Embedded Machine Learning

OBJECTIFS, DÉROULEMENT, CONCEPTS ET LES OUTILS

Sommaire

- Contexte
- Objectifs
- Fil rouge du cours
- Moyens
- Déroulement des séances
- Évaluation

Contexte

- Machine Learning EveryWhere
- Et pour l'embarqué ?
- Peut-on embarquer les algorithmes de prédiction du ML ?
- Oui :
  - Réalité : Android, iOS, plateformes quelconque (CPU, GPU, microcontrôleur, FPGA)
- Comment faire ?
  - Sans restriction de ressources, solution opensource(Tensorflow Lite).
  - Dans une optique DD, d'économie d'énergie et d'espace mémoire, et d'efficacité, et dans un optique industrielle souveraine et maîtrisée, cela mérite un certain **savoir faire**.

Contexte : choix ?

- Algorithmes ?
  - Approche algorithmique classique ?
  - Approche de type apprentissage supervisé ou non supervisé ?
  - Phases dans le processus du ML ? Peut-on les isoler ? Doit-on toutes les implémenter sur la cible ?
  - Préciser les interfaces nécessaires aux calculs ?
- Ressources ?
  - A-t-on besoin de langages de haut niveau interprétés ?
  - Faut-il un OS ?
  - Faut-il un processeur (CPU, GPU) capable ? Dans quelle mesure ?
  - Quelle puissance de calcul est requise selon les phases des algorithmes ?

Objectifs

1. Comprendre en profondeur le fonctionnement des algorithmes de ML.
2. Implémenter les différentes phases des algorithmes de ML :
   - Extraction de paramètres, apprentissage, prédiction.
3. Maîtriser la conception ML et le développement logiciel embarqué associé.
4. Utiliser le standard C++-20 dans le cadre du ML embarqué.
5. S'affranchir des dépendances logicielles :
   - Embarquer un code tiers implique des évolutions pas nécessairement souhaitées à vos produits et parfois des (re)qualifications coûteuses des systèmes.
   - On ne peut pas toujours embarquer un code tiers, fût-il opensource, cela dépend du contexte industriel.

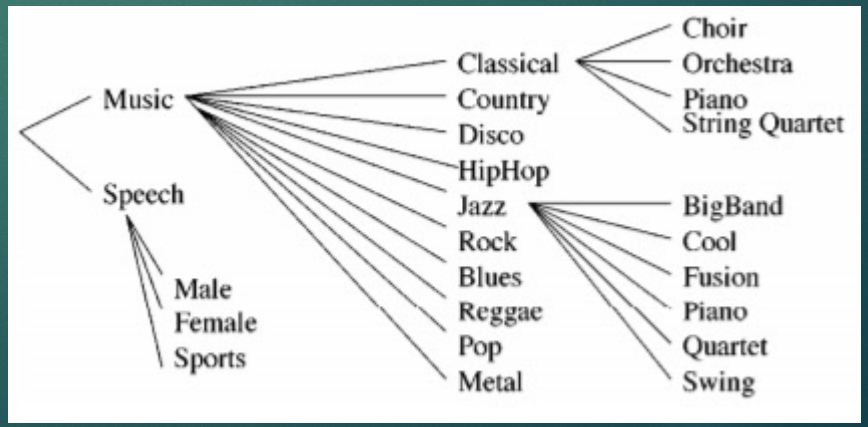
Les conséquences de ces objectifs (militants) :

- On réinvente parfois la roue, mais en même temps…
- On développe des compétences nouvelles qui permettent de faire évoluer les entreprises.
- On acquière une souveraineté sur les systèmes.
- On crée de la valeur dans toutes les dimensions.

Fil rouge

- La reconnaissance de style **musical** d'après un extrait de 30 secondes.

- Base de données GTZAN [1]:
  - 1000 pistes audio de 30 secondes
  - Format: .au (ou .wav)
  - Audio mono 16 bits échantillonné à 22050 Hz
  - 10 classes: Blues, Classique, Country, Disco, Hiphop, Jazz, Metal, Pop, Reggae, Rock



Moyens

Ce projet utilise nécessairement :

- un compilateur C++ à la norme 20, GCC-11 (10.2 min)
- cmake (>=3.16),
- Python 3.9 et Scikit Learn,
- Une Raspberrypi 4 avec OS 64 bits, GCC-11.
- Le dépôt GitLab mise à votre disposition :

https://gitlab.ensta-bretagne.fr/reynetol/embedded-machine-learning

Déroulement des séances

- Présentation du cours (1h)
- Extraction des paramètres des fichiers audio (3h)
- Arbre de décision (2h)
- Séparateur à Vaste Marge (SVM) (3h)
- Réseaux de neurones (ANN) (3h)
- Améliorations, optimisation et mesure des performances (4h)

Évaluation

- **Évaluation pour le 18/01/2022 (dépôt Moodle) :**
  - Rapport au format PDF police 11 de 10 pages maximum qui rend compte précisément de votre travail. Ce rapport ne contient pas l'intégralité du code, mais propose une analyse des parties pertinentes. Une analyse comparative des performances entre les différents algorithmes et les différentes implémentations est attendue.
  - Le code source (C++/Python/Matlab).

Évaluation, attendus et critères

- (**obligatoire**) Implémenter en C++-20
- (**obligatoire**) Veiller à donner la preuve que le code fonctionne sur la cible.
- (**obligatoire**) Évaluer (et mesurer) les complexités temporelle et mémoire de vos algorithmes.
- (**obligatoire**) Vérifier la performance de chaque classificateur embarqué sur les fichiers de test sélectionnés avant l'entraînement afin de les comparer.
- Programmer une extraction de paramètres en C++ performante :
  - (**obligatoire**) compléter le fichier au_reading.h afin de convertir les fichiers AU du dataset en DataVector en C++.
  - (**obligatoire**) efficace dans son exécution (1 Go de données musicales peuvent être traitées en 60 secondes ou moins),
  - minimale en termes de paramètres significatifs générés pour l'apprentissage :
    - (**obligatoire**) approche STFT,
    - (facultatif) approche MFCC.
- Arbres de décision :
  - (**obligatoire**) générer automatiquement le code C++ à compiler pour exécuter la prise de décision sur la cible,
  - (facultatif) proposer une implémentation de CART en C++-20,
  - (facultatif) proposer une implémentation simple de l'exécution d'une Random Forest en C++-20.
- Programmer une SVM :
  - (**obligatoire**) Élaborer une SVM optimale à l'aide de Scikit Learn, à partir des paramètres extraits en C++,
  - (**obligatoire**) Implémenter la prédiction en C++-20 sur la cible.
- Programmer un réseau de neurones :
  - (**obligatoire**) Implémenter un ANN en C++,
  - (facultatif) idem en utilisant le paradigme objet,
- (facultatif) Comparer les implémentations avec TensorFlow sur Raspberry Pi,
- (**obligatoire**) Comparer les approches et faire un tableau synthétique des caractéristiques et des performances de chacune des approches (CART, RF, SVM, ANN).

Évaluation, attendus et critères

Étiquettes de lignes	Somme de Points	Somme de Bonus
Analyse des différentes approches	2	
EXTRACTION MFCC		2
EXTRACTION STFT	3	
LECTURE .AU	2	
(vide)		
GENERATION AUTOMATIQUE DE CODE C++ (Sklearn ou CART.py)	2	
APRRENTISSAGE SKLEARN (sur données extraites depuis C++)	1	
PRÉDICTION EMBARQUÉE (CART, SVM, ANN)	3	
TENSOR FLOW LITE		2
Implémentation de la prédiction Random Forest en C++		2
Implémentation de CART en C++ à l'aide du paradigme objet		2
Implémentation ANN en C++ à l'aide du paradigme objet		2
Total général	**13**	**10**

Extractoin de paramètres audio « Features »

- Paramètres disponibles [1,2,3]:
  - **STFT/Spectrogramme: (μ,σ)**

$$X(\tau, f) = \int_{-\infty}^{+\infty} x(t)w(t-\tau)e^{-j2\pi f t}dt$$

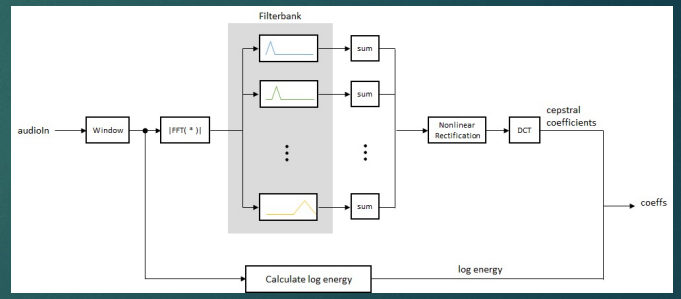
  - Zero Crossing Rate (ZCR):

$$ZC = \frac{1}{2}\frac{1}{T-1}\sum_{i=1}^{T}|signe(x[i]) - signe(x[i-1]|$$

  - Spectral Coefficient (SC):

$$C_t = \frac{\sum_{i=1}^{N} M_t[i] * i}{\sum_{i=1}^{N} M_t[i]}$$

  - **MFCC: (μ,σ)**

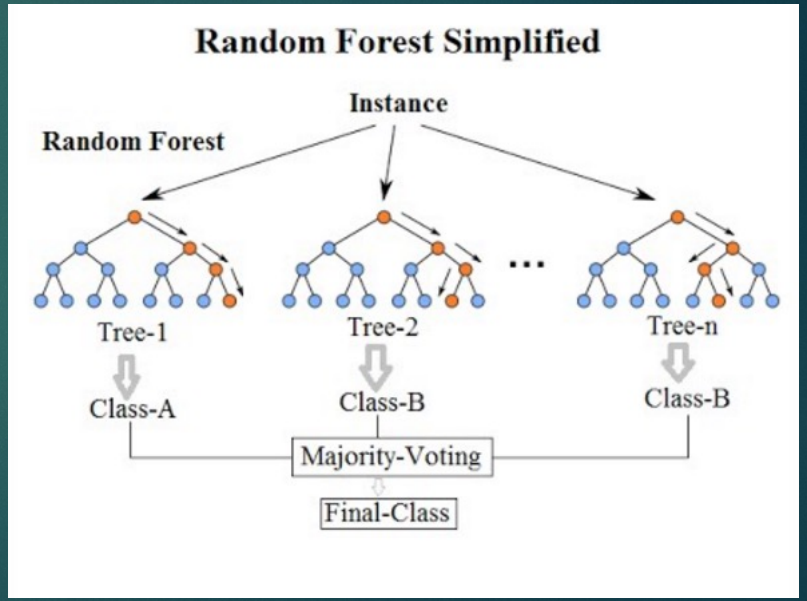


Objectif : réduire le nombre de paramètres et conserver des performances équivalentes.

[2] Ahmet Elbir et al. - Short Time Fourier Transform Based Music Genre Classification
[3] S. Sharma et al. - Novel Hybrid Model for Music Genre Classification based on Support Vector Machine

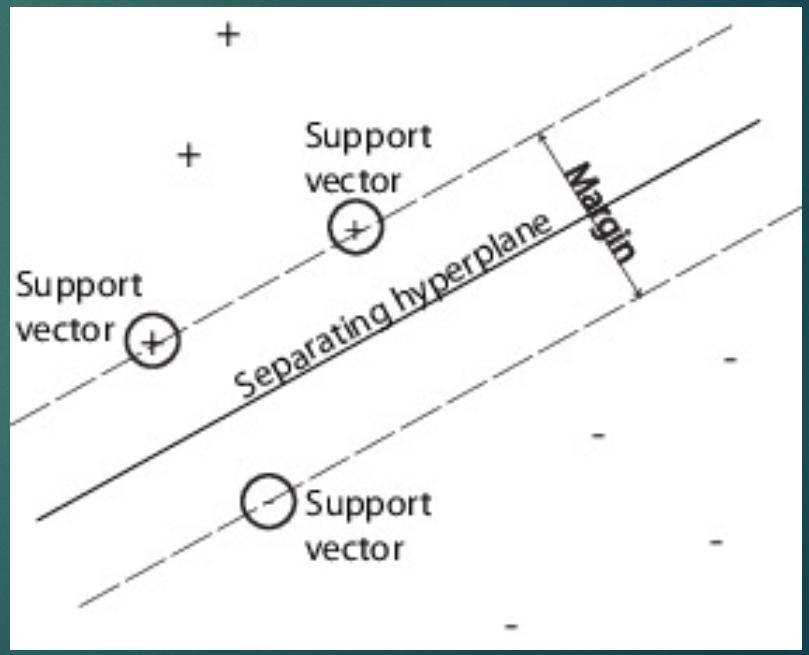
Classification:
Arbres de décision (CART) et Random Forests

- CART : supervisé et non paramétrique.
  - paramètres continus possibles en plus des discrets,
  - arbre binaire,
  - impureté d'un nœud (gini).
- Classification multi-classe possible.
- Programmation:
  - Decision Tress (Sklearn) / CART.py (OR)
  - Random Forest (Sklearn)
- Décision interprétable par l'être humain.



Classification: Séparateur à Vaste Marge (SVM)

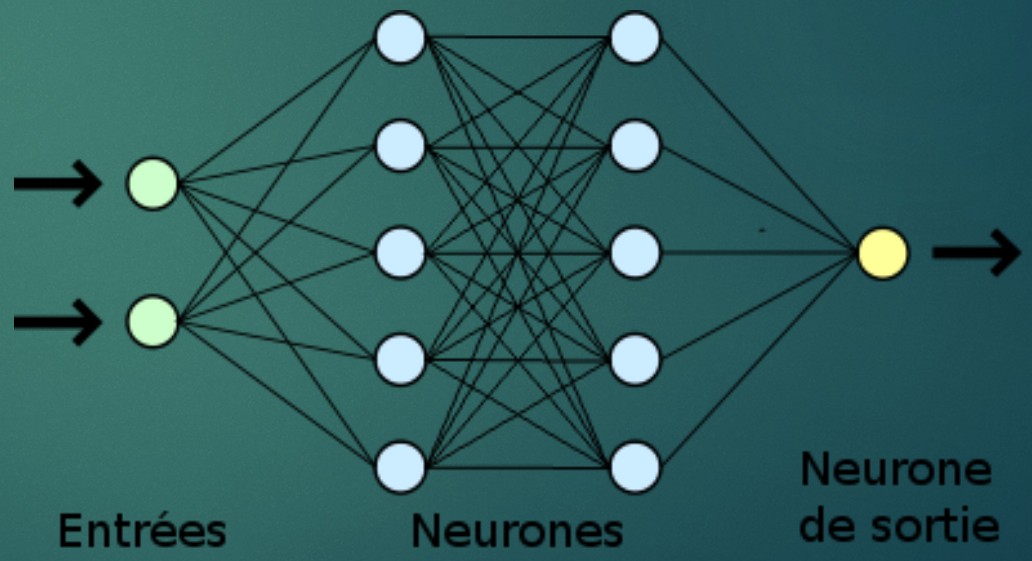
- Classificateur binaire linéaire [4]
- Classification multi-classe :
  - OneVsOne: $\binom{n}{2}$ classificateurs
  - OneVsAll: n classificateurs
- Programmation:
  - Scikit-Learn (LinearSVC, SVC)



[4] A. Guéron – Machine Learning avec Scikit-Learn (Ressources Machine Learning)

Classification: Réseaux de neurones (NN)

- Classificateur multi-classe [5]
- Programmation:
  - Sklearn
  - Tensorflow (Keras)



[5] A. Guéron – Deep Learning avec Keras et Tensorflow

Raspberry Pi 4

- Kit Raspbbery Pi 4:
  - Raspberry Pi 4 Model B 4 GB
  - MicroSD 16 GB, préchargée avec NOOBs
  - Boîtier noir Raspberry Pi 4
  - Alimentation 5,1V 3A
  - Câble micro HDMI/HDMI

Merci pour votre écoute