Projekt do předmětu Strojové učení a rozpoznávání (SUR)

Matej Sirovatka (xsirov00)
Veronika Nevařilová (xnevar00)

5. května 2025

1 Klasifikace osob pomocí audia

Klasifikátor audia se nachází ve složce `SRC/audio`. Je možné ho spouštět pro trénování s následnou evaluací modelu, nebo také přímo pro klasifikaci audio nahrávek z již existujícího uloženého modelu.

1.1 Spuštění

Klasifikátor se spouští příkazem:
`python3 classify.py [-train <data_dir>]/[--load] [--save] [-inference <inference_dir>]`

Význam parametrů je následující:

- `-train <data_dir>` – určuje, že se má natrénovat nový model z dat v dané složce. Daná složka musí obsahovat adresáře `train` a `dev`, ve které se nachází podadresáře s `wav` soubory pro každou osobu.
- `--load` – určuje, že se má načíst model ze souboru uloženého z dřívějšího trénování. Skript ukládá model vždy do souboru `speaker_classifier.pkl`, není proto třeba předávat parametr se souborem. **Při každém spuštění skriptu musí být specifikován model předáním buď parametru `--load` nebo `-train`.**
- `--save` – uloží načtený model do souboru `speaker_classifier.pkl`.
- `-inference <inference_dir>` – na načteném modelu provede inferenci s `.wav` soubory v adresáři `inference_dir` a výstup uloží do souboru `inference_results.txt` Pokud je model specifikován volbou `-train`, inference se provede po natrénování modelu..

Pro příklad, níže jsou příkazy, pomocí kterých byla vytvořena trénovací data spojením `.wav` souborů ze složek `train` a `dev` původních dat, a následně byl natrénován a vyhodnocen model na dodaných ostrých datech v adresáři `eval`:

```
python3 merge_dev_to_train.py ./SUR_projekt2024-2025
```

```
python3 classify.py -train ./all_train -inference ./eval
```

Pozn.: Cílem spojení souborů z `dev` do `train` bylo natrénovat model na všech dostupných datech za účelem co nejvyšší přesnosti na ostrých datech. Každopádně je samozřejmě možné předat místo spojené složky originální adresář `SUR_projekt2024-2025`.

1.2 Princip trénování klasifikátoru

Trénování klasifikátoru `GMMSpeakerClassifier` začíná převedením všech trénovacích dat pro každou osobu na MFCC (Mel-Frequency Cepstral Coefficients). Koeficientů se většinou používá 13, proto tento počet byl zvolen ze začátku i zde. Byly vyzkoušeny i jiné počty koeficientů, avšak vždy se přesnost klasifikátoru buď zhoršila, nebo zůstala stejná. Převod na MFCC provádí metoda `extract_mfcc_sequence`. Příznaky jsou v ní také normalizovány pomocí tzv. *z-score*, kdy dochází k jejich transormaci na nulový průměr a jednotkovou odchylku. Takto upravené příznaky jsou pak vstupem pro modelování GMM (směsi Gaussovských rozložení) pro každou třídu.

Modelování GMM je prováděno pomocí třídy `GaussianMixture` z knihovny `sklearn`. Výsledkem trénování je slovník obsahující jednu GMM pro každou třídu klasifikace.

Následná predikce probíhá tak, že se každý vzorek, který má být klasifikován, převede stejným způsobem jako trénovací data na příznaky. Následně se prvek ohodnotí vůči GMM každé třídy získáním logaritmické věrohodnosti. Jako výsledná třída je vybrána ta, u které má prvek tuto hodnotu nejvyšší.

Tento samotný postup měl na začátku přesnost zhruba 45 % na `dev` datech. Dalším úkolem proto bylo vyzkoušet jiné zpracování nebo implementovat dodatečné věci, které by mohly klasifikaci pomoci.

1.3 Ladění parametrů a trénování

Jako první byla vyzkoušena redukce dimenzí pomocí PCA (Principal Component Analysis) i LDA (Linear Discriminant Analysis). Bohužel, žádnou redukcí dimenzí se nepodařilo zvýšit přesnost klasifikátoru, a proto ve výsledném klasifikátoru PCA ani LDA ponechány nebyly. Důvodem, proč LDA

a PCA ke zvýšení přesnosti nepřispěly však může být to, že při získání příznaků z dat jsou zároveň příznaky již normalizovány.

Věc, která klasifikátoru naopak lehce pomohla, bylo přidání dalších příznaků k MFCC, a to první a druhé derivace těchto příznaků. Ty pomáhají lépe zachycovat změny MFCC v čase a jejich zrychlení či zpomalení.

Zvýšení počtu komponent GMM také přispělo k přesnosti modelu. Původně bylo nastaveno komponent 16, maxima přesnosti však při daném nastavení klasifikátor dosáhl při počtu 45 komponent. Takový počet komponent je však poměrně vysoký a trénování se s každým zvětšením počtu komponent zpomalovalo.

Hlavním nastavením, které významně přispělo ke zlepšení klasifikace, byla změna typu kovarianční matice při modelování GMM. Klasifikátor byl původně nastaven na diagonální matici. S ní dosahoval přesnosti zhruba 67 %. Po nastavení plné kovarianční matice se přesnost zvýšila na 73 %, a s nastavením `tied`, které znamená, že všechny komponenty mají stejnou kovarianční matici a liší se pouze ve středních hodnotách, klasifikátor dosáhl přesnosti 90,32 % na testovacích datech. Zároveň s danou kovarianční maticí není potřeba pro tuto přesnost až 45 komponent, ale stačí 35.

Během ladění parametrů a trénování bylo vyzkoušeno více úprav, např. přidání dalších příznaků k MFCC a derivacím, různý počet iterací při modelování GMM apod. Největších zlepšení přesnosti však bylo dosáhnuto nastavením popsaným výše. Celkově se tak podařilo zlepšit přesnost klasifikace více než 2x.

1.4 Možnosti zlepšení

Mezi další věci, které by mohly přispět ke zlepšení klasifikátoru, by mohlo patřit:

- **Přidání dalších příznaků k MFCC a derivacím**, které by mohly pomoci najít v datech další skryté vzory.
- **Augmentace dat** pro lepší generalizaci a lepší klasifikaci nahrávek dané osoby z např. jiných sezení.
- **Systematická a úplná optimalizace parametrů**, např. algoritmem *grid search*, kdy by byla nalezena ideální kombinace na rozdíl od pouhého *zkoušení* různého nastavení. Je možné, že by se tímto také mohl podařit najít např. ideální počet dimenzí pro LDA, který by vedl ke zvýšení přesnosti klasifikace.

2 Klasifikace osob pomocí obrazu

2.1 Popis řešení

Klasifikátor obrazu je implementován pomocí konvoluční neuronové sítě (CNN) a nachází se v souboru `SRC/image/cnn.py`. Tento skript umožňuje dvě hlavní operace: trénování modelu na poskytnutých datech a následnou predikci identity osob na nových obrázcích pomocí již natrénovaného modelu.

2.2 Spuštění

Skript `src/image/cnn.py` lze spravovat a spouštět pomocí virtuálního prostředí a správce balíčků `uv`, popis prosředí se nachází v souboru `src/image/uv.lock`, nebo také nainstalovaním balíčku přes `pip`. Popis pro pip se nachází v souboru `src/image/requirements.txt`.

Na natrénovaní modelu lze použít příkaz:

```
python3 src/image/cnn.py train --config src/image/configs/best.yaml \
--train_dir data/train_fixed \
--eval_dir data/dev_fixed \
--epochs 20 \
--lr 3e-4 \
--output_dir src/image/outputs \
--batch_size 4 \
--run_name baseline
```

Pro spuštění predikce pomocí existujícího, natrénovaného modelu (např. `src/image/outputs/baseline.pt`) na nových datech (např. v adresáři `data/evaluation/eval`) s využitím specifické konfigurace (uložené v `src/image/configs/best.yaml`), použijte následující příkaz:

```
python3 src/image/cnn.py predict --config src/image/configs/best.yaml \
--model src/image/outputs/baseline.pt \
--data_dir data/evaluation/eval \
--output out.csv
```

Tento příkaz načte model a konfiguraci, zpracuje obrázky v zadaném adresáři `data/evaluation/eval` a výsledné predikce uloží do souboru `out.csv` ve formátu CSV.

2.3 Architektura

Jako klasifikátor byla využita poměrně standardní architektura konvoluční sítě (CNN), která se skládá z konvolučních vrstev a maxpooling vrstev, spolu s Batch Normalization vrstvami. Jako nelinearita je použita aktivace Tanh, která dosahovala nejlepších výsledků. Tyto vrstvy postupně zvyšovaly počet kanálů a snižovaly šířku a výšku obrázku. Za těmito vrstvami se nachází plně propojené vrstvy s nonlinearitami mezi nima. Také bylo experimentováno s Dropout vrstvami, ale výsledky byly horší. Počet výstupů z plně propojených vrstev se postupně zvyšoval s hloubkou sítě.

Přesná konfigurace modelu je uložená v souboru `src/image/configs/best.yaml`.

Na trénovaní modelu byl použit optimalizátor Adam, spolu s chybovou funkcí Cross Entropy Loss. Před vstupem do této funkcie byly výstupy z plně propojených vrstev normalizovány pomocí softmax operace.

2.4 Ladění hyperparametrů

Pro dosažení optimálního výkonu modelu bylo provedeno experimentální ladění hyperparametrů. Cílem bylo nalézt kombinaci parametrů, která maximalizuje přesnost klasifikace na validačních datech.

Mezi hlavní laděné hyperparametry patřily:

- Rychlost učení (learning rate): Testovány různé hodnoty, např. 0.001, 0.0005, 0.0001, nakonec byla zvolena hodnota 3e-4.
- Velikost dávky (batch size): Experimentováno s velikostmi jako 16, 32, 64, nakonec byla zvolena hodnota 4, kde model dosahoval nejlepšího výkonu.

- Optimalizátor: Zkoušeny různé optimalizátory, např. Adam nebo SGD s momentem, nakonec byl zvolen Adam.
- Počet trénovacích epoch: Model byl trénován na různý počet epoch, sledovala se konvergence a riziko přeučení, také byl sledován tzv. *model collapse*.
- Architektura sítě: Byly zvažovány drobné úpravy v počtu a velikosti konvolučních a plně propojených vrstev (i když základní architektura byla zachována). Tohle bylo umožněno možností načíst a upravit konfiguraci před trénováním pomocí parametru `--config`.

Jako metoda pro hledání optimálních parametrů byl použit převážně manuální proces (iterativní zkoušení a vyhodnocování) v kombinaci s principy podobnými Grid Search pro některé parametry. Výkon byl hodnocen primárně pomocí metriky přesnosti (accuracy) na vyhrazené validační sadě dat. Na základě těchto experimentů byla zvolena konfigurace uložená v `src/image/configs/best.yaml`, která poskytovala nejlepší výsledky na validačních datech.

Tenhle model nedosahoval příliš dobrých výsledků (40% accuracy), a proto bylo zvoleno obohacení trénovací sady dat pomocí silné augumentace. Tohle následně zvýšilo přesnost na něco kolem 50% accuracy.

Posledním vylepšením modelu bylo rozšíření datové sady pomocí půlky evaluation dat, která byla tím pádem zmenšena. Tohle zvýšilo přesnost na něco kolem 70% accuracy, což je také finální výsledek. Skript `src/image/get_bigger_train.sh` umožňuje vytvoření této trénovací a evaluační sady dat.

2.5 Možnosti zlepšení

Výběr neuronové sítě jako klasifikátoru nebyla nejlepší volba, kvůli velmi nízkému množství dat. I tak se povedlo natrénovat poměrně funkční klasifikátor.

Zlepšit by se dalo mnoho ohledně architektury sítě, ku příkladu použitím prohledávacího algoritmu Grid Search pro nalezení optimálních parametrů. Největšího zlepšení by dosáhlo zvětšení množství dat, což však nemusí být možné. V případě malého množství dat by bylo ideální zvážit jiný druh klasifikátoru, ku příkladu Support Vector Machine nebo Random Forest, tyto klasifikátory dokážou dosáhnout lepších výsledků na malém množství dat.