Proiect: Identificator de cuvinte cheie (Keyword Spotting)

Acest proiect folosește un subset din setul de date Google Speech Commands și implementează o metodă de recunoaștere a unor cuvinte (ex. „yes”, „no”, „up”, „wow”) într-un flux audio live. Codul a fost organizat în mai multe fișiere:

1. **Augment_data.py**
2. **Preprocess_data.py**
3. **Live_detect.py**
4. **Train_svm.py** sau **train_rf.py** (în funcție de modelul folosit, train_rf.py dureaza foarte mult)

Modelul final este antrenat pe date augmentate (inclusiv cu zgomot), pentru a fi mai robust la condițiile din mediu real. În plus, există și o logică simplă de noise detection (identificare a segmentelor de zgomot / lipsă de vorbire).

Thought about word detection system for a second

Mai jos este un exemplu de README/documentație în care se descrie modul de funcționare al aplicației de detecție a cuvintelor (Keyword Spotting) și etapele importante din cod. Poți adapta și organiza informațiile în funcție de nevoi înainte de a genera un fișier PDF.

Proiect: Identificator de cuvinte cheie (Keyword Spotting)

Acest proiect folosește un subset din setul de date Google Speech Commands și implementează o metodă de recunoaștere a unor cuvinte (ex. „yes”, „no”, „up”, „wow”) într-un flux audio live. Codul a fost organizat în mai multe fișiere:

1. **Augment_data.py**
2. **Preprocess_data.py** (în text apare ca Preprocess_dsata.py – probabil un typo)
3. **Live_detect.py**

4. (Opțional) **Train_svm.py** sau **train_rf.py** (în funcție de modelul folosit)

Modelul final este antrenat pe date augmentate (inclusiv cu zgomot), pentru a fi mai robust la condițiile din mediu real. În plus, există și o logică simplă de noise detection (identificare a segmentelor de zgomot / lipsă de vorbire).

---

1. Augment_data.py

Scop

- Să crească volumul de date de antrenament prin diverse tehnici de augmentare (time shift, pitch shift, speed change, injectare de zgomot).
- Asta ajută la crearea unui model mai robust la variații de ton, viteză, și zgomot de fond.

Pași cheie

- **Încarcă fișierele .wav** dintr-un folder (subset) ce conține subfoldere cu diverse cuvinte (ex. „yes”, „no”, „up”, „down”, „wow”, plus _background_noise_ pentru zgomot).
- **Aplică augmentări**:
    - **Time shifting**: deplasează semnalul audio cu un procent din lungimea sa, circular.
    - **Pitch shifting**: schimbă tonalitatea cu ±1 și ±2 semitonuri.
    - **Speed change**: face semnalul mai lent sau mai rapid cu ±10%.
    - **Injectare zgomot**: amestecă semnalul cu fișiere din _background_noise_.
- **Salvează noile fișiere** într-un folder (subset_augmented).

Astfel, fiecare fișier inițial produce mai multe fișiere augmentate, crescând diversitatea dataset-ului.

2. Preprocess_data.py (în text, Preprocess_dsata.py)

Scop

- Să extragă caracteristicile (MFCC – Mel-Frequency Cepstral Coefficients) din fișierele audio și să le salveze într-un format numeric (NumPy .npy), alături de etichete.

Pași cheie

- **Parcurge fișierele .wav** din subset_augmented.
- **Extrage MFCC** (40 de coeficienți MFCC) pentru fiecare fișier, calculează media și abaterea standard (std) pe axa temporală, apoi concatenează rezultatele (producând un vector de 80 de elemente: 40 mean + 40 std).
- **Salvează** totul în două fișiere: features_data.npy (cu MFCC-urile) și labels_data.npy (cu etichetele).

Aceste fișiere .npy vor fi apoi încărcate de scripturile de antrenare (Train_svm.py, Train_rf.py etc.).

Thought about word detection system for a second

Mai jos este un exemplu de README/documentație în care se descrie modul de funcționare al aplicației de detecție a cuvintelor (Keyword Spotting) și etapele importante din cod. Poți adapta și organiza informațiile în funcție de nevoi înainte de a genera un fișier PDF.

Proiect: Identificator de cuvinte cheie (Keyword Spotting)

Acest proiect folosește un subset din setul de date Google Speech Commands și implementează o metodă de recunoaștere a unor cuvinte (ex. „yes”, „no”, „up”, „wow”) într-un flux audio live. Codul a fost organizat în mai multe fișiere:

1. **Augment_data.py**
2. **Preprocess_data.py** (în text apare ca Preprocess_dsata.py – probabil un typo)
3. **Live_detect.py**
4. (Opțional) **Train_svm.py** sau **train_rf.py** (în funcție de modelul folosit)

Modelul final este antrenat pe date augmentate (inclusiv cu zgomot), pentru a fi mai robust la condițiile din mediu real. În plus, există și o logică simplă de noise detection (identificare a segmentelor de zgomot / lipsă de vorbire).

---

1. Augment_data.py

Scop

- Să crească volumul de date de antrenament prin diverse tehnici de augmentare (time shift, pitch shift, speed change, injectare de zgomot).
- Asta ajută la crearea unui model mai robust la variații de ton, viteză, și zgomot de fond.

Pași cheie

- **Încarcă fișierele .wav** dintr-un folder (subset) ce conține subfoldere cu diverse cuvinte (ex. „yes”, „no”, „up”, „down”, „wow”, plus _background_noise_ pentru zgomot).
- **Aplică augmentări**:
    - **Time shifting**: deplasează semnalul audio cu un procent din lungimea sa, circular.
    - **Pitch shifting**: schimbă tonalitatea cu ±1 și ±2 semitonuri.
    - **Speed change**: face semnalul mai lent sau mai rapid cu ±10%.
    - **Injectare zgomot**: amestecă semnalul cu fișiere din _background_noise_.
- **Salvează noile fișiere** într-un folder (subset_augmented).

Astfel, fiecare fișier inițial produce mai multe fișiere augmentate, crescând diversitatea dataset-ului.

---

2. Preprocess_data.py (în text, Preprocess_dsata.py)

Scop

- Să extragă caracteristicile (MFCC – Mel-Frequency Cepstral Coefficients) din fișierele audio și să le salveze într-un format numeric (NumPy .npy), alături de etichete.

Pași cheie

- **Parcurge fișierele .wav** din subset_augmented.
- **Extrage MFCC** (40 de coeficienți MFCC) pentru fiecare fișier, calculează media și abaterea standard (std) pe axa temporală, apoi concatenează rezultatele (producând un vector de 80 de elemente: 40 mean + 40 std).

- **Salvează** totul în două fișiere: features_data.npy (cu MFCC-urile) și labels_data.npy (cu etichetele).

Aceste fișiere .npy vor fi apoi încărcate de scripturile de antrenare (Train_svm.py, Train_rf.py etc.).

---

3. (Exemplu) Train_svm.py / Train_rf.py

Scop

- Să încarce datele extrase (MFCC + etichete), să antreneze un model de clasificare (SVM sau Random Forest) și să îl salveze pe disc.

Pași cheie

1. **Încărcare date** din features_data.npy și labels_data.npy.
2. **Împărțire** train/test (ex. 80% vs. 20%) pentru a evalua corect modelul.
3. **Scaler** (StandardScaler): pentru a standardiza datele (opțional pentru Random Forest, recomandat pentru SVM).
4. **Antrenare model** (SVM sau Random Forest).
    - În cazul Train_svm.py, se poate folosi GridSearchCV pentru a găsi cei mai buni hiperparametri (C, gamma).
    - În cazul Train_rf.py, se pot alege parametrii optimi (n_estimators, max_depth etc.), dar dureaza extrem de mult pentru ca setul de date devine mare. Peste 12 ore.
5. **Evaluare** pe setul de test (calculare acuratețe, raport de clasificare).
6. **Salvare** a modelului în fișiere .pkl (svm_model.pkl, rf_model.pkl), alături de scaler (scaler.pkl).

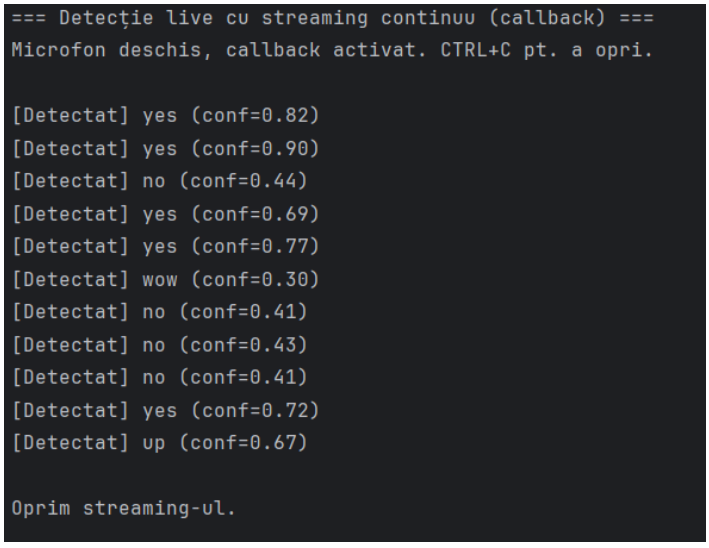
4. Live_detect.py (sau Preprocess_data.py, pentru varianta cu streaming)

Scop

- Să facă inferență (predicții) în timp real, folosind microfonul.

Pași cheie

1. **Inițializare PyAudio** pentru captură audio la 16 kHz, mono, chunk de 1024 eșantioane.
2. **Callback** (callback(in_data, frame_count, ...)) care primește bucăți de date audio (CHUNK) și le stochează într-un buffer (deque).
3. **În bucla principală**:
    - Se așteaptă să se adune 32000 bytes ≈ 1 secundă (16k semnale * 2 bytes).
    - Se extrage exact 1s de date și se convertește la numpy array.
    - Se face o verificare simplă de VAD (Voice Activity Detection) prin compararea energiei semnalului cu un prag (vad_energy_thresh).
    - Se normalizează amplitudinea (rms = 0.1).
    - Se extrag MFCC (40), se calculează media și abaterea standard → vector de 80 de elemente.
4. **Se folosește modelul și scaler-ul** (model.predict_proba, scaler.transform) pentru a obține eticheta prezisă și probabilitatea/confidența.
5. **Se compară confidența** cu un prag specific (THRESHOLDS) pentru fiecare cuvânt (pentru a evita prea multe false positive).
    - Dacă e peste prag, se consideră că s-a detectat cuvântul respectiv.
    - Pe lângă cuvintele target, există și o clasă „noise” care ajută la respingerea zgomotului.



5. Observații generale

1. **Modelul detectează cuvinte simple**: „yes”, „no”, „up”, „wow”.
    - Clasa „noise” apare pentru segmente care nu reprezintă niciun cuvânt țintă sau conțin doar zgomot.
    - În acest exemplu, cuvântul „down” a fost scos intenționat din *TARGET_WORDS* și se încearcă excluderea sa din predicția finală (chiar dacă modelul antrenat ar putea încă să îl recunoască).
2. **Fișierele de date**: subset (datele de bază) și subset_augmented (datele augmentate) trebuie create înainte de antrenare.
3. **În fișierul Live_detect.py** se folosește streaming audio în timp real. Acest lucru funcționează doar dacă există un microfon conectat și configurat corect la 16kHz.
4. **Pragurile (THRESHOLDS)** sunt obținute empiric sau pot fi ajustate după mai multe teste (pentru a echilibra rata de detectare și falsurile pozitive).

```
# Eliminăm și din THRESHOLDS
THRESHOLDS = {
    'yes': 0.60,
    'no': 0.39,
    'up': 0.56,
    'wow': 0.22,
    'noise': 0.50
}
```

6. Cum rulăm proiectul

1. **Pregătire set de date**:
    - Se ia subsetul din Google Speech Commands (sau un alt set).
    - Se plasează fișierele audio în structura subset/yes/, subset/no/ etc. + un folder subset/_background_noise_/ pentru zgomot.
2. **Augmentare** (opțional, dar recomandat pentru a îmbunătăți performanța):

python Augment_data.py

- Va genera fișiere noi în subset_augmented/.

3. **Preprocesare** (extragere MFCC + salvare etichete):

python Preprocess_data.py

- Vor fi create fișierele features_data.npy și labels_data.npy.

4. **Antrenare model** (Random Forest sau SVM):

python Train_svm.py

sau

python Train_rf.py

- Se încarcă fișierele .npy, se antrenează modelul și se salvează (svm_model.pkl / rf_model.pkl și scaler.pkl).

5. **Detectare live**:

python Live_detect.py

- Se pornește microfonul, se colectează flux audio, se extrag MFCC, iar modelul prezice în timp real cuvântul (dacă este peste un anumit prag).

---

7. Concluzii

- Acest proiect demonstrează o abordare simplă de **Keyword Spotting** și **Noise Detection**.
- Folosește tehnici standard de **MFCC** + **ML** (SVM, Random Forest) și un sistem minimalist de **Voice Activity Detection (VAD)**.
- Poate fi extins/îmbunătățit prin rețele neuronale (CNN, RNN), logistică adaptivă VAD, sau printr-o aplicație GUI.