Gymnázium, Praha 6, Arabská 14

Programování

Ročníkový Projekt

Timofej Kiselev

Language Classifier

Březen 2021

Prohlašuji, že jsem jediným autorem tohoto projektu, všechny citace jsou řádně označené a všechna použitá literatura a další zdroje jsou v práci uvedené. Tímto dle zákona 121/2000 Sb. (tzv. Autorský zákon) ve znění pozdějších předpisů uděluji bezúplatně škole Gymnázium, Praha 6, Arabská 14 oprávnění k výkonu práva na rozmnožování díla (§ 13) a práva na sdělování díla veřejnosti (§ 18) na dobu časově neomezenou a bez omezení územního rozsahu.

V Praze dne _____ Timofej Kiselev _____

Anotace

Language Classifier je projekt, který se zabývá klasifikaci textů podle jazyků. Podporuje 12 jazyků jejíchž seznam najdete v následujicí dokumentaci, nebo na stránce projektu v about. Součásti projektu je downloading a upravování textu z wikipedie, tvorba datasetu, učení machine learning modelu logistické regrese rozpoznávat jazyky a web interface. Projekt také kromě klasifikace jazyků určuje jaké z podporovaných jazyků text obsahuje a jejích přibližný začátek a konec. Projekt byl vytvořen v Google Colaboratory a PyCharm CE.

Language Classifier is a project that solves problem of classification texts by languages. It supports 12 languages, the list of which can be found in the following documentation, or on the project page in about. The project includes downloading and preparation of texts from wikipedia, creating a dataset, teaching a machine learning logistic regression model to recognize languages and a web interface. In addition to the classification of languages, the project also determines which of the supported languages the text contains and where is their aproximate beggining and end. The project was created in Google Colaboratory and PyCharm CE.

Zadání

Cílem práce je naučit machine learning model rozpoznávat v jakém jazyce byl psaný text, který dostane na vstup. (Klasifikace jaký jazyk to je z těch na kterých se učil). Součástí práce bude práce s velkými objemy dat a neurosítě. Učit se bude na základě textu z wikipedie v různých jazycích a zpracování datasetu z dat je také součástí práce.

Obsah

1. Úvod

1.1 Funkcionalita

Language Classifier, je software jehož hlavním účelem bylo vybrat a správně přiřadit k textu, který dostane na vstup, název jazyku ve kterém je text psaný, z jazyků, které program zná. V aktuálním stavu projektu dokáže rozeznat 12 jazyků. Tyto jazyky se dají najít jak v notebooku tak v views.py. Program dokáže rozeznat ve kterém z uvedených jazyků je text psán: čeština, ruština, francouzština, němčina, angličtina, polština, italština, ukrajinština, finština, bulharština, španělština, portugalština. Přidat nový jazyk do programu není těžké, jen v několika místech v kódu je potřeba přidat do listu jazyků nový jazyk a odkaz na wikimedia odkud soubor s texty v tomto jazyku bude stahovat ze kterých by se učil. Zbytek práce už splní automaticky.

Přidání nového jazyku by, ale vyžadovalo vytvoření nového modelu a proto celý proces by zabral přibližně (2-3 hodiny). Výhodou je ale to, že přidávání více jazyků naráz nezvyšuje čas tak výrazně a proces je automatizován. Hlavní ale je, že doopravdy ani není třeba spouštět část, která vytváří model, víc než jednou pro všechny jazyky co chcete, protože když model je hotový nebude zabírat moc místa a po jeho uložení se dá používat opakovaně do nekonečna. Mimo rozpoznání jazyků projekt se také zabývá segmentaci jazyků. To znamená, že navíc k rozpoznávání, které bylo primárním cílem, také naznačuje kde přibližně v textu jaký jazyk začíná a končí. Narozdíl od samotného rozpoznávání, segmentace v tomto případě nepotřebovala machine learning a byla uskutečněná algoritmicky. Nakonec k použití projektu byl v django vytvořen web interface, ve kterém se projekt dá jednoduše ovládat a také poskytuje návod k použití a přístup k dokumentaci.

1.2 Části projektu

Projekt Language Classifier se skládá z několika části. První až čtvrtá z nich jsou ve formě spustitelného kódu v google colab notebooku. Pátá a všechny ostatní části projektu jsou části, které jsou aktivní při samotném používání.

První části je stahování souborů s textem v různých jazycích a extrakce stažených souborů.

Druhá část je dostávání ze souborů samotných textů a jejich zpracování (převod na malá písmena, odstraňování specifických symbolů a tak dále). Poté zpracované texty se ukládají do nových souborů.

Třetí část vytváří dataset. Vytváří nové soubory s určitým počtem znaků a počtem řádek pro každý jednotlivý jazyk a poté je ve stejném poměru spojuje, promichává a rozděluje na train a test datasety, ze kterých každý má dvě položky. X se stringy určité délky, a Y s označením jazyku.

Čtvrtá část je tvorba a učení samotného modelu pomoci dřív vytvořeného datasetu. V tomto případě se používá sklearn knihovna a samotný model je logistická regrese.

Pátá část je kód v souboru classify.py, který používá uložený model ke klasifikaci a poté nachází kde přibližně jazyk začíná a kde končí.

Šestá část je django projekt s webovou aplikací, která funguje jako web interface projektu.

Sedmá je design web interface a různé nepatrné opravy a zjednodušení.

Za osmou bonusovou část považují pak dokumentaci, kterou zrovna čtete.

2. Tvorba projektu

V této kapitole se popisují některé nástroje, které byly použity při tvorbě projektu.

Python 3 je interpretovaný programovací jazyk ve kterém je vytvořen skoro celý projekt. Byla použita řada knihoven, některé z nichž zmínim také v této kapitole.

Google Colab nebo také Google Colaboratory je projekt založený na Jupyteru s tím rozdílem, že je cloudový. Notebook běží na serveru u google, jako úložiště používá google disk a také google poskytuje bonusy jako jsou například grafické karty, které jsou pro mě osobně jako člověka co se učí machine learning, ale jeho poslední kvalitní grafická karta je už několik let nefunkční, velice užitečné.

Jupyter (a jupyter lab, který jsem taky používal dříve, než přešel na google colab) je open source projekt založený na IPython, který poskytuje webové IDE s možnosti spouštět jednotlivé části kódu v různém pořadí a dalšími výhodami.

PyCharm CE je IDE od JetBrains. Používal jsem ho pro práci s Django. Pro data science typ práce preferuji Jupiter nástroje, ale pro práci s věcmi typu Django neznám pohodlnější IDE než PyCharm. Velice urychluje práci když můžu mít v jednom okně editor kódu, terminal, výstupy programu a file manager. Navíc má funkce, jako jsou například změna názvu proměnné všude kde byla použita a dokáže editovat bez problému i jiné soubory než .py, například .html.

Django je open source framework pro psaní webových aplikací napsaný v pythonu. Byl použit pro webový interface projektu.

Bootstrap je CSS framework, který byl použit pro design. Mimo jiné pro web interface byla použita řada dalších věcí jako například html a javascript.

Regular expression, nebo také regex, byl použit pro pohodlnější zpracovávání části textů.

Bylo použito hodně knihoven pro python 3 ze kterých chci zmínit tyto:

Pickle je určen k serializaci. V dobách kdy jsem se účastnil Kaggle soutěže jsem si pickle oblíbil pro jeho rychlost ukládání a načítání dat.

Pandas je knihovna určena k práci s daty. Je to můj oblíbený nástroj pro analýzu dat a v projektu se uplatnil při vytváření datasetu.

Numpy zjednodušeně řečeno rozšiřuje matematiku pythonu spoustou velice užitečných funkcí.

Sklearn je knihovna, která poskytuje vše potřebné k základům machine learningu. Obsahuje vše potřebné pro řešeni typických machine learning úloh i když má několik nedostatku customizace a chybí hodně nových pokročilejších nástrojů. Výhodou ale je, že vše co má dobře pracuje v kombinaci s ostatními věcmi co má a dají se spolu používat bez výskytu nečekaných problému.

3. Jak projekt funguje

3.1 Wikimedia Downloads

Jedním z hlavních problémů při tvoření jakéhokoliv machine learning modelu, jsou data. Ať algoritmy budou jakkoliv kvalitní a technické vybavení efektivní, bez dat na kterých by se mohl učit, nebude k ničemu. Takže jako první, ještě před výběrem tématu tohoto projektu, jsem se zamyslel nad tím, jaká data by mohla být potřeba. Zcela zřejmá odpověď je, že k rozpoznávání různých jazyků textů, jsou potřeba texty psané v různých jazycích. Zjistil jsem o tom, že wikipedie texty jsou všem volně dostupné ke stažení a že existují stránky jako například https://dumps.wikimedia.org/, určené právě k tomu. Nakonec se ukázalo, že textů je příliš hodně a na řešení problému, kterým se můj projekt zabývá, stačí daleko méně textů než jsou tam dostupné, ale omezit množství se dá vždy, když to získat více dat je obvykle problém. Dá se říct, že jsem měl štěstí, že s dostatkem textů na učení nebyl problém. Problém by byl jejich stahování, protože zabírají hodně místa na disku a moje připojení je pomalé, ale použití google colabu tyto problémy řeší.

3.2 Exporting textů

Po extrakci získaných souborů jsem zjistil, že s nimi nemůžu pracovat jak jsem zvyklý, protože jsou příliš velké aby se dali načíst do paměti. Když jsem hledal co s tím, našel jsem na internetu, co s takovými soubory. A také youtube video návod na načítání wikipedie souborů i s kódem na githubu od Jeffa Heatona. Nebylo to úplně to, co jsem potřeboval, ale stejně mi velice pomohlo a nakonec jsem dokázal načíst ze svých souborů texty. Texty jsem dále ukládal do csv a načítal do pandas dataframe kde s nimi dál pracoval. Poté se z textů odstraňují speciální symboly, které nejsou součásti textu, převést na malá písmena, odstranit čísla, vícenásobné mezery a další. Protože některé jazyky používají dost specifické symboly, nemohl jsem použít regex na jejich odstranění v tomto případě, takže jsem musel vypsat nejčastější sám. Poté jsem musel odstranit příliš krátké články a zbylé zpracované texty uložil jako useky po 200 znacích.

3.3 Tvorba datasetu

Takto se text zpracovává pro každý jazyk zvlášť a je pomoci pickle uložen pro následující tvorbu datasetu. Soubory se načítají a z nich se stejné množství řádek, o délce 200 znaků, (ve výsledném modelu stačilo 100000 řádek) spojuje do jednoho, ke každému se vždy přidává i označení v jakém jazyce úsek je, a úseky se náhodně promíchávají. Vzniká jedna tabulka s úseky textů o stejné délce s informací o tom v jakém jazyce je daný úsek. Později kolonku textů jsem označil jako X a kolonku jazyku jako Y. Vzniklý dataset jsem rozdělil na 2, train a test, kde train tvoří 70% řádek a je určen k učení modelu. Test tvoří zbylých 30% a slouží ke kontrole efektivity modelu na datech, se kterými se dřív nesetkal. To slouží například k detekci toho, že model si prostě zapamatoval data z train. A také ukazuje úspěšnost blízkou k té, kterou bude mít při reálném užívání.

3.4 Co je machine learning

Machine learning, nebo taky v češtině strojové učení, se říká algoritmům, které se dokážou učit. V tomto případě to znamená, že se dokážou přizpůsobovat konkrétni úloze, kterou musí řešit, a zlepšují se v její řešení. K učení potřebují dostávat data toho typu, které budou dostávat na vstup při exploataci, a také odpovědi k nim (v případě, že jde o supervised learning, jako v případě tohoto projektu). V průběhu učení se model snaží změnit své parametry tak, aby dával odpovědi, které se budou co nejvíc blížit správným. Dá se na to koukat jako na matematickou funkci, která dostane data jako proměnné, a funkční hodnota pro tyto data je odpověď, kterou musí vrácet.

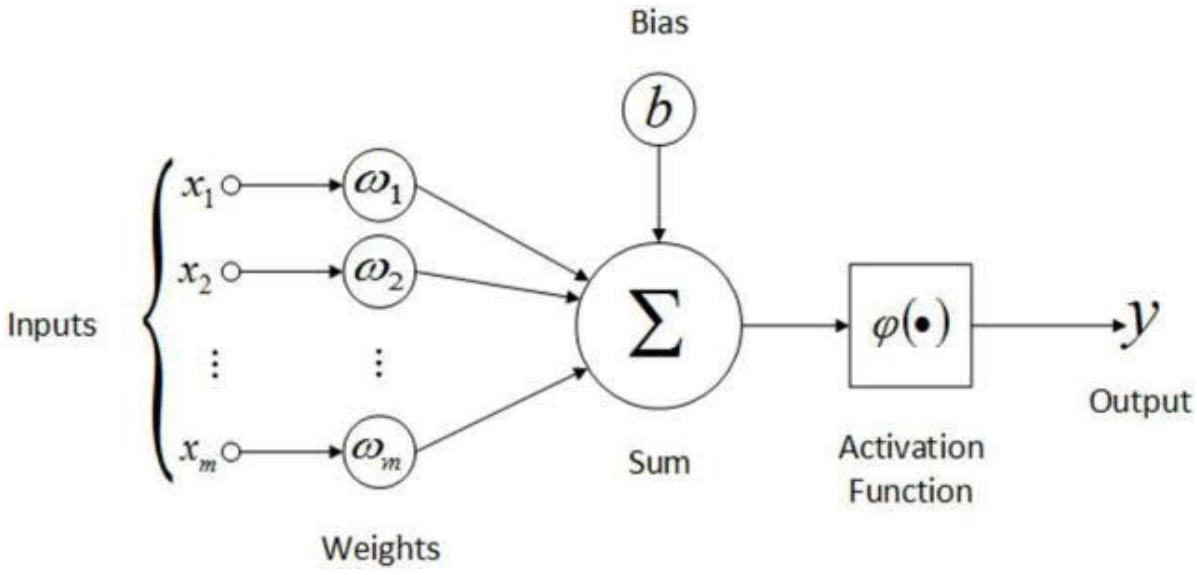
Machine learning se často používá pro úlohy, které je problematické řešit jinými způsoby. Typické příklady takových úloh, řešených pomoci machine learningu, jsou úlohy klasifikace, regrese, lokalizace, segmentace, ale i spousta dalších jako například natural language processing, generace nebo doplňování, komprese a tak dále. Pro tento projekt jsou důležité základní, již zmiňované, typy. Regrese se říká úloham, které mají za cíl ze známých dat spočítat výslednou číselnou hodnotu. Klasifikace se říká úlohám, které mají za cíl podle dat zařadit to, k čemuž se data vztahují, do nějaké z kategorií.

Do machine learningu se řadí hodně velmi odlišných věcí. Jako příklad uvedu často používané decision trees a neural networks, ale i těch je velké množství. Existuje ještě větší množství různých algoritmů, jak jednodušších tak složitějších, nebo jejich kombinací (například ensemble learning), ale v další kapitole se budu věnovat základům toho, jak fungují jen neural networks (neuronové sítě), protože to je metoda, která se v tomto projektu vyskytuje.

3.5 Úvod do neurosíti

Základní stavební jednotkou neuronové sítě je umělý neuron, který je zároveň nejjednodušším modelem neuronové sítě, kterému se říká perceptron. Perceptron již sám o sobě dokáže řešit některé lineární úlohy. Dokáže například předpovídat hodnotu neznámé lineární funkce podle proměnných, nebo rozdělit body do dvou kategorií, pokud jdou rozdělit přímkou, nebo rovinou. Obecně řečeno najít body ležící na přímce, nebo hyperplane v n dimenzionálním vektorovém prostoru, nebo rozdělit n dimenzionální vektorový prostor pomoci hyperplane na dva poloprostory a tím rozdělit body v něm do dvou kategorií.

Na obrázku č. 1 můžete vidět jak funguje perceptron. Inputs jsou hodnoty, které dostane na vstup. Fungují jako proměnné funkce. Poté se každá z nich násobí odpovídající váhou a výsledky se sečtou. Váhy jsou parametry neuronů, které se mění v procesu učení. Poté se ještě přičte bias, který udá posunutí a je dalším učitelným parametrem neuronů. Výsledek předchozích operací se obvykle označuje jako z. Tento výsledek z se podává na vstup do activation funkce. Activation funkcí je velké množství. Různé activation funkce mají své výhody a nevýhody. Může být jednoduše y=z a tudíž jako kdyby nebyla. Activation funkce ale obvykle lineární nebývá. Může být, například jako ReLU, pro záporné z rovná y=0 a pro kladné z být y=z. Často jsem se setkal také například s funkcemi Leaky ReLU, tanh, nebo sigmoid. Sigmoidě se ještě budu dále věnovat, protože se používá v tomto projektu.



Obrázek 1: Perceptron

Nakonec hodnota activation funkce, je hodnotou, kterou dostanete jako výstup z neuronu. Tento výstup už může být řešení, u některých jednoduchých úloh, nebo být použít jako vstup pro další neuron. Perceptron sice řeší některé úlohy dobře (data se často dají zpracovat takovým způsobem, aby se úloha dala řešit jednodušším modelem), ale ve většině případů je potřeba komplikovanější architektura. Například XOR (úplná disjunkce) se nedá udělat z méně než tří neuronů. Neurony mohou být uspořádány jak za sebou, tak ve vrstvách vedle sebe. Kromě vstupní a výstupní vrstvy se ostatním obvykle říká skrytá vrstva. Všechny neurony v první skryté vrstvě dostávají na vstup každou proměnnou z inputu. Každý neuron má svoje vlastní váhy a biasy, ale activation funkce je obvykle pro celou vrstvu stejná (obvykle i pro všechny skryté vrstvy a až výstupní má často jinou). Dále každý výstup každého z neuronů první vrstvy se podává jako vstup každému neuronu další vrstvy. A tak se to opakuje pro všechny vrstvy.

Teď zůstává otázka, jak se vlastně neurosíť učí? Aby se neurosíť naučila, je třeba upravit váhy a biasy. Původně se dají použít náhodné hodnoty. Dál je potřeba učit. K učení bude potřeba ještě loss funkce. Loss funkcí je taky hodně a vybírají se podle toho, jaký typ odpovědí a jaké hodnoty mají být výstupem neurosítě. Jednou z nejjednodušších, ale používaných loss funkcí je dokonce funkce MSE (mean squared error, nebo také v češtině rozptyl, nebo střední kvadratická chyba/odchylka), která je součásti středoškolské statistiky. V tomto projektu se řeší problém multi-label classification (narozdíl od binární klasifikace, tady jazyků je víc než dva a tudíž i kategorií taky). Typicky používánou loss funkcí pro multi-label classification je cross entropy a přesně ta se v tomto projektu používá. Loss funkce se používá pro výpočet toho, jak moc se výsledek neurosítě liší od očekávaného. Cílem je upravit parametry sítě takovým způsobem, aby se hodnota loss funkce blížila co nejvíce nule. Teď v tuto chvíli máme proměnné které jsou inputem, a tak protože jsou proměnnou funkce (neurosítě), jejíž hodnota je proměnnou loss funkce, jde o složenou funkcí. Proměnné a hodnotu loss funkce se dají taky představit jako bod v n dimenzionálním vektorovém prostoru. Všechny parametry funkce se dají představit jako list W0 = [w1, w2, ... , wn, b1, b2, … bn] a pro jejich úpravu je potřeba spočítat gradient. Gradient je vektorem derivací, v tomto případě loss funkce podle jednotlivých parametrů neurosítě. Pomoci chain rule (řetězové pravidlo, nebo pravidlo o derivaci složené funkce) se dá dojít až k derivacím loss funkce podle jednotlivých parametrů neurosítě, a jejich vektor bude gradient a udává směr největšího přírůstku v daném bodě. Pak pomoci gradientů se vypočítají nové hodnoty parametrů

funkce podle vzorce: W1 = W0 - alpha * gradient, kde alpha je learning rate (konstanta udávající rychlost, ale i přesnost učení se) a kde W1 jsou nové parametry funkce.

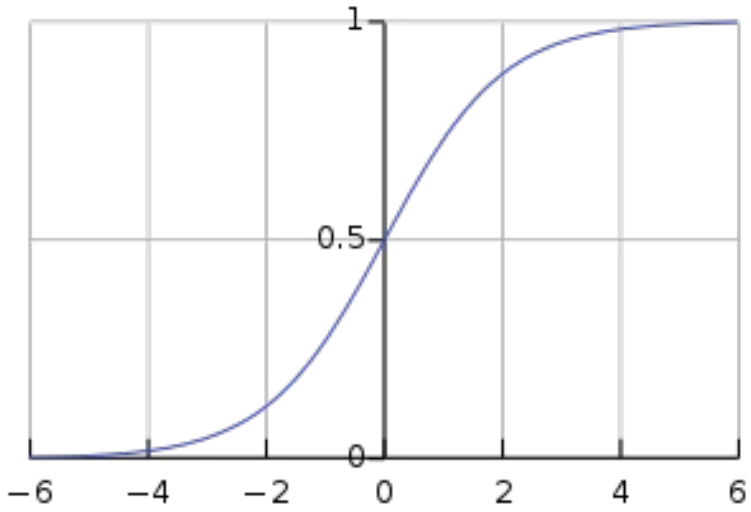
Výsledek vypadá jako krok proti směru největšího přírůstku loss funkce, takže směrem k lokálnímu minimu loss funkce. Takový krok se opakuje pro různá data v train datasetu a říká se tomu backpropagation (algoritmus zpětného šíření chyby). Problém dané metody je hlavně ten, že se neví žádný způsob zjistit, jestli minimum nalezené po několika takových krocích je opravdu nejlepší možný řešení, nebo ne. Samozřejmě existuje hromada chytrých triků jak v celý proces zjednodušit a urychlit. Například ukládání gradientů u předchozího kroku, nebo měnění learning rate konstanty v průběhu výuky, nebo i přidat dropout, který bude zmenšovat některé parametry aby neměli takový vliv a neurosíť si hůř vzpamatovávala výsledky a líp se učila obecně, ale takových triků je hodně a popisovat je zvlášť nemá smysl.

Teď když je vysvětlený hlavní princip fungování neurosítí se můžeme přesunout k vysvětlování toho, jak funguje konkrétně tento projekt.

3.6 Klasifikace jazyků

K realizaci klasifikace jazyků projekt používá knihovnu sklearn. Text se rozbíjí na tokeny, které jsou v tomto případě slova delší než dva znaky, protože krátká slova typu "do" nebo "a" jsou velice časté v různých jazycích a mají malou výpovědní hodnotu. Dále k vektorizaci slov se používá TfidfVectorizer. TF-IDF je metoda používaná k výpočtu toho, nakolik je každé slovo specifické a důležité. Minimální nutný počet výskytů slova v textech jsem nastavil na 50, protože menší budou buď velmi specifická slova, která se nikde nepoužívají, nebo vzácné překlepy, jména, názvy a další, které nijak nepomůžou poznat text. Jelikož neklasifikují téma textu, ale jeho jazyk, tak mi nevadí, že slovo se používá ve velkém množství textů, pokud je pořád několik jazyků, kde se nevyskytuje, tak to může znamenat, alespoň to, že text ve kterém je, skoro určitě není jeden z těch jazyků. Proto hodnota je nastavena tak vysoko, aby odstraňovala slova, která by se vyskytovala tak často, že se musí vyskytovat v textech psaných jak v latinkou, tak cyrilikou, a tudíž pravděpodobně bude nějaký z wikipedia tagu, nebo něco podobného, a neměly by se počítat. Teď když víme jaká slova jsou důležitá a jak moc, můžeme tyto informace o slovech, které máme v prvku datasetu, předat neurosíti. Neurosíť se v tomto případě používá LogisticRegression. Přesto, že v názvu logistická regrese je slovo regrese, používá se k řešení

úloh klasifikace. Logistická regrese znamená, že bude počítat pravděpodobnost jevu. V případě neurosítě tohoto projektu se skládá jen z jedné vrstvy. V této vrstvě je tolik neuronů, kolik jazyků klasifikuje. Na vstup dostává informaci o slovech jako input do každého z neuronů. Pro každé z nich má každý neuron různé váhy a výsledek předá do aktivační funkce. U logistické regrese je běžnou aktivační funkcí sigmoida. Má tu vlastnost, že jakýkoliv výsledek převede na pravděpodobnost. Je to rostoucí funkce od mínus nekonečna do nekonečna, která nabývá hodnot od 0 do 1 a udává pravdepodobnost jevu. Graf typické sigmoidy můžete vidět na obrázku č.2.



Obrázek 2: Sigmoid funkce

3.7 Segmentace

Poté vytvořený model se ukládá a načítá ze souboru pro následující využití. Na splnění zadání by to už stačilo. Pozná v jakém jazyce je psaný text, nebo jaký jazyk převažuje, ale já chtěl udělat ještě aby projekt uměl rozeznávat jednotlivé jazyky v textu když je jich víc a určovat jaká část je psaná v jakém jazyce. Tento problém jsem už řešil algoritmicky s využitím modelu na klasifikaci části textu, ale hledání přibližných hranic kde jeden jazyk přechází v druhý, je řešeno bez použití machine learningu. Tím se zabývá kód v souboru classify.py. Další problém je, že text co se dává do modelu se musí předem upravit, ale text, který se vrací uživateli s vyznačenými jazyky v něm musí obsahovat alespoň čísla a interpunkci, i když formátovaní, mnohonásobných mezer a podobných není problém se zbavit. Formátování textu je dokonce zablokované javascript

skriptem pro kopírování textu do inputu projektu. Program taky ignoruje text, který bude zapsaný mezi <> závorkami, aby se nepokoušel vyhodnocovat formátování jako jazyk a uživatelé neměli možnost donutit program použít jejích css, nebo i javascript. Samotná segmentace funguje tím způsobem, že jedna kopie originálního inputu bez textu mezi <> závorkami a nových řádku s vícenásobnými mezerami se uloží pro následující využití a návrat uživateli. Ke druhé kopií se použijí úpravy textu, které se používaly k textu, které model dostával na vstup když se učil. Dál se rozdělí na úseky po 5 slov, projdou se a zaznamená se jejích předpověď a nakolik je model jistý touto předpovědí. Poté se zkontrolují všechny prvky podle jejích sousedů, protože 5 slov rozhodně není dost pro správné vyhodnocení, a je třeba zkontrolovat jestli pár jazyků vyhodnocených jako jiný je opravdu jiný, nebo jen nebylo dost slov na jeho vyhodnocení. Po částečném spojení vznikají úseky textu s označeným jazykem, ale protože byli vybrané po 5 není jisté kde je hranice mezi jazyky ani přibližně. Musí být někde v rozmezí 10 slov na hranici mezi 5 posledními předchozího úseku a 5 prvními následujícího, ale ve vzácných případech kdy 5 prvních slov nového jazyku byli vyhodnoceny chybně, můžou spadat i 5 prvních a posledních slov dvou vedlejších úseku. V těchto místech se zas text rozdělí. V případě posledního úseku, který často nemá ani 5 slov, protože počet slov co uživatel zadá není vždy dělitelný pěti se poslední úsek vyhodnocuje spolu s předposledním. Dále se nachází hranice v jaké z části je a při tom se používá vyhodnocování pravděpodobnosti jazyku jednotlivých částí z modelu kde 2 krajní úseky už jsou dlouhé úseky a hledaná hranice je v krátkých mezi. Program zkouší jakou kombinací když vezme dosáhne stabilnější polohy hranice a už se nebude měnit když po další změně zkusí znovu. Pokud takovou polohu nenajde po několika pokusech (například pokud je velký úryvek jazyku co nezná, ale model různé časti toho úryvku považuje za různé, které zná) přestane zkoušet. Dále vyhledá jakým slovům v originálním textu odpovídají slova úseku, které klasifikoval a vrací originální text rozdělený podle úseků různých textů a označení jaký jazyk to je.

3.8 Web interface

Projekt také potřeboval, aby lidé ho mohli pohodlně používat a jako nejlepší možnost jsem si vybral udělat pro projekt web interface. Protože django je taky napsaný v pythonu, předešel jsem hodně problémům s načítáním a používáním modelu. Hned při spuštění se načte model a bude se čekat než uživatel vloží text a zmáčkne tlačítko "Detect Language". Program kontroluje jestli

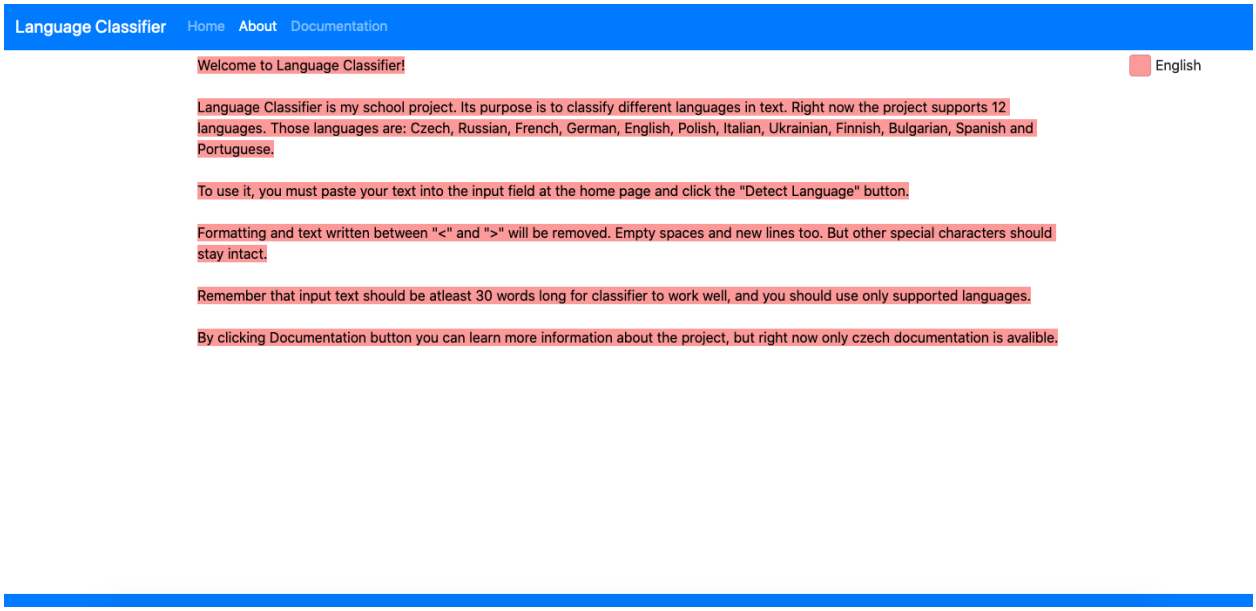
input nebyl příliš krátký, odstraňuje z již popsaných důvodu vše psané mezi <> závorkami a ruší nové řádky a vícenásobné mezery a tak dále. Po nutných opravách pokud text není vhodný ke klasifikaci, vypíše základní požadavek vložit několik vět delších než 5 slov a zmáčknout tlačítko. Pokud to vypadá, že vše vyhovuje podmínkám, provede s vloženým textem vše popsané v předchozí kapitole. Dále se k jazykum přiřazuje jejich barva a pro hranici mezi jazyky se nastaví přechod mezi těmito barvami jako gradient. Poté se vše uloží jako dictionary. Uživateli se po všem vrátí mírně upravený text, který zadal, ale s barevně vyznačenými jazyky. Jak to vypadá můžete vidět na obrázcích v následující kapitole.

3.9 Design

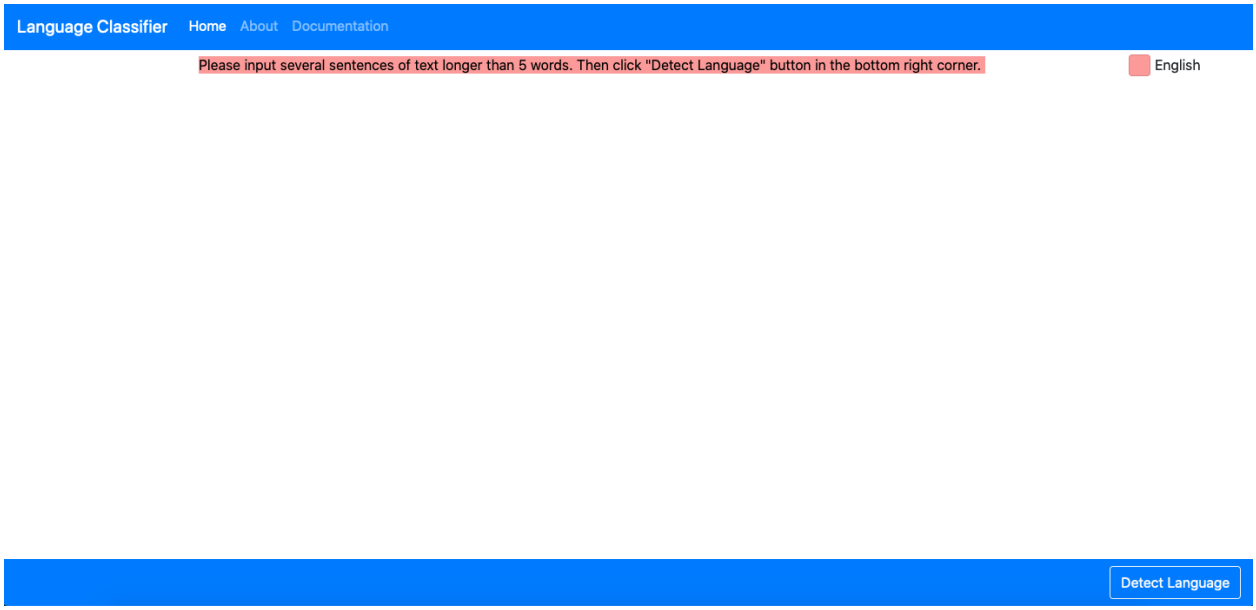
Pro design projektu byl používán hlavně material design pomocí bootstrapu. První řešený problém byl ten, že obyčejný textarea nepodporuje barevný text ani pozadí. Původně jsem chtěl udělat barevný text a ne pozadí, ale zjistilo se, že barevný text při přechodu mezi barvami je často nečitelný a nevypadá to tak dobře, protože se barva mění i když je mezera a žádný text není vidět. Z toho důvodů jsem se nakonec rozhodl pro barevné pozadí. Aby se povedlo vyznačit části textu jsem původně uvažoval o použití prosemirroru, ale nikdy jsem s ním nepracoval a nevím jak se dá použít s django. Nakonec jsem našel řešení a udělal to pomocí contenteditable="True". Barvy textu byly vybrané pomoci jednoho webu, který najdete ve zdrojích, takovým způsobem, aby se jednotlivé barvy dali lidským okem jednoduše odlišovat. Vedle okna, kde uživatel může zadávat text je legenda s vysvětlením jaký jazyk znamená jakou barvu. Na obrázku č. 3 můžete vidět jak to vypadá, se všemi jazyky co projekt umí rozpoznat. Texty v různých jazycích, které v tomto příkladu se rozeznávají, jsou také vzaty z wikipedie z náhodných článku v nahodných jazycích a nevím sám co se ve kterém z nich říká.

Obrázek 3: Screenshot, příklad klasifikace

Další funkce web interface je „About“, kde se v angličtině říká jak se má projekt používat. Říká se tam přibližně to samé co v následující kapitole. „Documentation“ otevře tuto dokumentaci jako pdf.



Obrázek 4: Screenshot, about



Obrázek 5: Screenshot, home

4. Jak správně používat

Po zapnutí vás bude vítat text „Please input several sentences of text longer than 5 words. Then click "Detect Language" button in the bottom right corner.“. Zadaný text musí být pokud možno alespoň 30 slov dlouhý, bez gramatických chyb a překlepů. Veškeré formátování a text psaný mezi „<“ a „>“ závorkami, včetně těchto závorek, budou odstraněny, ale většina speciálních symbolů bude ponecháno. Jazyk musí být jedním z podporovaných jazyků. Podporované jazyky jsou: čeština, ruština, francouzština, němčina, angličtina, polština, italština, ukrajinština, finština, bulharština, španělština a portugalština. Text můžete jednoduše zkopírovat a vložit. Poté co text bude připraven stačí zmáčknout tlačítko s nápisem „Detect Language“ v pravém dolním rohu. Po několika sekundach výsledek klasifikace bude barevně vyznačen.

5. Budoucí plány

V budoucnu je potřeba udělat několik změn, aby se předešlo omezením, které momentální verze má.

První omezení, které by se v budoucnu odstranilo je způsob tokenizace textu. V daném provedení model vnímá každé slovo jako celek. Hlavní nevýhodou je to, že model pak není schopen vnímat jazyky jako je například japonština. V první verzi jsem japonštinu zkoušel, ale vyšlo to tak, že jen nazýval japonštinou vše co nešlo nazvat něčím jiným, nebo nedávalo smysl. Například protože úsek textu byl příliš krátký. To je způsobeno tím, že japonština v textu na internetu, včetně wikipedie, nemá mezery. Mezery jsou součásti znaku pro závorky, a interpunkci. Mezi slovama ve větách nejsou vůbec a na rozeznání jednotlivých slov ve větách nejsou pro japonce potřeba. Tento problém by se dal řešit jiným způsobem tokenizace textu. Při použití n-gramů místo slov by se vnímali části slov a absence mezer by nevadila. To by také mohlo zvýšit přesnost modelu při překlepech, velkém množství atypických slov, a také umožnilo vnímat koncovky, kořeny slov, předpony a přípony zvlášť. Přesto myslím, že zlepšení pro typické evropské jazyky by bylo nepatrné. V případě nutnosti přidat podobné velice odlišné jazyky, bych doporučoval použít knihovnu YouTokenToMe, která na moment psaní této části byla považována za nejrychlejší tokenizer s chytrým použitím n-gram.

Další problém je, že algoritmus na segmentaci textu není nejlepší možný. Nebyl sice cílem projektu, ale stejně by bylo dobré, v případě oficialního release projektu, ho upravit. Jako v předchozím případě problémy přidávají specifické jazyky, v tomto případě to byla arabština. Arabština se píše od prava do leva a vznikají problémy jak s gradienty, protože se sekají v průběhu a pokračují na začátku věty z druhé strany řádky, tak s divnými posuvy. Řešení se skládá ze dvou části a mělo by možnost vylepšit přesnost segmentace i u jiných jazyků. Algoritmus by zůstal velice podobný aktuálnímu, jen by místo korku o velikosti několik slov (vzhledem k předchozímu vylepšení, se místo slov dá to samé udělat pro n-gramy), bylo posunutí pokaždé o jedno slovo. Místo proměnné krok by byla proměnná pro velikost okna, která by udávalo kolik tokenu by pokrývalo naráz a posouvalo se s každým krokem o jeden dopředu. Pro každý posun by model říkal k jakému jazyku úsek patří a tato informace by se zapisovala do listu délky všech tokenu textu tak, že nakonec by byl list obsahující informaci pro každou pozici,

kolikrát se token na dané pozici řadil modelem k úseku jakého jazyku. Zároveň s tím by se zapisovala pravděpodobnost daného vysledku, který by se získal použitím funkce softmax. Funkce softmax se používá když je třeba určit při klasifikaci pravdepodobnost každého vysledku z hodnot které jednotlivé výstupy mají (dostat hodnoty do rozmezí od 0 do 1 tak, aby dohromady dávali 1). Pro tokeny co nejsou na začátku a na konci textu by bylo tolik výsledku kolik je délka okna. Poté na základě pravděpodobností pro různé jazyky a té stejné informaci u přímých sousedů těchto slov, by se dalo zařadit jako patříci do úseku daného jazyku. Dále u jazyku kde se píše zprava doleva na začátku a konci musel najít usek odpovídající jedné větě a gradient dávat na obrácený konec okrajových vět. Tato kombinace okna segmentace a funkce softmax by také mohla ještě víc zmenšit nepřesnost segmentace u všech jazyků o pár slov.

V budoucnu by taky nebylo špatný umožnit nahrávat soubory s textem v různých formátech a provádět klasifikaci jazyků v nich, ale mimo přímého rozšíření funkcionality o tuto možnost, by to vyžadovalo i dodatečné kontroly souborů pro bezpečnost, protože se souborem se dá udělat daleko víc škod než jen s textem.

Nakonec zbývá úprava před releasem, která by neovlivnila funkcionalitu projektu, je reklama. Design projektu umožnuje zcela neškodně pro funkcionalitu umístit reklamu buď zleva v prázdném místě, nebo dole pod textem uprostřed stránky v nav baru s tlačítkem Detect Language. Byla by to praktická úprava, protože by umožnila z projektu vydělávat.

6. Závěř

Výsledkem tohoto projektu je program, který dokáže rozeznat 12 jazyků a vyznačit jejích přibližnou polohu v textu. Dosahuje až 96.4 procentní úspěšnosti v klasifikaci známých jazyků, což splňuje zadání a je velice dobrý výsledek. V průběhu práce jsem se naučil nové věci a zopakoval to co už věděl. Kromě vyřešených problémů práce také nabízí řešení problému, které zatím nebylo potřeba řešit, ale dají se použít v následujícím vývoji projektu. V dokumentaci se také vysvětlují základy neuronových síti, aby principu fungování projektu mohli porozumět i lidé, kteří téma machine learningu neznají. Kromě splněného zadání součástí práce je také příjemně vypadající web interface. Web interface je přehledný, praktický a slouží také pro přístup k dokumentaci a návodu k projektu.

7. Nástroje použité pro tvorbu

Google Colaboratory https://colab.research.google.com/notebooks/intro.ipynb
PyCharm CE https://www.jetbrains.com/pycharm/
Mdbootstrap https://mdbootstrap.github.io/bootstrap-material-design/
COLORBREWER 2.0 https://colorbrewer2.org/
Scikit-learn https://scikit-learn.org/stable/index.html
Django https://www.djangoproject.com/
Python https://www.python.org/about/
Google docs https://www.google.com/docs/
LibreOffice https://www.libreoffice.org/
Git https://git-scm.com/
Github https://github.com/

8. Seznam Obrázků

9. Zdroje

HEATON, Jeff. Read_wikipedia. *Github* [online]. Sep 12, 2019 [cit. 2021-03-28]. Dostupné z: https://github.com/jeffheaton/present/blob/master/youtube/read_wikipedia.ipynb

Stackoverflow [online]. [cit. 2021-03-28]. Dostupné z: https://stackoverflow.com/questions/18854620/whats-the-best-way-to-split-a-string-into-fixed-length-chunks-and-work-with-the/18854817

Stackoverflow [online]. [cit. 2021-03-28]. Dostupné z: https://stackoverflow.com/questions/49204225/python-split-string-every-three-words

Stackoverflow [online]. [cit. 2021-03-28]. Dostupné z: https://stackoverflow.com/questions/23996118/replace-special-characters-in-a-string-python

Scikit-learn [online]. [cit. 2021-03-28]. Dostupné z: https://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html?highlight=pipeline%20tfidf

Samsung-IT-Academy. *Github* [online]. Oct 30, 2020 [cit. 2021-03-28]. Dostupné z: https://github.com/Samsung-IT-Academy/stepik-dl-nlp

Cntk [online]. [cit. 2021-03-28]. Dostupné z: https://www.cntk.ai/pythondocs/CNTK_103B_MNIST_LogisticRegression.html

Stackoverflow [online]. [cit. 2021-03-28]. Dostupné z: https://stackoverflow.com/questions/16566069/url-decode-utf-8-in-python

WEBER, Rio. *Medium* [online]. May 8, 2018 [cit. 2021-03-28]. Dostupné z: https://medium.com/riow/django-static-files-dcc3f65a8f3a

Stackoverflow [online]. [cit. 2021-03-28]. Dostupné z: https://stackoverflow.com/questions/8956567/how-do-i-make-an-editable-div-look-like-a-text-field

Stackoverflow [online]. [cit. 2021-03-28]. Dostupné z: https://stackoverflow.com/questions/7355583/can-div-with-contenteditable-true-be-passed-through-form

Stackoverflow [online]. [cit. 2021-03-28]. Dostupné z: https://stackoverflow.com/questions/39884260/is-it-possible-to-set-horizontal-gradient-to-text-via-css-left-letter-one-colo

Scikit-learn [online]. [cit. 2021-03-28]. Dostupné z: https://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html?highlight=pipeline%20tfidfeme=Paired&n=12

Stackoverflow [online]. [cit. 2021-03-28]. Dostupné z: https://stackoverflow.com/questions/58980235/stop-pasting-html-style-in-a-contenteditable-div-only-paste-the-plain-text

Cloudinary [online]. [cit. 2021-03-28]. Dostupné z: https://res.cloudinary.com/practicaldev/image/fetch/s--7y0EX4vc--/c_limit%2Cf_auto%2Cfl_progressive%2Cq_auto%2Cw_880/https://raw.githubusercontent.com/DrakeEntity/project-Image/master/1_8VSBCaqL2XeSCZQe_BAyVA.jpeg

Wikimedia [online]. [cit. 2021-03-28]. Dostupné z: https://upload.wikimedia.org/wikipedia/commons/thumb/8/88/Logistic-curve.svg/320px-Logistic-curve.svg.png