Systém na rozpoznanie osoby podľa fotografií a nahrávok

Tento SW je implementovaný ako projekt do kurzu SUR/2019L na FIT VUT

Autori: Tibor Kubík a Andrej Ježík

Ako riešenie tohto projektu sme postavili nasledujúce systémy: - Systém na detekciu **rečníka** pomocou *Gaussian Mixture Model* (**GMM**)

- Systém na detekciu osoby z fotografie pomocou *Multi-layer Perceptreon Classifier* (MLPClassifier)

Systém na detekciu rečníka pomocou GMM

Tento systém je implementovaný v súbore `voice_GMM.py`. Pri jeho implementácii sme použili funkcie z knižnice `ikrlib.py` a to konkrétne funkciu na extrahovanie trénovacích a evaluačných `.wav` súborov a funkcie na klasifikáciu pomocou **GMM**.

Prerekvizity

Je nutné mať stiahnutý Python. Projekt je implementovaný a spustiteľný na verzii Python 2.7.17. Použitie tejto staršej verzie je hlavne z dôvodu použitia funkcii knižnice ikrlib.py, kde nám robilo problémy importovanie niekoľkých knižníc vo vyšších verziach. Neboli sme si istí licenciou na túto knižnicu, tak sme sa rozhodli nič nekopírovať/upravovať a použiť mierne zastaralú verziu Pythonu. Nutné knižnice: `matplotlib`, `numpy`, `glob` a `scipy` kompatibilné s verziou Pythonu, ktorou budete projekt spúšťať. Ďalej je nutné dodržať hierarchiu súborov. Mala by vyzerať nasledovne:

```
.
|
└── src
|      | voice_GMM.py
|      | face_PCA_MLPerceptron.py
|      | ikrlib.py
|
└── data
       |
       └── train
       |       |
       |       └──  non-target
       |       |       | f401_03_p01_i0_0.png
       |       |       | f401_03_p01_i0_0.wav
       |       |       | .
       |       |       | .
       |       |
       |       └── target
       |               | m429_03_p01_i0_0.png
       |               | m429_03_p01_i0_0.wav
       |               | .
       |               | .
       |
       └── eval
              | personToEval01.png
              | personToEval02.wav
              | .
              | .
```

Pričom podsúbory v **train** definujú triedy. **Je nutné, aby dáta osoby, o ktorej chceme zistiť, či je to práve tá osoba - teda target, boli posledným priečinkom v priečinku train!** Z toho vyplýva, že je veľmi podstatné dodržanie aj názvov jednotlivých priečinkov.

Spustenie

Aplikácia sa spúšťa príkazom `python2.7 voice_GMM.py` (prípadne `python voice_GMM.py`), pričom prepokladáme, že sa nachádzame v súbore `src` . Nie je nutno dodávať žiadne argumenty. Je možné meniť cesty k trénovacím a evaluačným dátam. Konkrétne na riadkoch **18 - 21**. Je nutné špecifikovať počet tried, do ktorých budú dáta klasifikované.

```
 TRAIN_DIR = "../data/train"
EVAL_DIR = "../data/eval"
N_OF_CLASSES = 2            # edit this according to the number of training classes.
```

Výsledok klasifikácie

Výsledok klasifikácie sa nachádza v súbore **audio_GMM.txt**. Ten bude obsahovať pre každý .wav súbor, ktorý sa nachádza v priečinku data/eval jeden riadok, ktorý hovorí o výsledku klasifikácie. Tento riadok obsahuje meno klasifikovaného súboru, hodnotu log-likelihoodu daného súboru voči triede target a tvrdé rozhodnutie o tom, či daný súbor patrí do triedy target alebo nie(viď zadanie projektu).

Čo sme si odniesli pri implementácii tohto systému + naše postrehy

Vyskúšali sme si prakticky veci z prednášok, a to hlavne ako funguje GMM. Pri implementácii sme skúšali, aké su vplyvy rôznch

faktorov na kvalitu klasifikátoru, napríklad počet iterácii alebo počet gaussoviek pre jednotlivé triedy. Kvalitu systému sme skúšali tak, že sme ho vyhodnocovali ako na videných dátach, tak aj na nevidených - z pôvodne trénovacích dát sme niekoľko presunuli do evaluačných. Pre obe možnosti sme nastavovali rôzne hodnoty vyššie spomínaných faktorov. Klasifikátor sme taktiež skúšali tak, že sme preorganizovali trénovacie dáta. Pre každú osobu sme si vytvorili zvláštnu triedu. Klasifikátor tak dosahoval menšej hodnoty False Alarmu, no nakoniec sme sa rozhodli trénovať predsa len na dvoch triedach - target a non-target.

Systém na detekciu osoby z fotografie pomocou PCA a MLPClassifier

Tento systém je implementovaný v súbore `face_PCA_MLPerceptron.py`. Pri jeho implementácii sme použili funkciu `png2fea` z knižnice `ikrlib.py` na extrakciu fotografii. Ďalej sme použili rozsiahlu ML knižnicu `sklearn`, z ktorej sme si importovali **PCA** a **MLPClassifier**.

Prerekvizity

Je nutné dodržať rovnaké prerekvizity ako v predošlom systéme + je nutné mať nainštalovanú knižnicu `sklearn`. Je nutné dodržať rovnakú štruktúru a aj skutočnosť, že trieda target musí byť poslednou triedou vrámci priečinku train(opäť je teda kľúčové dodržanie názvov priečinkov).

Spustenie

Aplikácia sa spúšťa príkazom `python face_PCA_MLPerceptron.py`, pričom prepokladáme, že sa nachádzame v súbore `src`. Nie je nutno dodávať žiadne argumenty. Je možné meniť cesty k trénovacím a evaluačným dátam. Konkrétne na riadkoch **26 - 28**. Je nutné špecifikovať počet tried, do ktorých budú dáta klasifikované.

```
TRAIN_DIR = "../data/train"
EVAL_DIR = "../data/eval"
N_OF_CLASSES = 2
```

Výsledok klasifikácie

Výsledok klasifikácie sa nachádza v súbore **image_PCA_MLPerceptron.txt**. Ten bude obsahovať pre každý .png súbor, ktorý sa nachádza v priečinku data/eval jeden riadok, ktorý hovorí o výsledku klasifikácie. Tento riadok obsahuje meno klasifikovaného súboru, hodnotu pravdeodobnosti(0-1), či daný súbor patrí do triedy target a tvrdé rozhodnutie o tom, či daný súbor patrí do triedy target alebo nie(viď zadanie projektu). Prah rozhodovania je nastavený na 0.6, teda ak si je klasifikátor istý s pravdepodobnosťou aspoň 60% usúdime tvrdé rozhodnutie 1, inak 0.

Čo sme si odniesli pri implementácii tohto systému + naše postrehy

Pri tomto systéme sme si vyskúšali **Analýzu hlavných komponentov**(PCA) spolu s chovaním **Multi-layer perceptronu** v praxi. Pre PCA je kľúčové vybrať správny počet komponent, na čo sme použili pomocnú funkciu, ktorá nam vykreslí graf, ktorý nám ukázal, aký počet komponent by mal byť dostačujúci. Rovnako sme si poskúšali rôzne nastavenia, ktoré ovplyvňovali kvalitu extrahovaných dát. Pri nastavovaní samotného klasifikátoru bolo možností ešte viac. Skúšali sme kombinácie rôznych aktivačných funkcii, rôzne solvre pre optimalizáciu váh, rôzne počty skrytých vrstiev a podobné možnosti, ktoré ponúka MLP klasifikátor z knižnice sklearn. Konečné nastavenie nám po mnohých pokusoch (+ sme sa riadili aj odporučanými nastaveniami z dokumentácie a taktiež sme si nechali poradiť niektorými doporučovanými nastaveniami komunity) dáva pomerne kvalitné výsledky na videných aj nevidených dátach.

Ostatné pokusy

Okrem týchto dvoch systémov sme skúšali postaviť aj ďalšie systémy. Tie však zostanú neodovzdané, pretože si zaslúžia oveľa viac priestoru na ich vypracovanie a teda dosahovanie kvalitných výsledkov. Za zmienku možno stojí rozpracovaný klasifikátor využívajúci extrakciu pomocou **Histogram of oriented gradients**(HOG) či veľmi populárnej knižnice **Facenet** a stavba klasifikátora technológiami **tensor** a pomocu Metódy podporných vektorov - **Support Vector Machines**(SVM).