1 GMM na detekovanie podľa hlasovej nahrávky

Na spracovanie zvuku sme zvolili metódu GMM. Pri implementácii sme používali jazyk Python, konkrétne verziu 3.8.

1.1 Spustenie

Požadované knižnice:

- `getopt`
- `numpy`
- `sklearn.mixture`

Knižnice boli nainštalované nástrojom `pip`. Na správne fungovanie musí byť použitý Python3. Na spustenie skriptu, ktorý zároveň klasifikátor aj natrénuje, je potrebné spustiť súbor `gmm.py` s argumentami:

- `tdir` – cesta k adresáru s trénovacími dátami obsahujúcimi target
- `ntdir` – cesta k adresáru s trénovacími dátami neobsahujúcimi target
- `testdir` – cesta k adresáru s dátami, ktoré chceme vyhodnotiť

Príklad spustenia

```
python3.8 gmm.py --tdir=target_train/ --ntdir=non_target_train/ --testdir=eval/
```

1.2 Implementácia

Na extrahovanie MFFC príznakov sme použili funkciu z importovanej `ikrlib`, v ktorej sme spravili len mierne úpravy, aby fungovala na novšej verzii Pythonu.

Klasifikátor GMM sa skladá z dvoch súborov, jeden na trénovanie – `train_gmm.py` a druhý na spustenie samotného klasifikátoru `gmm.py`. Ten importuje `train_gmm` a gaussovky natrénuje. Adresáre, v ktorých sa nachádzajú dáta na natrénovanie a testovanie sú predávané ako parametre skriptu.

Na natrénovanie a vytvorenie samotných gaussoviek sme použili triedu `GaussianMixture` zo `sklearn.mixture`. Konkrétne metódu `fit` na natrénovanie a neskôr metódu `score` na vyhodnotenie. Skript volá funkcie z `train_gmm.py`, konkrétne:

- `target_gmm` – extrahuje priznaky hľadanej osoby z predaného adresára použitím funkcie `wav16khz2mfcc` z `ikrlib`. Vytvorí a natrénuje gaussovku a vráti ju.
- `non_target_gmm` – vytvorí a natrénuje gaussovky použitím dát v zadanom adresári. Zároveň prijíma ako argument počet súborov prislúchajúcich jednej osobe. Pre každú osobu vytvorí jednu gaussovku a všetky vráti v jednom zozname.

Skript ďalej načítava príznaky z testovacieho adresára a vyhodnocuje ich logaritmickú pravdepodobnosť použitím metódy `score` na gaussovky, ktoré vrátili funkcie v trénovacej fáze.

Výsledok vyhodnotí podľa najvyššej pravedpodobnosti. Ak najvyššia logaritmická pravdepodobnosť(skóre) prislúcha gaussovke targetu, je to target, inak to nie je target. Skóre do výsledného súboru s výsledkami bolo vypočítané ako rozdiel skóre targetu a najvyššieho skóre z ostatných osôb(non-targets).

1.3 Testovanie

Výsledky sme priebežne testovali na dátach v adresári `target_dev` a `non_target_dev`, ako trénovacie sme zvolili tie v ostatných adresároch. Pri vyhodnocovaní sme trénovali na dátach vo všetkých štyroch adresároch.

Postupne sme skúšali viac spôsobov implementácie a priebežne ich testovali. Skúšali sme vytvoriť len dve gaussovky – jednu pre target a druhú pre ostatné nahrávky, ale osvedčila sa metóda jednej gaussovky na jednu osobu a následné porovnanie podľa najvyššieho skóre.

Taktiež sme spozorovali vyššiu presnosť klasifikátora pri použití funkcie na extrakciu mffc z `ikrlib` namiesto funkcie z `python_speech_features`.

1.4 Možné vylepšenia

Úpravami, ktoré by mohli zlepšiť výsledky by mohli byť prípadne iné parametre vytvorených gaussoviek alebo použitie aj iných vlastností audia ako sú mffc.

2 Konvolučná neurónová sieť (CNN) – rozpoznávanie .png obrázkov

Pri klasifikácii obrázkov sme sa rozhodli zvoliť metódu, v ktorej sme využili konvolučnú neurónovú sieť (CNN). CNN sme implementovali v jazyku Python vo verzii 3.7 – nakoľko v novšej verzii 3.8 nie sú dostupné knižnice, ktoré sme využili. Konkrétne sme využívali `TensorFlow` využívajúci `Keras API` a knižnicu `numpy`.

2.1 Spustenie

Potrebné knižnice: `numpy, TensorFlow - Keras`

Nasledujúci postup je opísaný tak, aby bolo možné dosiahnuť čo najpodobnejšie výsledky ako v našom odovzdanom riešení – napr. voľba trénovacích dát, či počet augmentovaných súborov. Dáta nebudú identické, pretože augmentácia prebieha náhodne a taktiež NN sa nemusí natrénovať identicky.

1. Preorganizovanie trénovacích dát – všetky vopred poskytnuté dáta (vrátane testovacích) vložiť do jedného adresára s dvoma podadresármi – `target` a `no_target` (**názvy sú povinné**)

2. Odstránenie .wav súborov pomocou skriptu: `remove_wavs.py [DIR]`

   ```
   python3.7 remove_wavs.py ./cesta_k_trenovacim_datam/target
   python3.7 remove_wavs.py ./cesta_k_trenovacim_datam/no_target
   ```

3. Augmentácia trénovacích dát – `data_augm.py [DIR] [N]`

   ```
   python3.7 data_augm.py ./cesta_k_trenovacim_datam/target/         354
   python3.7 data_augm.py ./cesta_k_trenovacim_datam/no_target/      192
   ```

4. Načítanie trénovacích dát a ich preformátovanie vhodné k trénovaniu – `load_data.py [DIR]`

   ```
   python3.7 load_data.py ./cesta_k_trenovacim_datam/
   ```

   (pozor, zadávame cestu len do adresára, kde sa nachádzajú podadresáre target a no_target)

5. Trénovanie modelu – `python3.7 train_model.py`

6. Predikcia na dátach s využitím modelu a generovanie výsledkov

   ```
   predict.py [DIR_DATA] [MODEL] [OUTPUT]
   predict.py  ./cesta_k_EVAL_datam model.h5 results.txt
   ```

   (je možné využiť už skôr natrénovaný model – naše modely nájdete v adresári `/trained_models/`)

2.2 Implementácia

Implementácia klasifikácie pozostávala z niekoľkých častí – od prípravy dát, cez samotnú CNN až po konečnú predikciu – vyhodnotenie.

Príprava dát sa skladala z 3 častí, ktoré boli implementované ako 3 nezávislé programy:

1. `remove_wavs.py` – odstránenie .wav súborov z adresárov, kde sa nachádzali trénovacie dáta

2. `data_augm.py` – augmentácia obrázkov

3. `load_data.py` – načítanie obrázkov a ich formátovanie

Augmentácia obrázkov je implementovaná tak, že je možné si zvoliť, koľko obrázkov má byť vytvorených a cyklus prechádza súbor po súbore v danom adresári a aplikuje náhodnú transformáciu – pridá šum, zrotuje obrázok alebo ho preklopí. Množstvo šumu či o koľko stupňov bude obrázok otočený je náhodné.

Nakoniec prebieha načítanie obrázkov, kedy prichádza k prevodu do grayscale, úpraví jednotne rozmery obrázkov a konvertuje ich do formátu `numpy array`. Podľa zložky, v ktorej sa nachádzajú, program dokáže uložiť numpy array aj s ich labelami, ktoré sú neskôr využité pri trénovaní. Výstupom tohto programu sú súbory vo formáte `.pickle`, ktoré sú neskôr načítavané.

Trénovanie CNN prebieha taktiež v samostatnom programe `train_model.py`. Program si načíta dáta a labely zo súborov `.pickle` a pomocou funkcií z `API Keras` je vytvorená CNN - má 2 vrstvy, bez Dropout, je použitá aktivačná funkcia `softmax` a objektívna funkcia `binary-crossentropy`. Trénovanie je implementované na 10 epoch, pričom 20% dát je oddelených a stávajú sa validačnými.

Pri testovaní natrénovaného modelu sa preukázalo, že ak sme zvolili menej epoch, tak sa model mal tendenciu mýliť častejšie. Problém však bol, že sme pravdepodobne model vždy pretrénovali, hoci nám dával pomerne dobré výsledky. Pretrénovanie sa prejavovalo tým, že CNN si svojími rozhodnutiami bola istá na 100%, hoci sa niekedy pomýlila. Toto sme sa pokúšali riešiť Dropoutami a epochami, prípadne nastaviť menší/väčší batch_size. Táto chyba sa nám však nepodarila odstrániť, preto sme ostali na vyššie uvedených nastaveniach.

Testovanie modelu – na základe testovacích dát je možné v `test_model.py` vyhodnotiť úspešnosť pomocou `confusion matrix`.

Trénovanie modelu sme uskutočňovali najprv na pôvodnom zadanom rozdelení, no potom sme zistili, že ak si rozdelíme dáta tak, že zväčšíme počet trénovacích dát, model je presnejší. Týmto sme však mohli testovať na menšej vzorke, čo nebolo úplne ideálne. Finálny model, ktorý rozhodoval o evaluačných dátach sme natrénovali na všetkých daných olabelovaných dátach.

Predikcia na evaluačných dátach prebieha v `predict.py` Najprv sa načítajú evaluačné dáta, sformátujú sa podobne ako pri `load_data.py` a následne prebieha predikcia na danom modele, ktorého výsledky sa zapisujú do súboru zadaného formátu.

Rozdelenie implementácie na menšie samostatné programy nám umožnilo efektívnejšie pracovať s jednotlivými časťami, ktoré sú nutné pre dosiahnutie zadania.

2.3 Možné vylepšenia

Vhodným vylepšením by bolo doplniť trénovacie dáta, nakoľko sme mali zadanú pomerne malú dátovú vzorku. Ideálnym riešením by bolo použiť nejakú predtrénovanú neurónovú sieť. Druhou možnosťou by mohla byť zmena architektúry – pridanie/odobranie konvolučnej vrstvy.