Strojové učení a rozpoznávání
2019 / 2020

Projekt SUR - Klasifikátor osoby

Vypracoval: Jan Lorenc (xloren15) Datum: 1. 4. 2020

Obsah

1. Zadání

Úkolem projektu bylo rozpoznat jednu konkrétní osobu mezi jinými lidmi na základě obrázku a hlasové nahrávky. K dispozici jsme dostali trénovací i testovací data, pomocí nichž jsme měli natrénovat klasifikátory řeči a obrazu, abychom dosáhli co nejlepších výsledků při klasifikaci reálných dat.

2. Motivace

Pro můj rostoucí zájem o strojové učení jsem si chtěl na projektu vyzkoušet co nejvíce klasifikační technik, třeba i nad rámec zadání. Můj cíl byl zkusit vytvořit klasifikátory založené na vlastním matematickém výpočtu, modelech dostupných v knihovnách Pythonu zaměřených na umělou inteligenci nebo i natrénování neuronové sítě. Ve výsledku bych se rozhodoval na základě hlasování jednotlivých klasifikátorů, čímž bych dosahoval lepších výsledků, neboť chybu jednoho by mohly ostatní opravit.

3. Implementace

3.1. Prostředí

Když jsem se dozvěděl o volnosti programovacího jazyka, okamžitě jsem se rozhodl pro C# a jeho knihovnu ML.NET. Toto prostředí je staré sotva rok a již nabývá velké popularity zejména v oboru data science a zpracování obrazu, proto jsem se s ním chtěl jako správný .NET programátor seznámit.

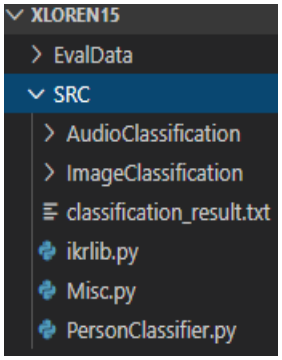
Zjistil jsem však, že zatím bohužel nemá podporu zpracování řeči, a tak jsem se nakonec rozhodl pro Python verze 3.8 z důvodu, že nyní ve fázi učení se dané problematiky nemá cenu trápit s nedostatky vývojového prostředí a je lepší se více soustředit na zadaný úkol.

3.2. Knihovny

Pro spuštění klasifikátoru je třeba mít nainstalované moduly NumPy a Sklearn pro potřeby klasifikačních výpočtů. Dále využívám modul OpenCV (cv2) pro načítání obrázků a SciPy pro načítání zvukových souborů. Vhod mi přišla i knihovna Pickle pro ukládání natrénovaných modelů. V neposlední řadě mám v projektu zahrnutou knihovnu ikrlib, která vyžaduje navíc moduly Matplotlib a Imageio, neboť scipy.misc.imread již v nových verzích není podporována a musel jsem tedy provést tuto drobnou úpravu.

3.3. Spuštění

Pokud máte na svém stroji nainstalovaný Python ideálně verze 3.8 (nevím, kolik verzí dolů bude správné spuštění zvládat) a výše popsané moduly, lze při uvedeném souborovém stromu ze složky „SRC“ program spustit následujícím příkazem:




```
python .\PersonClassifier.py ..\EvalData both
```

Jak je vidět, hlavním skriptem řešení je soubor PersonClassifier.py, který sice není moc obsáhlý, neboť pouze volá konkrétní klasifikátory AudioClassifier a ImageClassifier implementované ve vlastních stejně pojmenovaných souborech, řídí však chod programu a vyhodnotí celkový výsledek klasifikace.

Prvním parametrem skriptu je povinná cesta k datům, které má program rozpoznat. Při chybějícím parametru nebo neplatné složce končí program chybou 1.

Druhý parametr je volitelný a může nabývat hodnot „img“, „audio“ nebo „both“ (výchozí) určující, zda-li se má klasifikovat pouze obraz, zvuk či obojí. Jiná než uvedená hodnota vede opět na chybu 1.

4. Řešení

Netrvalo dlouho než jsem vystřízlivěl z nápadu udělat mnoho klasifikátorů, které by o výsledném rozhodnutí mohly hlasovat. Důvodem nebyl nedostatek snahy, nýbrž úspěchu v realizaci této ideje. Jak lze vydedukovat z popsaných knihoven, z neuronových sítí sešlo a v následujících dvou kapitolách rozeberu detailněji mé postupy a řešení jednotlivých klasifikací.

Ve výsledku jsem vytvořil jeden klasifikátor pro řeč implementující GMM a tři klasifikátory obrazu založené na lineární logistické regresi, support vector machines a random forest.

Natrénované modely ukládám pomocí pickle modulu do souborů v daných složkách a při opětovných spouštění programu již klasifikátory netrénuji, pouze načítám modely. Pro opětovné natrénování stačí soubor s modelem smazat, přejmenovat nebo přesunout do jiného adresáře. V odevzdaném adresáři jsou již modely natrénované a existují tam i prázdné složky pro testovací a trénovací data.

4.1. Zpracování řeči

Implementovat GMM klasifikátor jsem měl v plánu od začátku, nicméně chtěl jsem se vyhnout co nejvíce použití knihovny ikrlib, byť jsem se jí tedy inspiroval. Mfcc příznaky jsem extrahoval pomocí funkce mfcc() modulu python_speech_features, nicméně v porovnání s tím, co mi vracela mfcc funkce z ikrlib, jsem se nakonec rozhodl v tomto ohledu ikrlib využít, neboť python_speech_features mi vracela docela jiné hodnoty a na klasifikaci to bylo znát. To, že v průměru 10/13 příznaků byly nuly, jsem přidáním šumu vyřešil, výrazné rozdíly mezi minimálními hodnotami jsem již za problém považoval. V klasifikaci jsem pak postupoval podobně, jako ve vzorovém příkladu na rozpoznávání pohlaví, a z ikrlib jsem si vypůjčil ještě funkce train_gmm a logpdf_gmm.

Počet iterací a komponent jednotlivých tříd jsem určil experimentálně. Na začátku jsem si řekl, že vypíchnout pár shluků cílů z celkové masy bude asi vyžadovat vícero menších gausovek, zatímco necílovým vzorkům bude stačit méně větších a svou teorii jsem si potvrdil. Čím více jsem zvyšoval poměr počtu komponent pro cíle a ostatních ve prospěch cílů, tím si byl klasifikátor jistější v rozpoznání cíle. Bohužel se tím však i zvyšovala pravděpodobnost chyby, že se za cíl určí i necílový vzorek. Byl jsem nucen poměry tedy opět snižovat a po chvíli experimentování jsem s 60 iteracemi a 6 komponentami pro každou třídu dospěl k úspěchu 64/70 na testovacích datech. Stejný počet komponent znamenal, že u cílů produkoval nižší skóre, nicméně rozpoznával slušně.

Lineární klasifikátory i neuronové sítě mi tu však naprosto selhaly. Vzhledem k tomu, že o lineární klasifikaci řeči jsem toho našel pramálo, tušil jsem, že to nebude vhodná metoda a praxe mi to osvědčila. Zásadními problémy bych tu viděl nedostatek trénovacích dat a nekonzistenci délek nahrávek. Druhý problém jsem řešil nulovým „paddingem“. Tu bych asi viděl chybu a možné téma na budoucí vylepšování. Má teorie je taková, že vzhledem k tomu, že mnoho mfcc příznaků je blízké nule a nebýt přidaného šumu by nulové byly, tak ještě s vycpáním nahrávek dalšími nulami dostávám majoritně nulová data, což mi ještě v kombinaci s nedostatkem trénovacích dat způsobovalo značný overfitting. Nezbylo mi tedy než si vystačit pouze s GMM klasifikátorem.

4.2. Zpracování obrazu

Zde jsem se rozhodl aplikovat to, kde jsem u řeči selhal, tedy lineární modely knihovny sklearn a vyzkoušení si neuronové sítě. V úspěch klasifikátorů jsem tu byl optimističtější, neboť obraz se dá lineárně zpracovat lépe a snáz. Navíc jsem věděl, že můj problém s nekonzistencí velikosti dat zde již nemusím řešit, protože všechny obrázky měly stejnou velikost. Mýlil jsem se. Model logistické regrese i neuronové sítě keras knihovny mi opět selhaly a já netušil proč. U CNN jsem to opět přisuzoval nedostatku dat a skutečnost, že jsem obrázky převedl do odstínů šedi, množství dat nepomohla, nicméně většina lineárních modelů ani nezkonvergovala.

Při hledání řešení jsem narazil na poměrně obskurní, byť co se nápadu týče zajímavou, metodu převodu dat z RGB barev do HSL histogramů a klasifikovat na základě těchto nových hodnot. K mému překvapení již zde modely konvergovaly s klasifikační úspěšností až 40%. Velmi slabé, ale aspoň to už něco dělalo.

Během práce s již barevnými daty jsem si všiml, že jsem si předtím obrázky pouze špatně zpracoval a nyní již se správně načtenou, navíc i barevnou (předtím jsem pracoval pouze s šedotónovou reprezentací) sadou dat mi již sklearn modely začaly dávat dokonce až překvapivě přesné výsledky. Kromě opravy chyby ve formátu datové matice jsem i testovací a trénovací data spojil do jedné sady a nechal metodu „train_test_split“ z sklearn.model_selection náhodně rozdělit testovací a trénovací data v poměru 20:80, neboť promíchání dat v obecnosti vylepšuje trénování.

S novými úspěchy jsem se rozhodl obnovit svou ideji vícero hlasujících klasifikátorů a tak jsem nakonec použil tři modely knihovny sklearn. Zkoušet znovu neuronovou síť už se mi přiznám se vůbec nechtělo, zejména, když ony klasifikátory byly velmi přesné.

Všechny tři mnou použité modely berou stejná vstupní data a z uživatelského hlediska fungují téměř stejně, což mi velmi usnadnilo implementaci, neboť mi téměř vše stačilo dělat jen jednou.

Prvním klasifikátorem je model lineární logistické regrese LogisticRegression(). Ten umí klasifikovat několika algoritmy, já se rozhodl využit liblinear knihovnu, neboť tu v dokumentaci doporučovali pro menší počet dat, což je náš případ. Aplikuje L1 regularizaci pro penalizaci chyby.

Dalším je model SGDClassifier(), který aplikuje SVM algoritmus natrénovaný stochastickým gradientním sestupem využívající L2 regularizace pro penalizaci chyby. Jelikož hned při výchozích nastaveních model klasifikoval 100% testovacích dat správně, nechal jsem to tak.

Vzhledem k tomu, že můj systém lze zařadit mezi takzvané ensemble klasifikátory, neboť můj celkový výsledek je založen na hlasování několika menších klasifikátorů, napadlo mě zkusit použít model pracující na podobném principu. Zvolil jsem tedy random forest, který sestavuje několik rozhodovacích stromů (les) a rozhodne se na základě hlasování mezi těmito stromy o výsledku, a poněvadž se model používá stejně jako předchozí dva, implementačně mně to skoro nic nezabralo. Opět jsem dosahoval velmi dobrých výsledků, proto jsem znovu ponechal výchozí nastavení modelu.

5. Výsledky

S mými ve výsledku čtyřmi klasifikátory jsem velmi spokojen. GMM pro řeč si na trénovacích datech rozhodně nebývá tolik jistý jako ty pro obraz ani tak konzistentní, neboť po každém novém natrénování dává mírně jiné výsledky, nicméně v průměru se na testovacích datech drží úspěšnost 85-91%, což je slušné.

Jednotlivé klasifikátory obrázků měly všechny téměř stejnou přesnost v průměru 95-100% a při jejich posuzování jako celku byly bezchybné. Nikdy se nestalo, že by se spletly dva zároveň, tedy pokud jeden udělal chybu, ostatní dva ho opravily a výsledek byl vždy správný. Funkce train_test_split mi zajišťovala vždy různé kombinace testovacích a trénovacích dat a i když jsem si vzal několik vzorků bokem, které modely nikdy neviděly ani jako testovací ani jako trénovací data, tak i tyto zvládly klasifikovat bezchybně.

Jako celkový výsledek klasifikace jsem nechal klasifikátory hlasovat. Výsledné skóre je tedy v procentech a nabývá hodnot 0, pokud všechny 4 tvrdí, že se nejedná o cíl, 25 v případě, že ¼ si myslí, že to cíl je a je jedno který. Pro tyto dvě hodnoty skóre je výsledné rozhodnutí 0, jakože se nejedná o cíl. U skóre 50, tedy kdy si stejný počet myslí, že se jedná o cíl a stejný počet, že ne, se přikláním k rozhodnutí 1, neboť buď si řeč myslí, že to je on a obrázek si není jistý, proto zvolím, že ano, nebo si řeč myslí, že ne, ale obrázek si je jistý na 67%, tu jdu sice proti logice předchozí části tohoto souvětí, ale na trénovacích datech si klasifikátory obrázků jsou mnohem jistější, proto jim dávám větší váhu. Dále pak 75 a 100 jsou jasná rozhodnutí pro 1.

Uvědomuji si, že tento způsob nebere v potaz jistotu jednotlivých klasifikátorů a tedy numerická hodnota skóre není úplně přesná, nicméně GMM a SGD vyhodnocují navzájem nekompatibilní skóre a logistická regrese s random forest vrací procentuální pravděpodobnost. Těžko to lze normalizovat. Moje takové vyhodnocení potom vlastně míří za určením tvrdého rozhodnutí, které na evaluačních datech nebylo vůbec špatné, avšak měkké skóre tímto mohlo utrpět.

Výsledky jednotlivých klasifikátorů se nachází ve složkách AudioClassifier (Result_GMM.txt) a ImageClassifier (Result_LogisticRegression.txt, Result_SGD.txt, Result_RandomForest.txt). Soubor s celkovým řešením „classification_result.txt“ s nachází ve složce SRC. Reprezentanty jednotlivých klasifikací, tedy GMM za audio, hlasování oněch tří image klasifikátorů (Result_Overall.txt) za image a celkový výsledek klasifikace jsem ještě překopíroval do složky Results v kořenu odevzdaného adresáře pod patřičnými názvy.

6. Vylepšení

Těch lze najít mnoho, některé jsem již zmínil. V první řadě bych zlepšil normalizaci skóre jednotlivých klasifikátorů, abych mohl dát přesnější procentuální výsledné skóre. Dále by stálo za to si s již dobře načtenými obrázky zkusit naimplementovat nějakou neuronovou síť. Najít způsob lepšího zpracování řečových dat pro potřeby neuronových sítích. Vyplatilo by se i sehnat více trénovacích dat nebo vyzkoušet techniku data augmentation, například použitím ImageDataGeneratoru knihovny Keras.

7. Zhodnocení

Jak jsem již popsal, cesta k výsledkům byla dlouhá, náročná a plná neúspěchů. Zdaleka více klasifikátorů nefungovalo než fungovalo, nicméně ve výsledku jsem tomu rád. Kdyby vše fungovalo jako po másle, nenaučil bych se toho tolik. Díky tomu jsem si prošel velké množství modelů, algoritmů, pythonovských knihoven a celkově získal větší vhled do problematiky.