Rapport de Projet 2

SYSTEME D'INDENTIFCATION ET VERIFICATION DU LOCUTEUR

PRÉPARÉ PAR:
FIRDAWSE GUERBOUZI

ENCADRÉ PAR :
M KHARROUBI JAMAL

Sommaire

Objectifs

- Identifier et Vérifier l'identité de manière fiable les locuteurs à partir de leurs caractéristiques vocales.
- Comprendre et batir l'architecute de reconnaissance de locuteur
- Exploiter les forces des modèles Gmm Pour la reconnaissance de locuteur
- Fournir des mesures de performance et des évaluations précises pour permettre l'amélioration du système et l'optimisation des paramètres.

Introduction

La reconnaissance de locuteur est un domaine pluridisciplinaire qui fait appel à une multitude de connaissances et de compétences. Contrairement à l'empreinte vocale, qui est encore en développement, il est plutôt question d'identifier et d'authentifier la signature vocale unique de chaque individu. Les techniques utilisées dans ce domaine varient en fonction de la tâche de reconnaissance à réaliser, qu'il s'agisse de la vérification de l'identité d'un locuteur ou de l'identification d'un locuteur parmi plusieurs.

Malgré les bonnes performances obtenues en laboratoire, les systèmes de reconnaissance de locuteur restent encore insuffisants pour répondre aux exigences de domaines nécessitant un haut degré de sécurité. Cela souligne la nécessité de continuer à améliorer les techniques de modélisation, d'explorer de nouvelles approches et de prendre en compte les défis spécifiques liés à la sécurité et à la fiabilité des systèmes.

Dans ce rapport, nous explorerons l'utilisation des modèles GMM (Gaussian Mixture Models) dans un système de reconnaissance de locuteur. Nous mettrons en évidence l'importance de la qualité de la dataset, des techniques d'optimisation des paramètres et de l'évaluation rigoureuse pour obtenir des performances fiables.

Architecture du système : **Identification** et **Vérification**

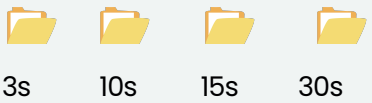
1. **Collecte des enregistrements**

2. **Extraction des mfccs**

3. **Suppression du silence**

4. **Modélisation gmm**

5. **Separation des tests en segments**



6. **Calcul des scores**

7. **Identification**

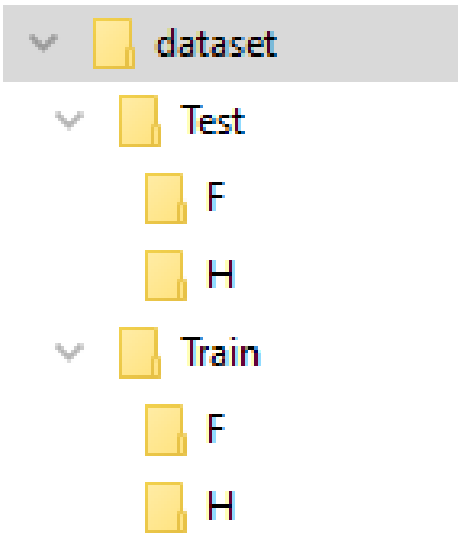
- **Recherche du score du modèle qui maximise**
- **Calcul du taux d'erreur**
- **Comparaison**

8. **Vérification**

- **Calcul du Taux de faux acceptés et Taux de faux rejets pour un intervalle de seuils**
- **Tracer de la courbe DET et considérer un seuil**
- **Prendre une décision**

Collecte de la dataset

- La première étape du projet consistait à collecter un ensemble de données. Étant donné que notre projet se concentre sur l**a reconnaissance du locuteur dans un ensemble fermé,** nous avons limité la collecte à des enregistrements audio de 1 minute pour l'entraînement et de 1 minute pour les tests pour chaque membre de notre classe.
- Nous avons choisi d'étudier chaque sexe séparément, car les hommes et les femmes présentent des caractéristiques bien distinctes.



- Notre base de données initial est formé de 36 enregistrements (36 min) dont 20 convients aux femmes et 16 aux hommes

- De plus nous avons segmenter chaque enregistrement de Test en des segments de 3s , 10s ,15 s et 30s et donc on a fini par avoir 576 segments de test . (18*20*6*4*2=576)

Lien du drive :
https://drive.google.com/drive/folders/1hOytKy3TyQVWsns7AVw1upvZtNLpDljg?usp=sharing

Extraction des MFCCS

Dans la deuxième étape, nous avons procédé au chargement des données audio à partir de fichiers WAV. En utilisant la fonction "mfcc" fournie par le module "python_speech_features", nous avons calculé les coefficients MFCC correspondants pour chaque enregistrement vocal. Ces coefficients ont ensuite été stockés dans un dossier appelé "mfcc", avec une structure identique à celle de l'ensemble de données initial.

```
# Définition des paramètres MFCC
numcep = 13
nfilt = 26
nfft = 512
lowfreq = 0
highfreq = None
preemph = 0.97
ceplifter = 22
winlen = 0.025
winstep = 0.01

lang_dir = 'dataset/Train/F/'
for filename in os.listdir(lang_dir):
    (rate, sig) = wav.read(lang_dir+filename)
    mfcc_feat = python_speech_features.mfcc(sig, rate, winlen=0.025, winstep=
    root, ext = os.path.splitext(filename)
    with open('mfcc/train/F/'+root+'.mfcc', 'wb') as f:
        pickle.dump(mfcc_feat, f)
```

Prétraitement

Afin d'exploiter pleinement ma base de données il était nécessaire de filtrer les trames de silence qui présentent des caractéristiques communes à tous les enregistrements.
Pour cela, j'ai utilisé une modélisation bigaussienne basée sur l'énergie pour classifier les trames de chaque enregistrement en deux groupes : celles avec une énergie élevée et celles avec une énergie faible.

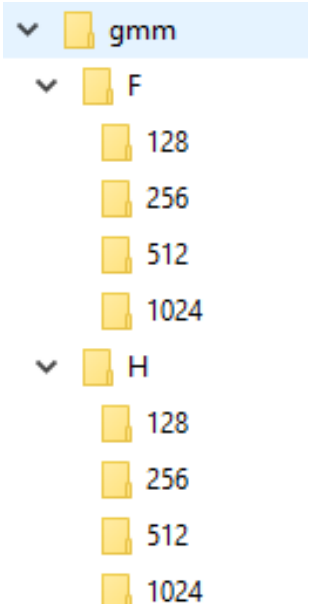
```
n_components = 2  # number of GMM components
gmm = GaussianMixture(n_components=n_components)  # create GMM object
gmm.fit(mfcc_feat)  # fit GMM model to data
log_prob = gmm.score_samples(mfcc_feat)
# Select frames with log-likelihood above a threshold
threshold = np.percentile(log_prob, 10)  # adjust percentile to select more o
non_silent = mfcc_feat[log_prob >= threshold]
```

Modélisation GMMS

la phase suivante consistait à construire un modèle GMM pour chaque locuteur et à le former en utilisant son enregistrement correspondant. Il est crucial de sélectionner soigneusement le nombre de gaussiennes dans un modèle GMM, car cela peut avoir un impact significatif sur les performances d'un système de reconnaissance. Ainsi, j'ai décidé de créer quatre modèles GMM avec des nombres de gaussiennes différents : 128, 256, 512 et 1024. Cette diversité de modèles me permettait de comparer leurs performances respectives et de déterminer celui qui conviendrait le mieux à ma tâche de reconnaissance de locuteur.

```
def model(lang_dir,nbr):
    for filename in os.listdir(lang_dir):
        root, ext = os.path.splitext(filename)
        with open(lang_dir+filename, 'rb') as f:
            file = pickle.load(f)
        gmm = GaussianMixture(n_components=nbr, covariance_type='full')
        gmm.fit(np.vstack(file))
        if('/F/'in lang_dir):
            with open('gmm/'+'/F/'+str(nbr)+'/'+root+'.'+str(nbr)+'.gmm', 'wb') as f:
                pickle.dump(gmm, f)
        else :
            with open('gmm/'+'/H/'+str(nbr)+'/'+root+'.'+str(nbr)+'.gmm', 'wb') as f:
                pickle.dump(gmm, f)
```

J'ai créé une fonction appelée 'model' qui génère un modèle avec le nombre de gaussiennes correspondant pour tous les enregistrements présents dans le dossier spécifié. Ces modèles sont ensuite **stockés avec l'étiquette de l'enregistrement correspondant**, ce qui facilite l'identification ultérieure des locuteurs.




```
lang_dir = 'mfcc/train/H/'
model(lang_dir,128)
model(lang_dir,256)
model(lang_dir,512)
model(lang_dir,1024)
```

Segmentation des fichiers tests

Pour évaluer concrètement la performance du système de reconnaissance, j'ai choisi de diviser chaque enregistrement de 60 secondes en segments de différentes durées, à savoir **3s, 10s, 15s et 30s.** Cette approche nous permet d'analyser les performances du système sur des échantillons de temps variés, en prenant en compte la variation potentielle des performances en fonction de la durée de l'enregistrement. Ainsi, nous pourrons déterminer la durée optimale pour obtenir les meilleurs résultats de reconnaissance.

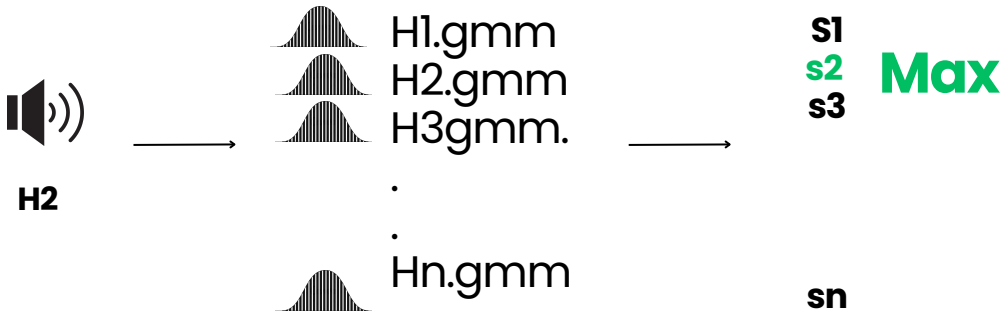
```
def segment(filename,nbr,lang_dir):
    with open(lang_dir+filename, 'rb') as f:
        file = pickle.load(f)
    num_frames, num_coefficients = file.shape
    # Calculate the number of frames per segment
    root, ext = os.path.splitext(filename)
    frames_per_segment = num_frames // nbr
    for i in range(nbr):
        start_frame = i * frames_per_segment
        end_frame = (i + 1) * frames_per_segment
        segment = file[start_frame:end_frame, :]
        if('/F/'in lang_dir):
            with open('Segments/'+str(int(60/nbr))+'/F/'+root+'.'+str(int(60/nbr))+"."+str(i
                pickle.dump(segment, f)
        else :
            with open('Segments/'+str(int(60/nbr))+'/H/'+root+'.'+str(int(60/nbr))+"."+str(i
                pickle.dump(segment, f)
```

Afin de simplifier la tâche, j'ai créé la fonction "segment()" qui permet de répartir chaque fichier audio en segments en fonction du nombre de segments souhaité(nbr) , en supposant que chaque fichier MFCC représente une durée de 60 secondes. J'ai veillé à ce que c**haque segment extrait conserve son étiquette d'origine** tout en précisant sa durée, afin de faciliter l'identification et le traitement ultérieur des segments.


```
lang_dir='mfcc/test/F/'
for filename in os.listdir(lang_dir):
    segment(filename,20,lang_dir)
```

Identification

Pour identifier chaque locuteur, il est nécessaire de parcourir tous les modèles GMM de tous les locuteurs et de **rechercher celui qui maximise** une certaine mesure de similarité ou de probabilité.



- J'ai crée d'abord une Fonction qui calcule le score correspondant à un modèle GMM spécifique pour un fichier MFCC donné.

```
def score(file,gmm):
    log_likelihood = gmm.score_samples(file)
    mean =  np.mean(log_likelihood)
    return mean
```

- Puis une fonction qui parcourt chaque fichier dans le dossier MFCC des enregistrements, calcule le score pour chaque modèle GMM et stocke chaque score, étiquette et modèle dans une dataframe.

```
def calculate_gmm_scores(lang_dir,nbr):
    gmms,roots = loadGmm(lang_dir,nbr)
    models = []
    results = []
    for filename in os.listdir(lang_dir):
        scores =[]
        names = []
        root, ext = os.path.splitext(filename)
        with open(lang_dir+filename, 'rb') as f:
            file = pickle.load(f)
        for i in range(len(gmms)):
            root, ext = os.path.splitext(filename)
            scores.append(score(file,gmms[i]))
            results.append({
                        'Recording Name': root,
                        'Model Name': roots[i],
                        'Score': score(file,gmms[i])
                    })
    df = pd.DataFrame(results)
    return df
```

	Recording Name	Model Name	Score
0	H1.3.1	H1.128	-50.886092
1	H1.3.1	H2.128	-454.760850
2	H1.3.1	H4.128	-549.383008
3	H1.3.1	H5.128	-590.224754
4	H1.3.1	H6.128	-257.973788
5	H1.3.1	H7.128	-88.847773
6	H1.3.1	H8.128	-73.618699
7	H1.3.1	H9.128	-73.161958

Le stockage des résultats obtenus sous cette structure a été d'une utilité énorme pour toutes les étapes qui suivent !

Test et comparaison

Il était également nécessaire d'analyser les performances de mon modèle en fonction de la durée des enregistrements et du nombre de gaussiennes. Pour cela, j'ai testé mes modèles en commençant par des enregistrements de 3 secondes, puis en augmentant progressivement la durée à 10 secondes, 15 secondes et enfin 30 secondes, pour chaque configuration de nombre de gaussiennes spécifique.Pour cela j'ai dévelopé les fonctions suivantes

```
def verify_max_score(df):
    # Find the index of the maximum score
    max_score_index = df['Score'].idxmax()

    # Get the Recording Name and Model Name of the observation with the maximum score
    max_score_recording = df.loc[max_score_index, 'Recording Name']
    max_score_model = df.loc[max_score_index, 'Model Name']

    # Verify if the Recording Name and Model Name start with the same first 2 characters
    same_start = max_score_recording[:2] == max_score_model[:2]

    return same_start
```

- La fonction **verify_max_score()** est une fonction qui renvoie un booléen. Elle cherche le score du GMM (Gaussian Mixture Model) du locuteur qui maximise pour chaque segment, puis vérifie s'il appartient à la même identité.

```
def calculError(df,sec):
    errors = []
    for j in range(9):
        #print(j)
        for k in range(int(60/sec)):
            if(j!=2):
                extracted_df = df[df['Recording Name'] == 'H'+str(j+1)+'.'+str(sec)+'.'+str(k+1)]
                extracted_df = extracted_df.reset_index(drop=True)
                verify_max_score(extracted_df)
                errors.append(verify_max_score(extracted_df))
    return errors.count(False) / len(errors)
```

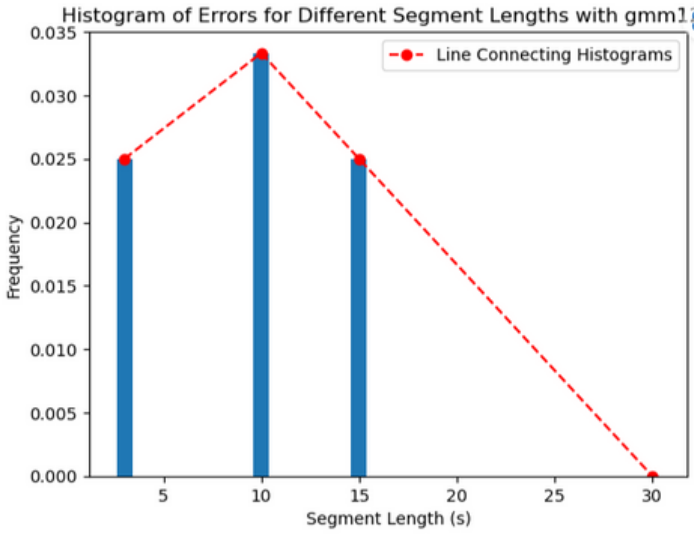
- La fonction **calculError()** est une fonction qui, dans la même dataframe, extrait les occurrences de chaque segment d'MFCC (par exemple, H1.3.1.mfcc) et fait appel a la fonction précédente. À la fin, elle renvoie le rapport d'erreur pour les segments d'une durée spécifique.

Test et comparaison

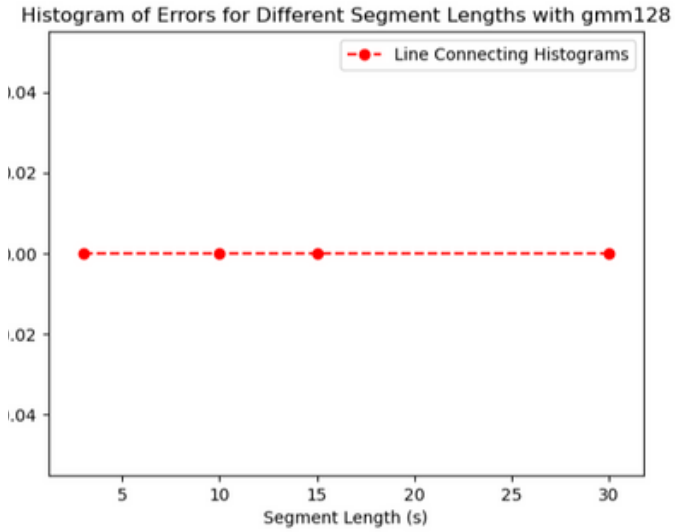
Les fonctions précédentes permettent de tester facilement toutes les différentes segments en variant les nombres de gaussiennes, et les résultats étaient comme suit:

- 128 gaussiennes

Femmes

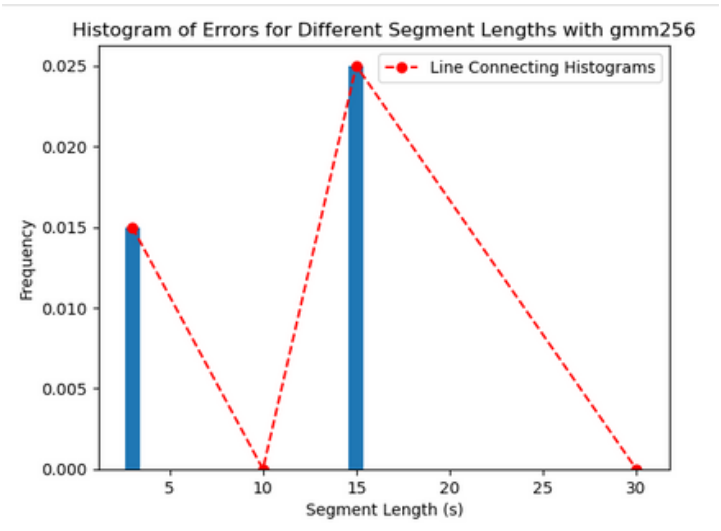


Hommes

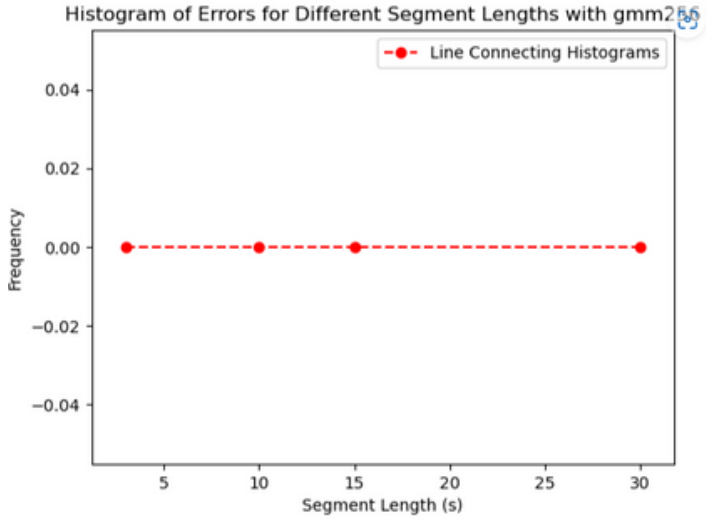


- 256 gaussiennes

Femmes



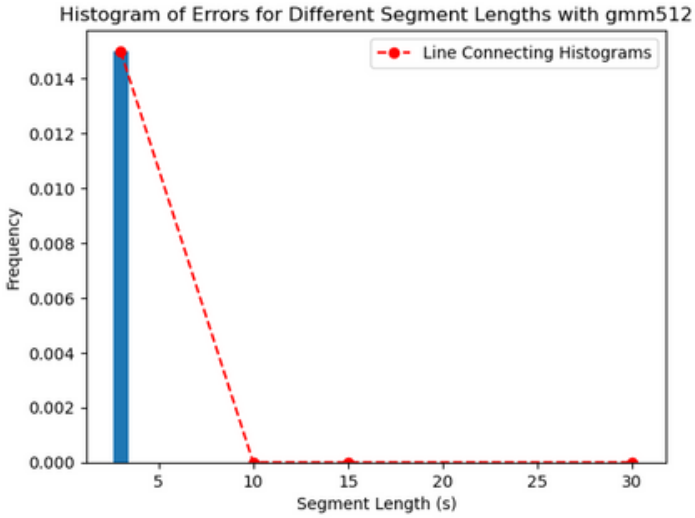
Hommes



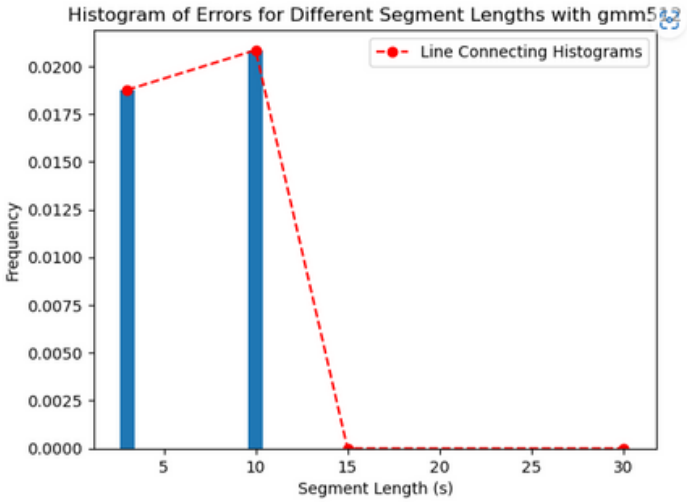
Test et comparaison

- 512 gaussiennes

Femmes

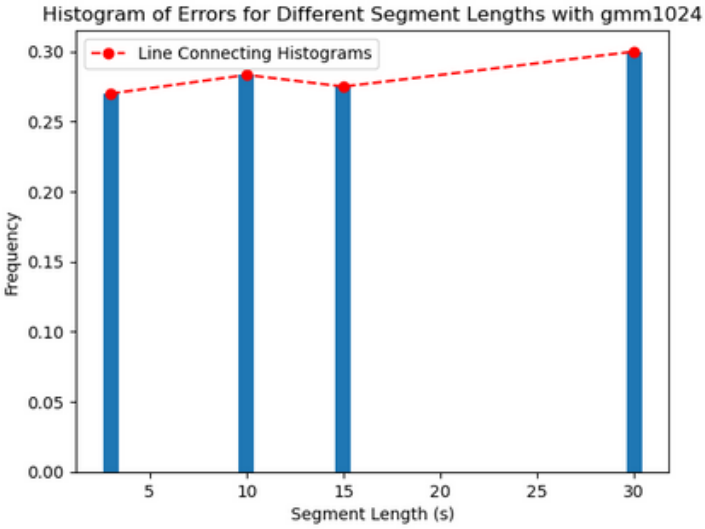


Hommes

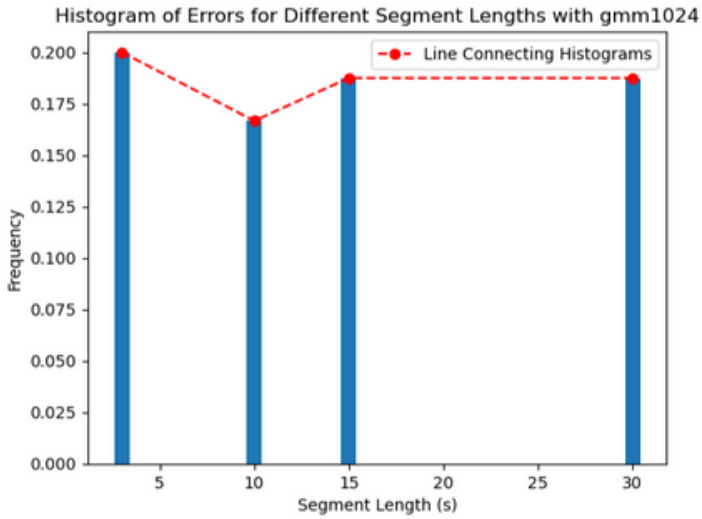


- 1024 gaussiennes

Femmes



Hommes

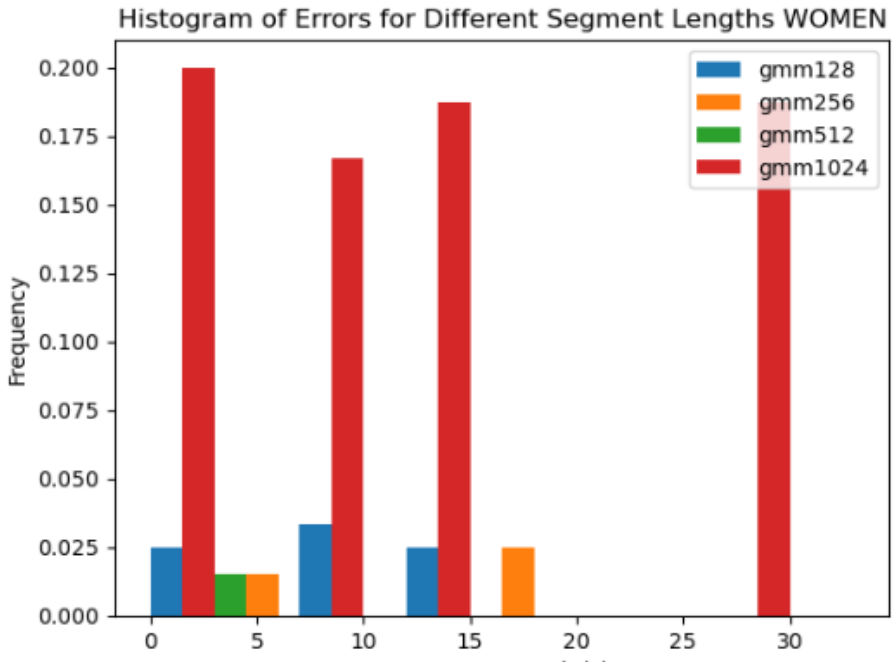


D'après tous les différents tests effectués, il est observé qu'en augmentant la durée de l'enregistrement, le taux d'erreur diminue. Plus la durée de l'enregistrement est longue, plus les modèles GMMs ont tendance à obtenir de meilleurs résultats en termes de taux d'erreur

comparaison

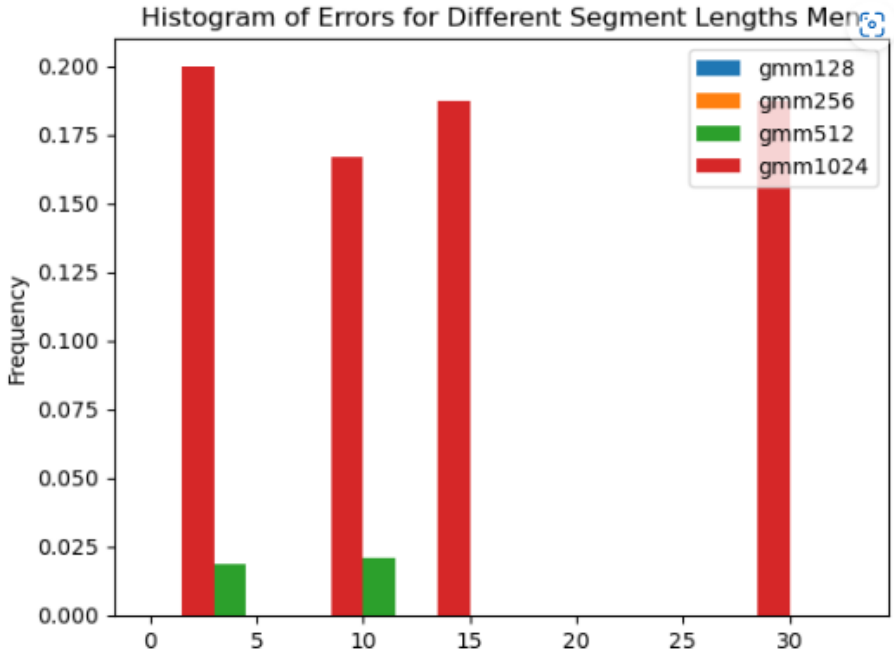
L'observation de l'histogramme récapitulatif met en évidence une tendance intéressante : à mesure que le nombre augmente (1024), le taux d'erreur augmente également. Cette corrélation peut être attribuée à la faiblesse de la dataset utilisée. Lorsque la dataset est de qualité insuffisante, cela signifie qu'elle peut manquer de diversité, de représentativité ou de quantité de données pertinentes.

- Femmes



Pour les enregistrements des femmes les meilleurs performances on été obtenues par les gmms de 512 et les enregistrements de 30s

- Hommes



Pour les enregistrements des hommes les meilleurs performances on été obtenues par les gmms de 128 et 256 et des enregistrements de 30s

Vérification

Pour la tâche de vérification, il était crucial de **déterminer le seuil** sur lequel la décision du système serait basée pour l'acceptation ou le rejet. Pour ce faire, les étapes suivantes ont été nécessaires:

- Variation d'un intervalle de seuil, allant du score minimum au score maximum.
- Calcul du taux de faux rejets et du taux de faux acceptés pour chaque seuil de l'intervalle.
- Traçage de la courbe DET (Detection Error Tradeoff) représentant les variations du taux de faux rejets par rapport au taux de faux acceptés.
- Détermination du score seuil à partir du point TEE (Taux d'Erreur Équivalent), qui correspond au point où le taux de faux rejets est égal au taux de faux acceptés.

Ces étapes permettent de définir un seuil de décision optimal, basé sur l'équilibre entre le taux de faux rejets et le taux de faux acceptés, afin de maximiser les performances de la vérification du système.

Pour ce faire nous avons calculer le nombre total d'acces client et le nombre total d'acces imposteur

```
# Calculate NAI and NAC once
NAI = (df['Recording Name'].str[:2] != df['Model Name'].str[:2]).sum()
NAC = (df['Recording Name'].str[:2] == df['Model Name'].str[:2]).sum()
```

Nous avons développer cette fonction qui pour chaque seuil donné calcule le Taux du faux acceptés et celui de faux rejetés

```
def det(df, seuil):
    FA = ((df['Score'] > seuil) & (df['Recording Name'].str[:2] != df['Model Name'].str[:2])).sum()
    FR = ((df['Score'] < seuil) & (df['Recording Name'].str[:2] == df['Model Name'].str[:2])).sum()
    TFA = FA / NAI
    TFR = FR / NAC
    return TFA, TFR
```

Et pour un intervalle de 5000 valeurs [min, max], on utilise la fonction précédente pour calculer les résultats, qui sont ensuite stockés dans des listes.

```
minimum = df['Score'].min()
maximum = df['Score'].max()
seuils = np.linspace(minimum, maximum, num=5000)
tfa_list = []
tfr_list = []

for seuil in seuils:
    tfa, tfr = det(df, seuil)
    tfa_list.append(tfa)
    tfr_list.append(tfr)
```

Courbe de DET

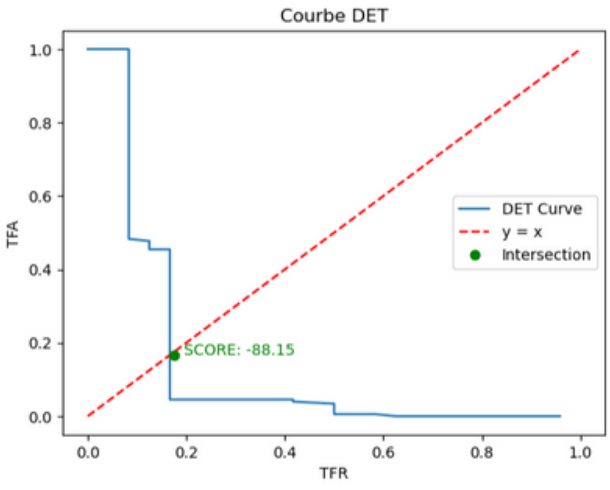
Par la suite, nous essayons de tracer les deux listes précédemment calculées et de représenter graphiquement la courbe y = x. Nous recherchons le point approximatif où le taux de faux acceptés (TFA) est égal au taux de faux rejets (TFR).

```
# Find intersection with the line y=x
threshold = 0.01  # set a threshold for the intersection
intersection_point = None
for index, (tfa, tfr) in enumerate(zip(tfa_list, tfr_list)):
    if abs(tfa - tfr) < threshold:
        intersection_point = (tfa, tfr)
        intersection_index = index
        break

# Plotting the DET curve
plt.plot(tfr_list, tfa_list, label='DET Curve')

# Plotting the line y = x
plt.plot([0, 1], [0, 1], color='red', linestyle='--', label='y = x')

tee_score = seuils[intersection_index]

# Plotting the intersection point
if intersection_point is not None:
    plt.plot(*intersection_point, 'go', label='Intersection')
    tee_score = seuils[intersection_index]
    plt.text(intersection_point[0] + 0.02, intersection_point[1], f'SCORE: {tee_score:.2f}', color='green')

plt.xlabel('TFR')
plt.ylabel('TFA')
plt.title('Courbe DET')

# Adding legend
plt.legend()

# Display the plot
plt.show()
```

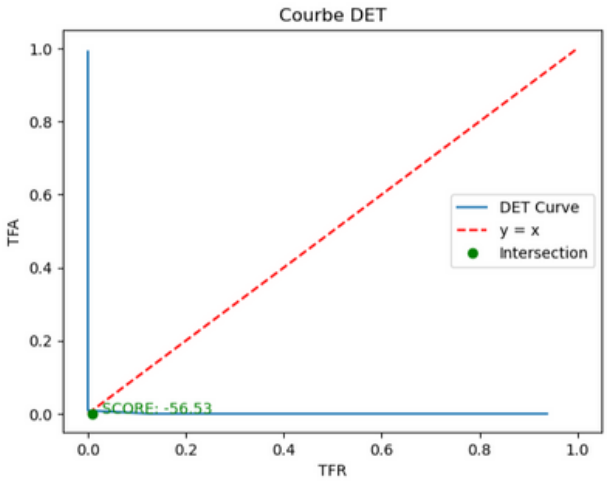
Nous avons effectuer le test sur les enregistrements d e30 s puisqu'ils offrent les meilleurs performances et les résultats étaient comme suit :

- **Gmm de 128 gaussiennes**

Femmes



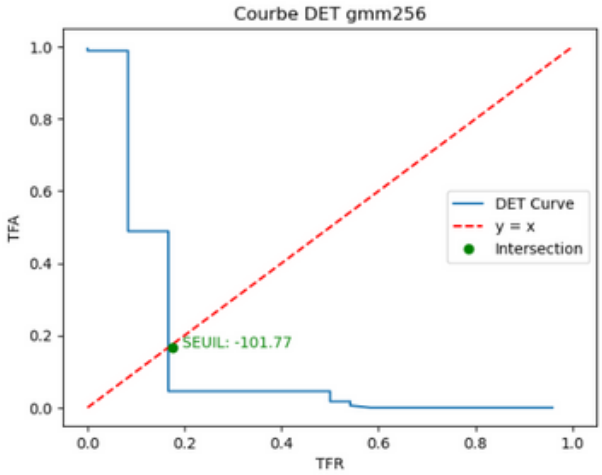
Hommes



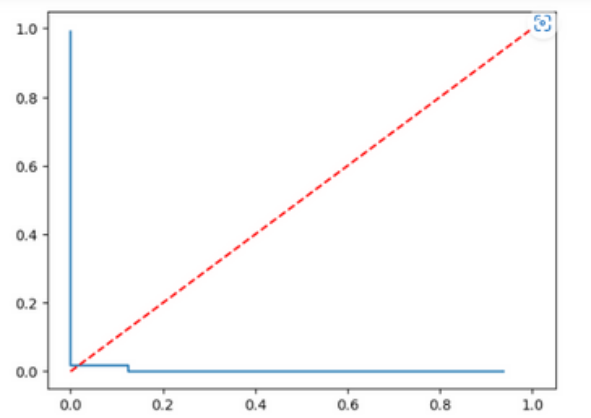
Courbe de DET

- **Gmm de 256 gaussiennes**

Femmes

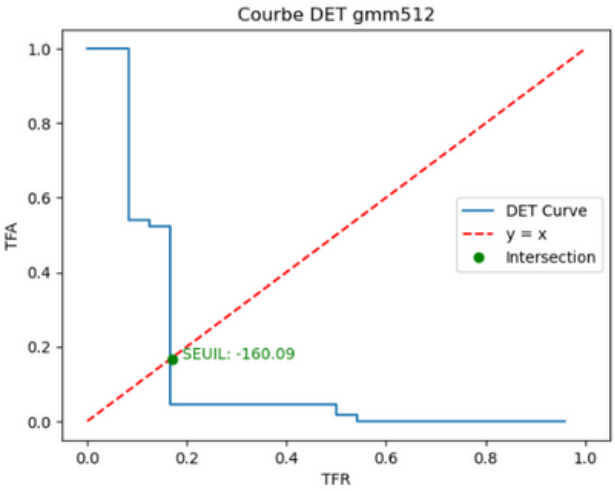


Hommes

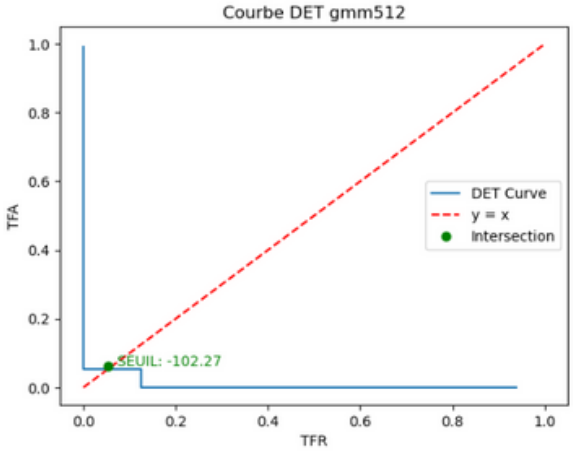


- **Gmm de 512 gaussiennes**

Femmes

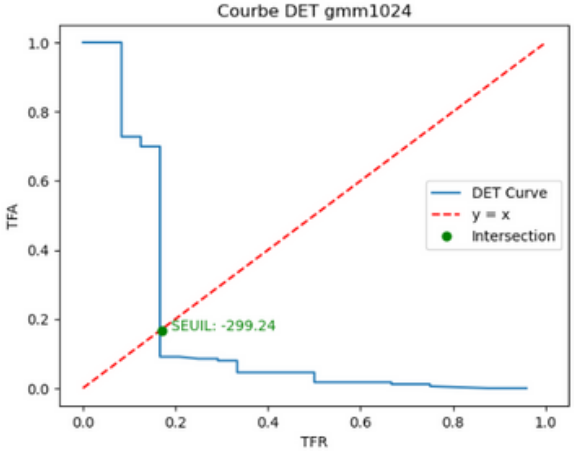


Hommes

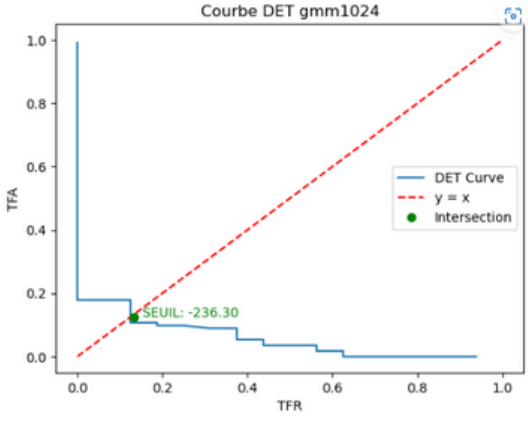


- **Gmm de 1024 gaussiennes**

Femmes

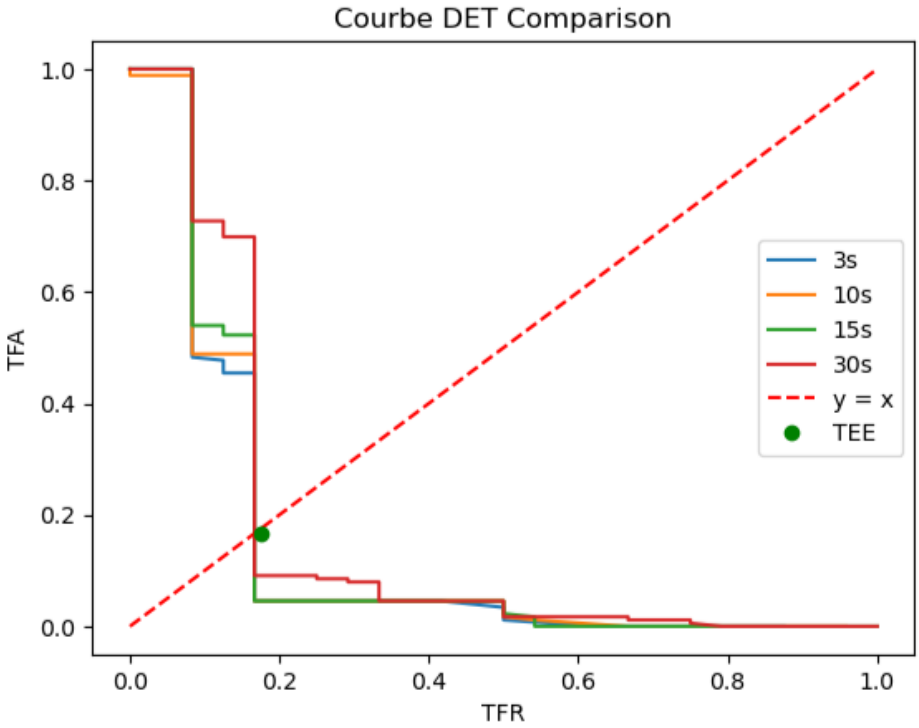


Hommes

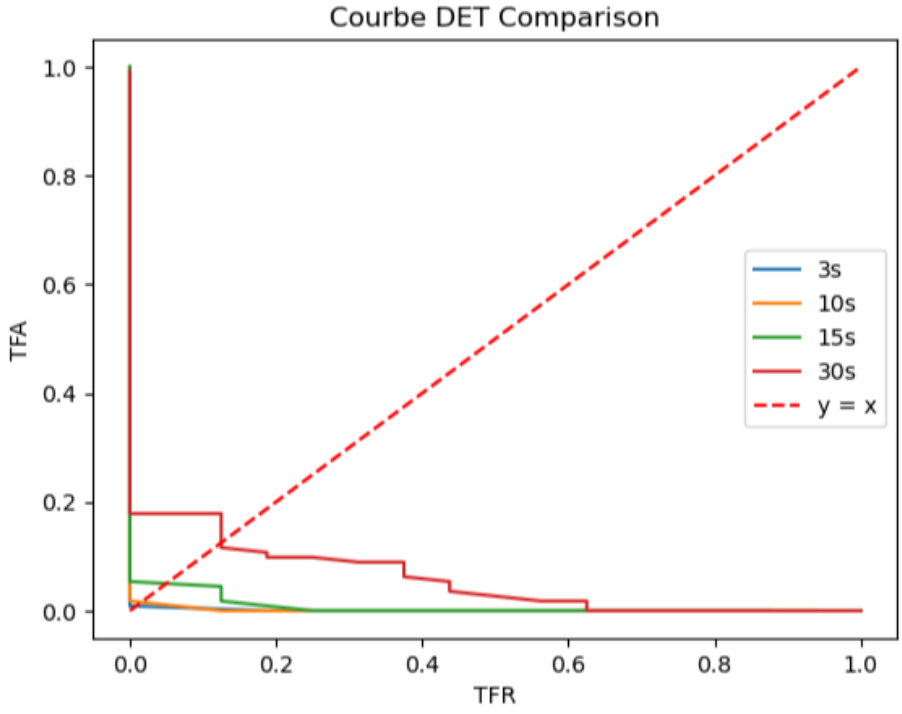


Comparaion

- femmes



- Hommes



Conclusion et pistes d'amélioration

Ce projet a été une expérience passionnante d'apprentissage et d'analyse dans le domaine de la reconnaissance de locuteur. Il a débuté par la collecte d'une dataset équilibrée et diversifiée, suivie d'étapes cruciales telles que le prétraitement des données et l'élimination du silence basée sur l'énergie.
En utilisant les modèles GMM, nous avons pu exploiter efficacement les caractéristiques vocales pour la reconnaissance et la vérification du locuteur.
En résumé, ce rapport met en évidence l'importance fondamentale de la qualité de la dataset, de l'optimisation des paramètres et de l'évaluation à l'aide de la courbe DET dans un système de reconnaissance de locuteur basé sur les modèles GMM.
Cependant, pour améliorer les performances et répondre aux exigences de domaines spécifiques, plusieurs pistes d'amélioration sont proposées. Parmi celles-ci, nous suggérons l'exploration de modèles plus avancés tels que les réseaux neuronaux, l'augmentation de la taille et de la diversité de la dataset, l'utilisation de techniques de prétraitement avancées, la fusion de plusieurs modèles et l'intégration de l'apprentissage profond.
En adoptant ces pistes d'amélioration, il est possible de renforcer les performances du système de reconnaissance de locuteur et de répondre aux besoins de domaines nécessitant une identification précise des individus et une sécurité élevée.
Ce rapport offre ainsi un aperçu des possibilités et des défis de la reconnaissance de locuteur, et encourage les recherches futures pour continuer à améliorer cette technologie pluridisciplinaire.