď.

Layout je usporiadaný na riadky, každý list predstavuje jeden riadok a nakoniec sa vytvorí z celého okna dlaždicové usporiadanie zarovnané na najväčší počet elementov v nejakom riadku a počet riadkov. Jediný problém je, že keď už okno s nejakým layout vytvoríme, nevieme ho už dynamicky upravovať. Čiže ak chceme napr. zobraziť iné vstupné polia na hyperparametre pre rôzne modely, tak musíme vytvoriť nový layout a nové okno, a zatvoriť to staré. Naštastie je to celkom rýchly proces, takže to nie je pre užívateľa veľmi nepríjemné.

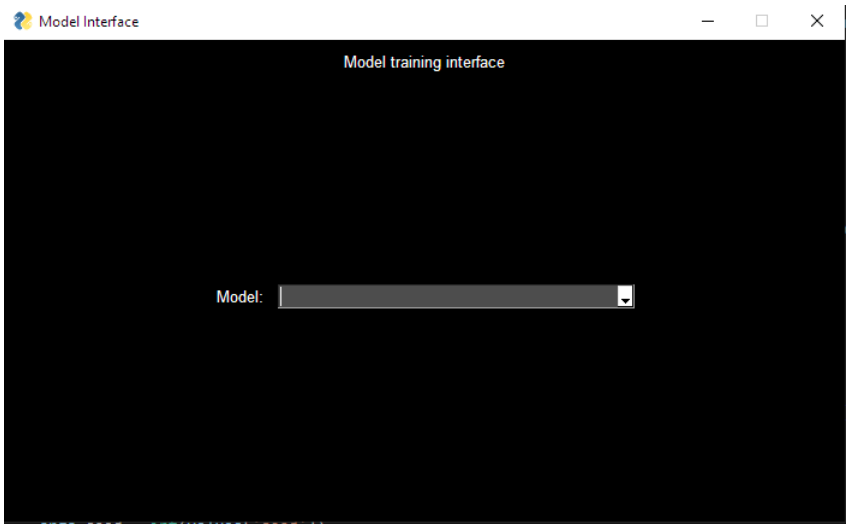

Figure 9: Úvodná obrazovka, vygenerovaná pomocou kódu v Figure 8

Najprv je vidieť úvodné okno, kde máme drop-down menu obsahujúce všetky dostupné modely.

Po výbere nejakého modelu sa nám okno premení podľa vybraného modelu tak, aby sme vedeli zadať rôzne hyperparametre. Každý hyperparameter má pred sebou jeden sg.Text, ktorý obsahuje jeho názov. Aby okná neboli príliš dlhé, tak maximálny počet elementov v jednom riadku je 8.

Po stlačení Train sa spustí trénovanie modelu, pričom môžeme vidieť postup v textovom boxe pod týmto tlačítkom. Po dokončení tréningu sa v ňom vypíšu rôzne metriky a model sa uloží.

Každému elementu sa dá priradiť vlastný kľúč, ktorý sa využíva vyhodnotenie toho, či užívateľ interagoval s týmto elementom alebo zadal nejakú hodnotu do poľa. Na získavanie dát z vstupných polí existuje slovník values, kde sú hodnoty uložené podľa ich kľúčov, ale nedá sa nastaviť príjmaný typ, všetko čo je tu uložené je string.

V main funkcii je list ktorý obsahuje trojice (model_func, preprocess_func, train_func), pričom z funkcií model_func a train_func sa pomocou retrospection zistia hyperparametre a názvy modelov.

Názvy získame pomocou funkcie get_model_names(), ktorá sa pre danú funkciu len pozrie na jej docstring a ako meno zoberie prvý riadok.

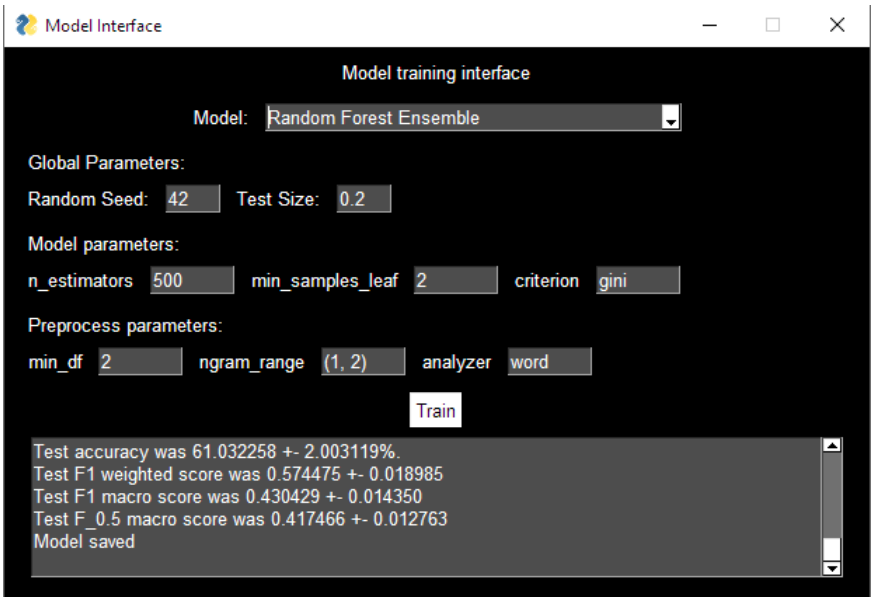

Figure 10: Výber modelov

Ak existujú nejaké default hodnoty, tak sa nastavia ako počiatočný text v poliach. Inak pole zostane prázdne. Ak užívateľ zadá niekde hodnotu, ktorá má nesprávny formát, dostane o tom hlásenie v textovom poli a tréning nezačne. Vytvorenie UI elementov pre dané hyperparametre, má za úlohu funkcia create_row_with_params. Pozrie sa na argumenty funkcie a spraví to, čo bolo povedané vyššie.

Existuje funkcia inspect.signature(), ktorá nám pre daný zavolateľný objekt vráti jeho signatúru vo forme Signature. Signature objekt obsahuje práve informácie o argumentoch, takže pri konverzií hodnôt z values sa použije buď typ anotácie pre daný parameter, alebo typ východzej hodnoty.

Nakoniec sa vytvorí vlákno, ktorému predáme train_func a parametre pre model a preprocess a ten nám do argumentov model a transformer uloží výsledok tejto funkcie. Tento train_func je práve ten training_wrapper, aby sa každý model dal jednoducho natrénovať z tohoto modulu.

Takže na zhrnutie main() sa stará celkovo o GUI, create_row_with_params() vytvorí riadok pre hyperparametre, get_model_names() nájde ich mená a thread_train_function() je daná nejakému vláknu a tá natrénuje model.

Jediná funkcia, o ktorej som nehovoril, je is_special_arg(), čo je len pomocná funkcia na zistenie, či sa pre daný argument má vytvoriť vstupné pole alebo nie.

7 Postup na vyskúšanie GUI

Na úvodnej obrazovke môžeme vybrať model, čo chcem natrénovať. Potom preň upravíme hyperparametre, pričom by mali byť v rovnakom formáte ako východzie hodnoty ktoré sa už v jednotlivých oknách nachádzajú. Potom stlačíme tlačítko Train, ktoré daný model natrénuje a uloží. Postup tréningu môžeme sledovať v textovom boxe pod tlačítkom Train (Figure 10).

8 Záver

Projekt sa skladá z niekoľko skriptov, ktoré zhromaždia dáta, pretransformujú ich do datasetov vhodných na trénovanie modelov a dá nám k dispozícií jednoduché rozhranie, pomocou ktorého vieme nastaviť hyperparametre modelov a natrénovať ich.

V budúcnosti by bolo dobré spraviť aj GUI pre všeobecné scrapovanie príspevkov s danými kľúčovými slovami a v danom časovom rozmedzí a taktiež na manuálnu anotáciu príspevkov.

Rozšíriteľnosť v zmysle pridávania nových modelov môže byť celkom priamočiara. Stačí, aby modul v ktorom sa daný model bude nachádzať, obsahoval nejakú funkciu podobnú training_wrapper(), a dodržoval zvyšok rozhrania definovaného v demo.py. Tým mám na mysli veci ako: argumenty na trénovanie obsahujú args, prep_args na správnej pozícií, vracanie tuple (model, transformer) atď.

Dokumentácia pre každú funkciu bude dostupná v priečinku Docs, bude len vygenerovaná z docstrings každej funkcie pomocou Sphinx.

References

[1] Christopher M.Bishop. “Neural Networks for Pattern Recognition”. In: 1996. Chap. 4.

[2] Jürgen Schmidhube Sepp Hochreiter. “Long short-term memory”. In: *Neural Computation* (1997), pp. 1735–1780.

[3] Akiko Aizawa. "An information-theoretic perspective of tf–idf measures". In: *Information Processing Management* 39.1 (2003), pp. 45–65. ISSN: 0306-4573. DOI: https://doi.org/10.1016/S0306-4573(02)00021-3. URL: https://www.sciencedirect.com/science/article/pii/S0306457302000213.

[4] Kyunghyun Cho et al. *On the Properties of Neural Machine Translation: Encoder-Decoder Approaches*. 2014. DOI: 10.48550/ARXIV.1409.1259. URL: https://arxiv.org/abs/1409.1259.

[5] Jimmy Ba Diederik P. Kingma. *Adam: A Method for Stochastic Optimization*. URL: https://arxiv.org/abs/1412.6980.

[6] *Gamestop website*. URL: https://www.gamestop.com/.

[7] *pandas DataFrame documentation*. URL: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html.

[8] *pandas documentation*. URL: https://pandas.pydata.org/docs/index.html.

[9] *PRAW (Python Reddit API Wrapper) documentation*. URL: https://praw.readthedocs.io/en/stable/.

[10] *PSAW (Pushshift.io API) repository*. URL: https://github.com/pushshift/api.

[11] *PySimpleGUI documentation*. URL: https://pysimplegui.readthedocs.io/en/latest/.

[12] *r/wallstreetbets subreddit*. URL: https://www.reddit.com/r/wallstreetbets/.

[13] *Reddit API*. URL: https://www.reddit.com/dev/api/.

[14] *Scikit-learn website*. URL: https://scikit-learn.org/stable/.

[15] *TensorFlow documentation*. URL: https://www.tensorflow.org/api_docs/python/tf.

[16] *Tweepy website*. URL: https://www.tweepy.org/.

[17] *Twitter API*. URL: https://api.twitter.com/.