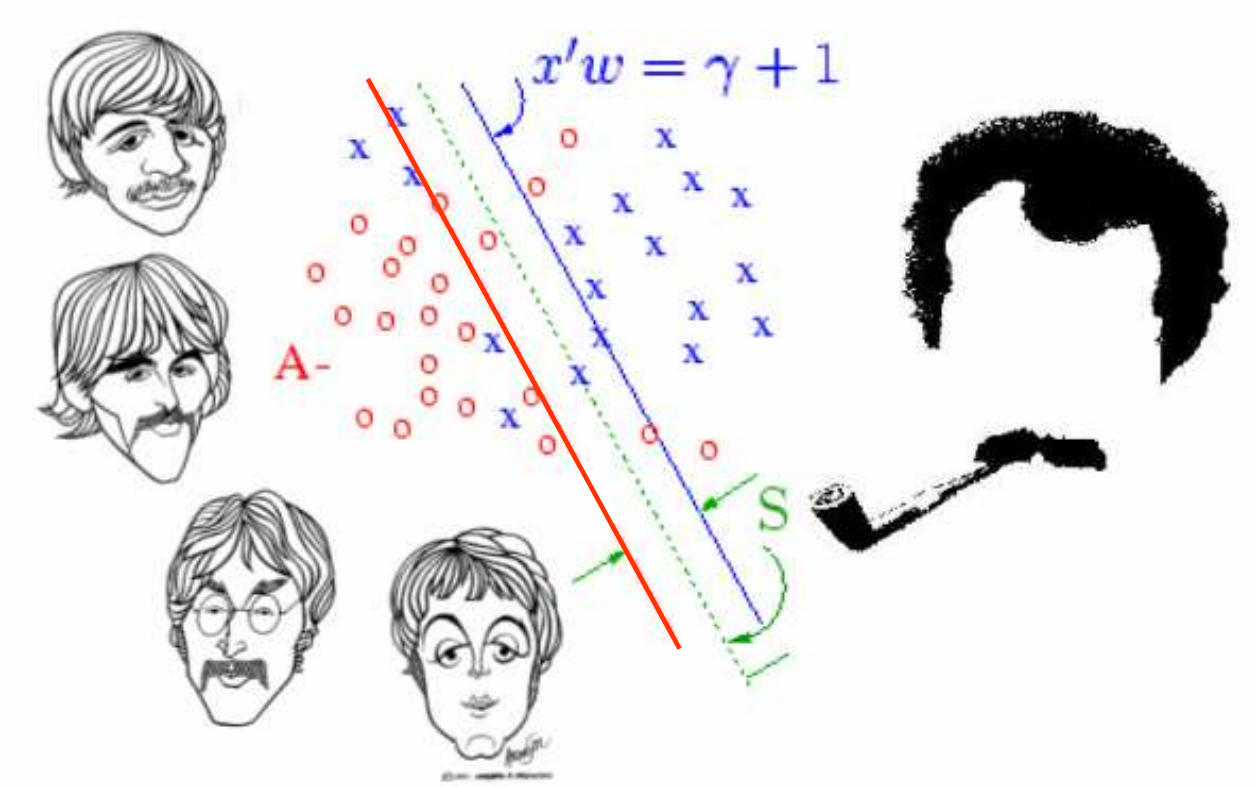
MASTERE MVA 2004/05

Application des SVM pour la discrimination de contenus musicaux :

« The Beatles » vs. George Brassens



Francisco Sánchez Vega (sanchez@enst.fr)

Cours d'introduction aux théories de l'apprentissage de ***Robert Azencott***

Paris, le 31 janvier 2005

Sommaire

1 Introduction

Ce projet a été développé pour le cours « Introduction aux théories de l'apprentissage » de Robert Azencott à l'ENS Cachan, dans le cadre du Mastère de Recherche en Mathématiques Appliquées « Mathématiques, vision, apprentissage ».

J'ai ai essayé d'utiliser les outils et les idées qui ont été présentées pendant le cours pour faire face à un défi concret dans le domaine de la classification de contenus musicaux, et plus concrètement, j'ai voulu m'en servir des svm pour apprendre l'ordinateur à faire la distinction entre les chansons du groupe britannique « The Beatles » et celles du chanteur français George Brassens.

Dans ce rapport, on fera tout d'abord une présentation de la problématique abordée et des outils choisis pour chacune des étapes du processus de traitement de données.

Après, on présentera la méthode d'analyse PLP (Prédiction Linéaire Perceptuelle) qui est celle que j'ai utilisée pour l'obtention des vecteurs de caractéristiques (ou « *feature vectors* ») pour l'apprentissage.

Ensuite, on décrira quelques points d'intérêt concernant les deux moteurs svm utilisés dans mon étude : OSU-SVM (interface pour Matlab du moteur LIBSVM de Chih-Jen Lin et al.) et SVM-Light de Thorsten Joachims.

On montrera aussi les résultats des simulations effectuées et on fera une étude de l'influence du choix des différents paramètres pour l'apprentissage.

Finalement, on fera une petite discussion et une mise en valeur du travail accompli dans laquelle on parlera des principales difficultés rencontrées, ainsi que des possibles améliorations face à l'avenir.

2 Modélisation du problème à traiter

2.1 Choix du sujet et objectifs du projet

Quand j'ai rencontré M. Azencott la deuxième semaine de décembre pour fixer un sujet pour mon projet, je n'avais pas encore aucune idée concrète dans ma tête. C'est pourquoi il m'a proposé d'aborder un problème générique consistant à générer des vecteurs à des composants aléatoires, créer une fonction affine de coefficients aléatoires aussi et essayer de faire l'apprentissage sur ces éléments.

Quelques semaines plus tard, par contre, je continuais à chercher un sujet originel pour mon projet. Un jour, quand j'écoutais de la musique dans ma chambre, j'ai trouvé une idée qui m'a paru très attractive: serait-il possible d'utiliser des svm pour apprendre un ordinateur à faire la distinction entre deux groupes de musique ou deux chanteurs différents ?

C'est ainsi que j'ai décidé d'utiliser les idées que l'on a appris pendant le cours du Mastère pour essayer de réussir à ce but, en choisissant un exemple concret : « The Beatles » vs. George Brassens.

En principe, il s'agit d'une idée originelle, qui n'est pas basée sur aucun article préexistant et qui a comme but d'évaluer les performances des svm sur un problème réel. Je suis sûr qu'il y en aura des nombreux travaux réalisés sur le sujet de la classification automatique de musique et que certains d'entre eux utiliseront des svm, mais mon objectif n'était pas celui de me placer dans le cadre de l'état de l'art, sinon plutôt d'arriver à construire un classificateur par moi-même et évaluer ses performances.

C'est ainsi que j'ai commencé à travailler sur mon projet avec des nombreuses doutes sur les résultats finaux. Comment obtenir des *features* conduisant à une classification robuste ? Même si on trouve les *features* les plus performantes, les vecteurs de caractéristiques seront-ils séparables ? Quel taux de réussite et d'erreur peut-on aspirer à atteindre avec une approche de ce type ? ... Dans la suite de ce rapport, j'essaierai de répondre à ces questions à partir des résultats obtenus empiriquement, en expliquant toujours les concepts mathématiques sous-jacents.

2.2 Description de la problématique abordée

Comme je viens d'expliquer, l'objectif pratique du projet est de construire un classificateur capable de décider si une certaine chanson appartient aux Beatles ou à Brassens.

Pour la plupart d'entre nous, qui avons entendu ces chansons très souvent, il peut sembler une tache facile. Le fait qu'ils chantent en des langues différentes (l'anglais et le français), par exemple, rend la tache de la discrimination assez évidente pour une personne de langue anglaise ou française. On pourrait, par contre, se demander si cela serait le cas pour un enfant chinois qui n'a jamais entendu aucune chanson de ce type.

Dans le cas de l'ordinateur, on se place dans une perspective similaire à celle de l'enfant chinois: il faut arriver à apprendre des critères pour la classification sans aucun type de connaissance préalable.

Tout d'abord on peut penser à construire une base d'apprentissage avec quelques chansons de « The Beatles » et de Brassens dont on supposera connues les étiquettes correspondantes.

On va lire ces chansons avec Matlab, et on va obtenir des vecteurs très longs avec des valeurs pour la forme d'onde correspondant à chaque chanson. Il semble évident de couper ces vecteurs dans des morceaux plus petits correspondant à des *frames* temporels, de telle sorte que, pour chaque chanson, on obtiendra tout un ensemble de vecteurs de forme d'onde.

On pourrait penser à faire notre classification sur ces vecteurs directement, mais il ne semble pas que cette option soit très performante. C'est pourquoi, on cherchera à trouver une méthode pour extraire des caractéristiques ou « features » à partir des de ces valeurs pour un certain intervalle temporel, jusqu'à obtenir des vecteurs de caractéristiques appropriés pour la classification avec des svm.

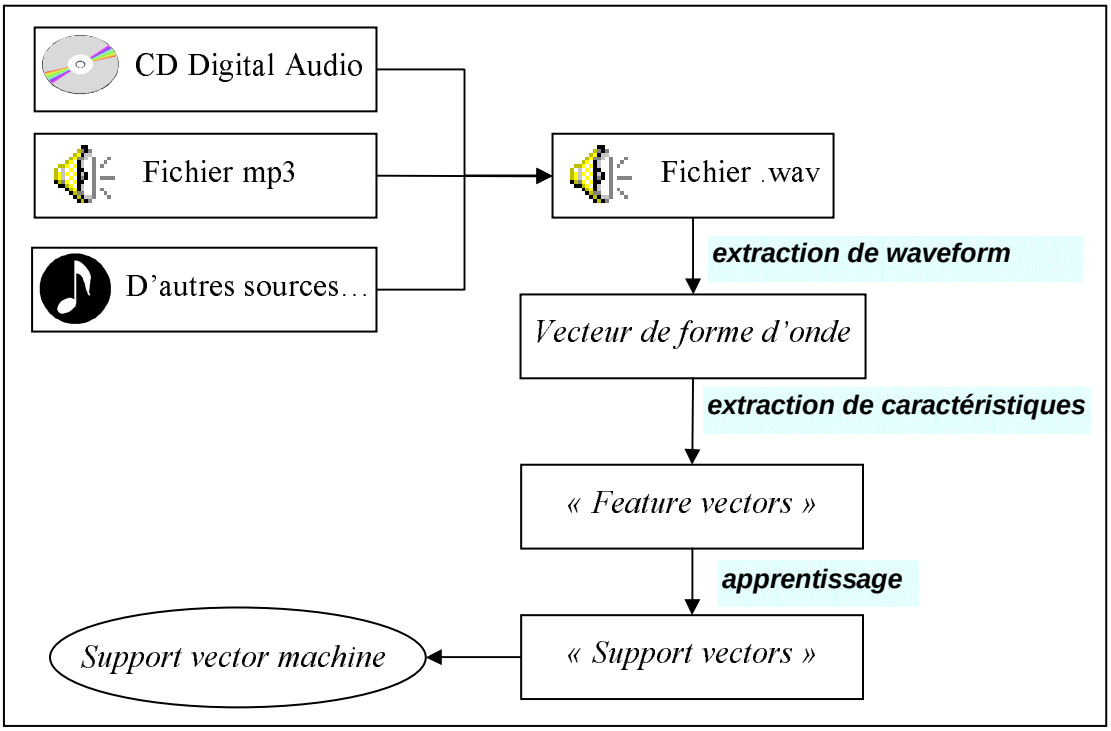
Une fois que l'on ait obtenu ces vecteurs, on pourra penser à utiliser les chansons de la base d'apprentissage pour entraîner un svm concrète, et puis on testera les performances de ce machine sur une autre base de test avec des chansons différentes dont nous supposerons inconnues les étiquettes et qui nous permettront donc d'évaluer la capacité de généralisation. Le choix d'un noyau ou kernel approprié pour ce svm, ainsi que le choix d'une parametrisation adéquate associée constituera un point critique pour la réussite ou l'échec de l'approche adoptée.

Lorsque l'on aura finit l'apprentissage, on pourra utiliser le svm pour classifier des chansons en prenant des petits fragments et en classifiant les vecteurs de caractéristiques associés.

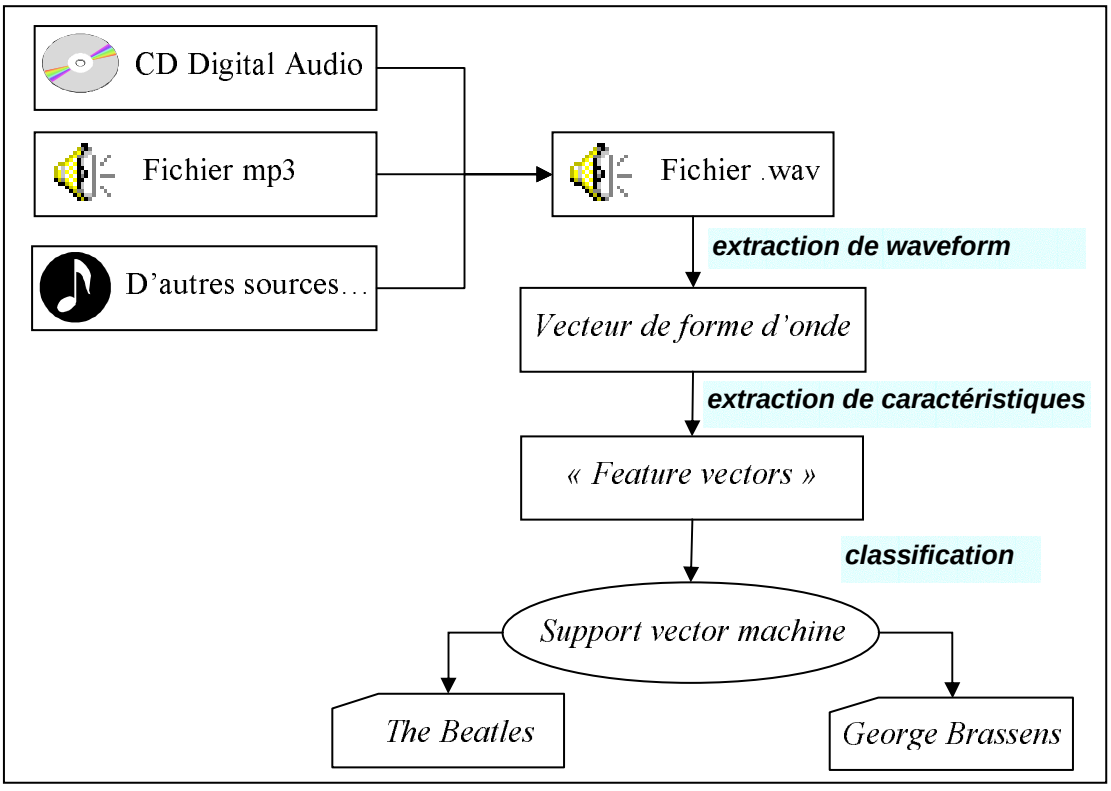
2.3 Schémas conceptuels pour l'apprentissage et la classification

Le processus d'apprentissage et classification que l'on vient de décrire peut être représenté schématiquement de la manière suivante :

Pour l'apprentissage



Pour la classification



3 L'extraction des caractéristiques: obtention des « *feature vectors* »

Comme on a déjà expliqué, le choix d'un ensemble pertinent de caractéristiques constituera un point critique qui aura une très grande influence sur les performances de notre classificateur.

On commence par prendre une source de musique, par exemple un fichier mp3 ou une piste audio d'un CD, et on le transforme au format .wav pour pouvoir le lire facilement à Matlab.

De cette manière, on arrive à avoir une représentation de la signal audio dans le domaine du temps, caractérisé par sa forme d'onde ou « waveform data » en anglais.

Une première approche, que l'on pourrait considérer « naïve », serait de choisir comme vecteurs de caractéristiques des intervalles temporelles de ce vecteur directement. Par contre, il semble que cela serait, en effet, une approche trop naïve car les vecteurs que l'on devrait considérer seraient trop longs pour des petits intervalles de temps (si l'on considère, par exemple, un fréquence d'échantillonnage de 44Khz, on aurait besoin de vecteurs de dimension 440 pour chaque 10ms., ce qui ne semble pas être une option très attractive). En plus, la variance temporelle serait peut-être excessive pour permettre d'arriver à des résultats satisfaisants.

Ceci dit, on peut faire un peu de la recherche pour connaître quelles sont les méthodes habituelles pour l'extraction de caractéristiques dans le domaine de la reconnaissance de formes appliqué aux signaux audio.

Apparemment, il semble qu'il existe tout un ensemble d'algorithmes issues des domaines tels que l'apprentissage statistique, la théorie de l'information et de la communication, le traitement de signal et même de celui de la psychologie et la linguistique, qui ont été développés au cours des années et qui permettent d'obtenir de représentations paramétriques des sons.

Entre les plus importants, on peut citer :

- L'analyse de puissance spectrale (FFT)
- L'analyse prédictive linéaire (LPC)
- La prédiction linéaire perceptuelle (PLP)
- L'analyse cepstral en escale de Mel (MEL)

Pour la réalisation de mon projet, j'ai décidé d'utiliser l'approche PLP, combinée avec une technique complémentaire appelée RASTA, donc je ferai une petite description technique dans le reste de cette sous-section.

Puisque l'objectif du projet n'est pas de faire une étude approfondie de ces autres types de techniques, on se limitera ici à citer leur existence. Le lecteur intéressé peut en avoir plus d'information sur la référence [1].

3.1 L'Analyse Prédictive Linéaire (LPC)

Au cours des dernières années, l'analyse basée sur des prédictions linéaires est devenu une des techniques le plus utilisées pour l'estimation des paramètres pour les signaux audio dans le domaine de la reconnaissance de voix.

L'idée principale sous-jacente est celle du fait qu'un fragment ou échantillon d'un discours à un moment donné peut être approximé par une combinaison linéaire d'échantillons passés. En minimisant la somme des différences quarrés (sur un intervalle fini) entre les échantillons actuels du discours et les valeurs prédises on arrive à obtenir un ensemble de paramètres unique, qui constitueront les coefficients utilisés pour l'analyse prédictive.

Avant d'être utilisés pour la reconnaissance, ces coefficients sont transformés dans un ensemble plus robuste de paramètres que l'on appelle coefficients « cepstra ».

En anglais, le mot « cepstrum » est une anagramme du mot « spectrum ». Le terme était introduit par Bogert et al. en 1963 pour designer le travail avec des données du domaine de la fréquence comme si elles étaient dans le domaine du temps [2]. Ils ont travaillé avec des représentations dans lesquelles ils utilisaient un axe x, qu'ils ont appelé de « quefrencies », prenant des secondes comme unités, mais indiquant des variations dans le spectre fréquentiel.

Du point de vue mathématique, pour obtenir les cepstrum on décompose le signal S_n et on obtient une excitation E_n et un filtre linéaire $H(e^{j\theta})$.

Dans le domaine de la fréquence, on arrive à une expression de la forme :

$$S(e^{i\theta}) = H(e^{i\theta})E(e^{i\theta})$$

Ensuite, on calcule le logarithme complexe de cette expression :

$$log(|S(e^{i\theta}|) = log(|H(e^{i\theta})|) + log(|E(e^{i\theta})|)$$

Et finalement, on obtient les cepstrum après calculer la transformée inverse, comme on observe dans le diagramme suivant :

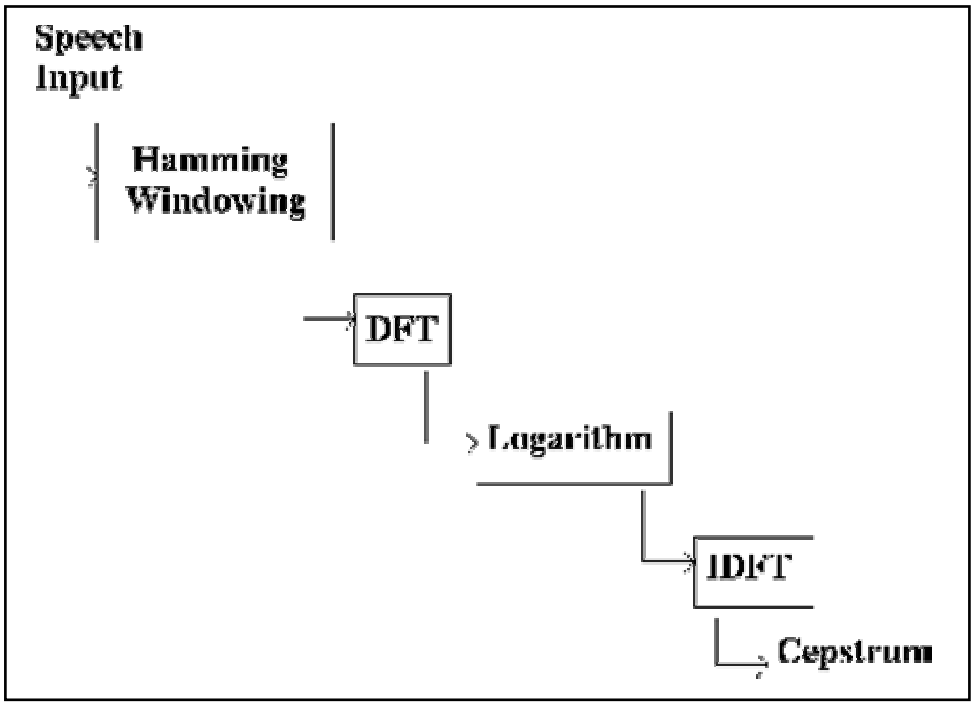


Schéma de bloques pour l'obtention des cepstrum [2]

3.2 La Prédiction Linéaire Perceptuelle (PLP) et le filtrage RASTA

La Prédiction Linéaire Perceptuelle, utilise cette même idée de base, puisqu'elle travaille avec le spectre à court terme du signal. Par contre la PLP modifie ce spectre à travers quelques transformations basées sur des idées psychophysiques et dont l'objectif finale est d'améliorer le processus de classification et de reconnaissance.

La PLP a été introduite en 1985 par Hermansky et al [3]. Il s'agit d'une technique qui essaie de profiter des propriétés psycho-acoustiques du système auditif humain pour optimiser le travail sur le spectre audible. Notamment, le PLP utilise ces trois propriétés :

- La résolution spectrale de la bande critique
- La courbe d'equalisation de volume (« *equal-loudness curve* »)
- La loi de puissance intensité volume (« *intensity-loudness power law* »)

Ces aspects rendent les techniques PLP plus proches de l'audition humaine que les techniques LPC, et donc elles permettent l'obtention de paramètres plus robustes.

Du point de vue mathématique, la méthode utilise le spectre en puissance à court terme P(w), pour lequel on calcule la convolution avec un schéma de masquage pour la bande critique simulée. Ensuite, on fait un re-échantillonnage de la bande critique selon des intervalles de l'escale de Bark. On fait une opération de pre-emphasis avec une courbe d'equalisation de volume préfixée et finalement on comprime le spectre résultant avec

une non-linearité de type racine cubique. Le modèle de pôles résultant est consistent avec les phénomènes observés pour la perception auditive chez l'humain [2].

Le schéma de bloques associé est celui de la figure ci-jointe.

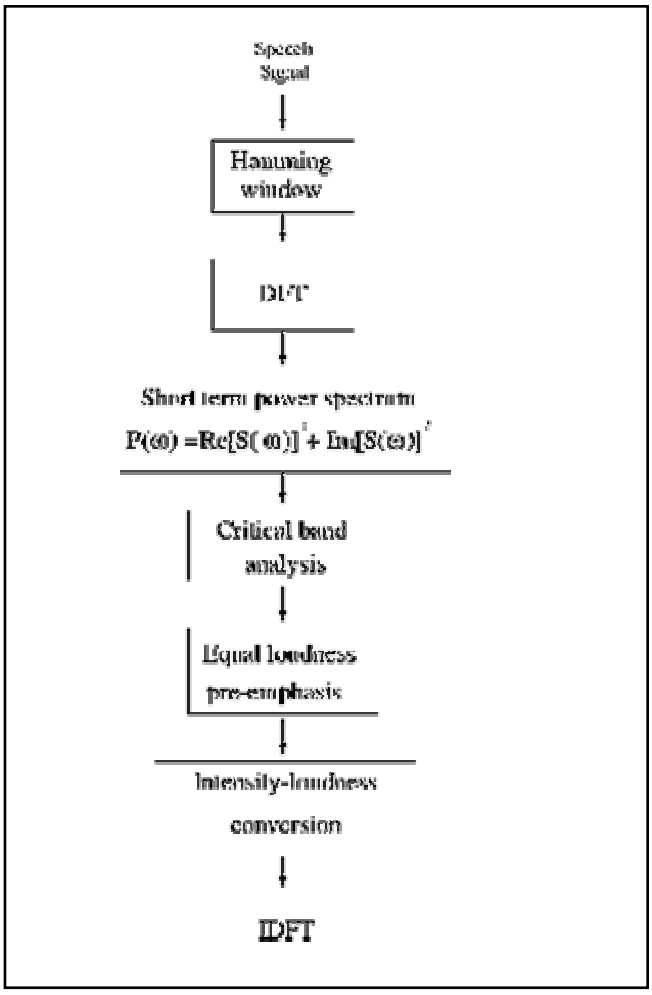


Schéma de bloques pour le PLP

Pour une description plus détaillée du point de vue mathématique des formules intervenant dans les diverses étapes de la méthode PLP, le lecteur peut consulter les références [2,3].

Le terme RASTA provient de l'anglais « *RelAtive SpectrAL Technique* ». Les méthodes basées sur le travail avec le spectre à court terme (comme le LPC et le PLP, que l'on vient de présenter) sont plutôt vulnérables quand les valeurs spectrales considérées sont modifiées par la réponse fréquentielle du canal de communication. Le RASTA est donc, un ensemble de techniques utilisées pour rendre les coefficients plus robustes face à ce type de facteurs de l'environnement qui peuvent être considérés stables ou que varient très lentement.

La méthode RASTA replace le spectre à court – terme en bande critique conventionnel de PLP par une estimation spectrale moins sensible à ce type de variations, selon un processus qui peut être décomposé en quatre étapes [2] :

1. Calcul du spectre de puissance en bande critique comme pour l'analyse PLP.
2. Transformation de l'amplitude spectrale avec une transformation non - linéaire comprimante et statique.
3. Filtrage de la trajectoire temporelle de chaque composant spectral transformé.
4. Transformation du fragment audio filtré avec une transformation non - linéaire expansive et statique.
5. Multiplication par la courbe de equalisation en volume et augmentation de la puissance 0.33 dB pour simuler la loi de puissance de l'audition.
6. Calcul d'un modèle « tous - pôles » du spectre résultant comme pour le PLP.

A nouveau, on peut observer le schéma de bloques associé dans la figure ci – jointe :

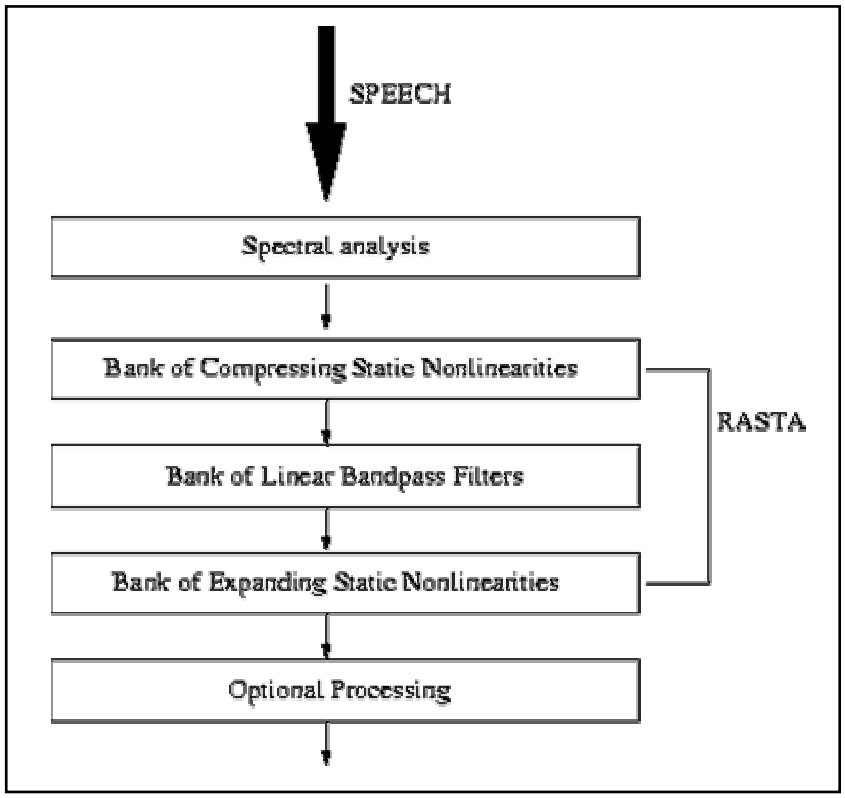


Schéma de bloques pour le RASTA-PLP [2]

Le choix des fréquences pour les filtres en bande intermédiaire utilisés détermine les composantes spectrales qui interviendront sur les paramètres finaux.

Les techniques PLP et RASTA nous permettront d'obtenir un vecteur de caractéristiques de dimension 13 pour chaque frame temporel considéré. Apparemment, dans le domaine de la reconnaissance de sons il est utile de travailler aussi avec les valeurs des dérivés de premier et deuxième ordre de ce vecteur, ce qui nous conduira à travailler finalement avec des vecteurs de dimension 39.

3.3 L'implémentation sur Matlab

En 1997, Mike Shire, un étudiant du groupe de Morgan (co-auteur de l'article original sur le RASTA, avec Hermansky [4]) à l'ICSI, a proposé une implémentation en Matlab pour les méthodes PLP-RASTA. La dernière version est téléchargeable en ligne, à partir du site de Dan Ellis à l'université de Columbia [5], qui en a fait quelques améliorations.

Bien que dans le site d'Ellis on puisse trouver tout un ensemble routines et fonctions Matlab reliées à des différents aspects de la codification prédictive tels que l'escale de Bark ou les algorithmes de Levison-Durbin, pour mon projet j'ai utilisé uniquement les fonctions rastaplp.m et deltas.m .

La fonction rastaplp.m permet d'obtenir les coefficients cepstra dont on a besoin, à partir des données de forme d'onde à Matlab, selon un schéma de la forme :

[cepstra, spectra] = rastaplp(samples, sr, dorasta, modelorder)

Les arguments d'entrée sont :

- samples : les données de forme d'onde. Par exemple, celles obtenues avec la commande wavread de Matlab.
- sr – taux d'échantillonnage (*sampling rate*)
- dorasta – paramètre permettant d'activer (1) ou désactiver (0) le filtrage RASTA.
- modelorder – paramètre permettant de choisir l'ordre pour le modèle PLP.

La sortie est composée de deux matrices contenant les coefficients cepstra et des coefficients spectraux respectivement. Dans les matrices, chaque colonne correspond à un frame temporel, tandis que chaque ligne correspond à une caractéristique ou feature.

La fonction delta.m permet d'obtenir les dérivés des coefficients cepstra dont on a parlé ci-dessus d'une manière directe et facile à paramétrer :

D = deltas(X,W)

Les arguments d'entrée sont :

- X : matrice contenant les coefficients cepstra ordonnés par lignes et colonnes selon le modèle décrit pour rastaplp.m .
- W : nombre de points de la fenêtre pour le calcul du dérivé selon une pente linéaire simple.

La sortie est une matrice de la même dimension que X.

Pour l'obtention des vecteurs avec les dérivés de deuxième ordre, il suffit d'utiliser la fonction deltas.m de manière réitérée.

3.4 Un exemple concret: la chanson « Let it be »

On va essayer de montrer un exemple concret pour les concepts que l'on vient d'expliquer à travers le cas de la chanson « *Let it be* » de « The Beatles ».

Imaginons que, tout au début, nous avons la chanson dans un fichier « letitibe.mp3 ».

Le premier pas consiste à transformer ce fichier dans un fichier .wav que puisse être lu à Matlab. Pour cela, on peut utiliser le logiciel freeware mpg123 :

```
[system command line prompt]> mpg123 –w letitbe.wav letitbe.mp3
```

A continuation, on peut utiliser la commande *wavread* de Matlab pour obtenir une matrice avec les données de forme d'onde associées à la chanson.

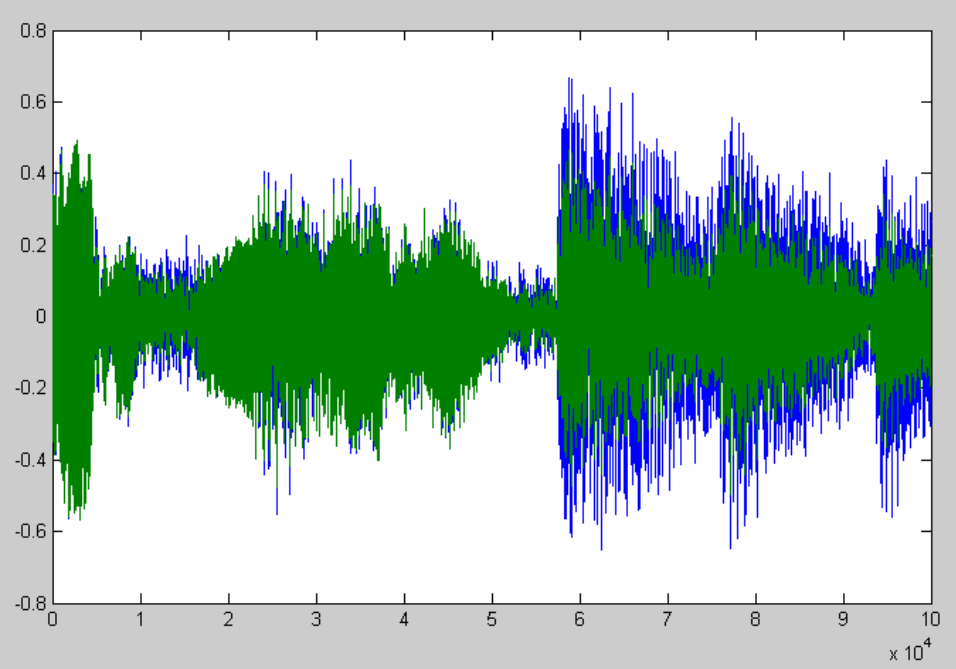
Pour qu'il soit plus facile de visualiser les résultats, on travaillera pour cet exemple avec un intervalle de 100.000 échantillons et un offset d'un million d'échantillons à partir du début de la chanson :

```
>> [d,sr,nbits] = wavread('letitbe',[1000001 1100000]);
```

Puisque la chanson est échantillonné à 44 khz, cet intervalle correspond à un peu plus de deux secondes de musique.

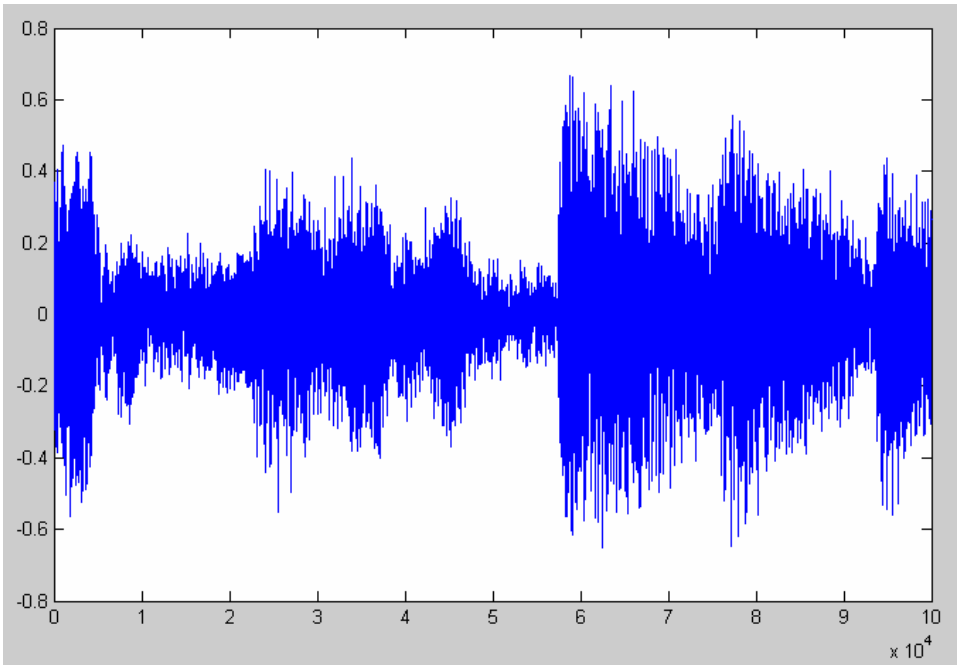
Voici la forme d'onde que l'on vient de lire à partir du fichier .wav :

```
>> plot(d)
```



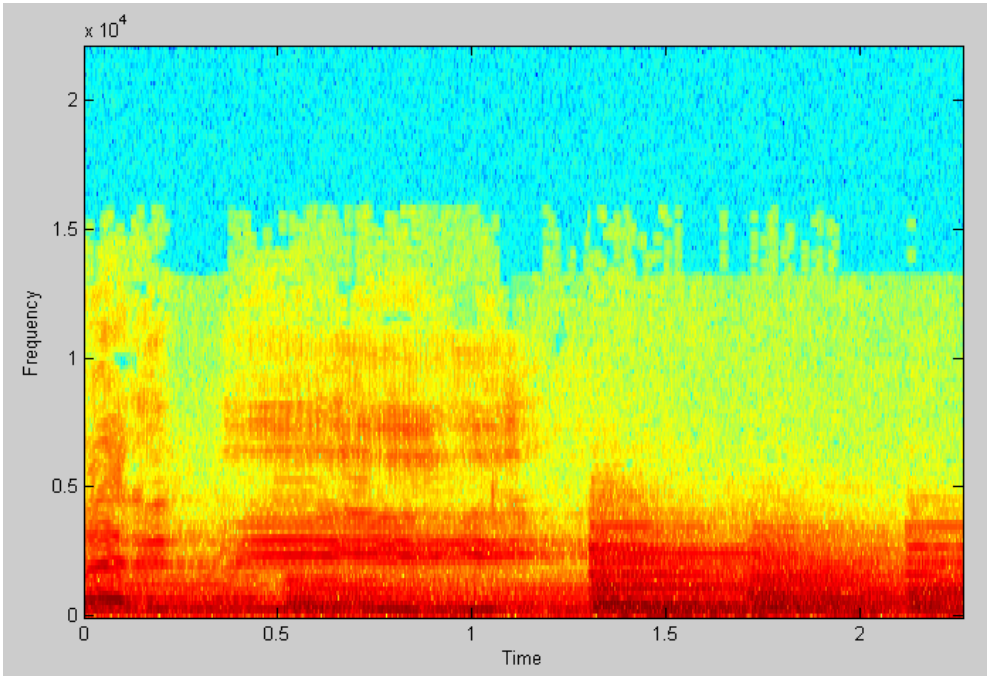
Puisque l'on travaille avec une chanson stéréo, on obtient une matrice ' *d* ' avec deux colonnes, qui correspondent aux deux canaux musicaux. Pour notre problème de reconnaissance, nous ignorerons le deuxième canal et nous travaillerons donc avec un signal mono :

```
>> plot(d( :,1))
```



Le spectrogramme du signal est de la forme :

```
>> specgram(d, 256, sr);
```



Si l'on veut, on peut écouter le signal à l'aide de la commande *sound* de Matlab :

```
>> sound(d( :,1),sr,nbits)
```

Sur ce signal, on peut déjà appliquer nos algorithmes PLP et RASTA.

Considérons, par exemple, une transformation avec PLP d'ordre 12 et filtrage RASTA désactivé :

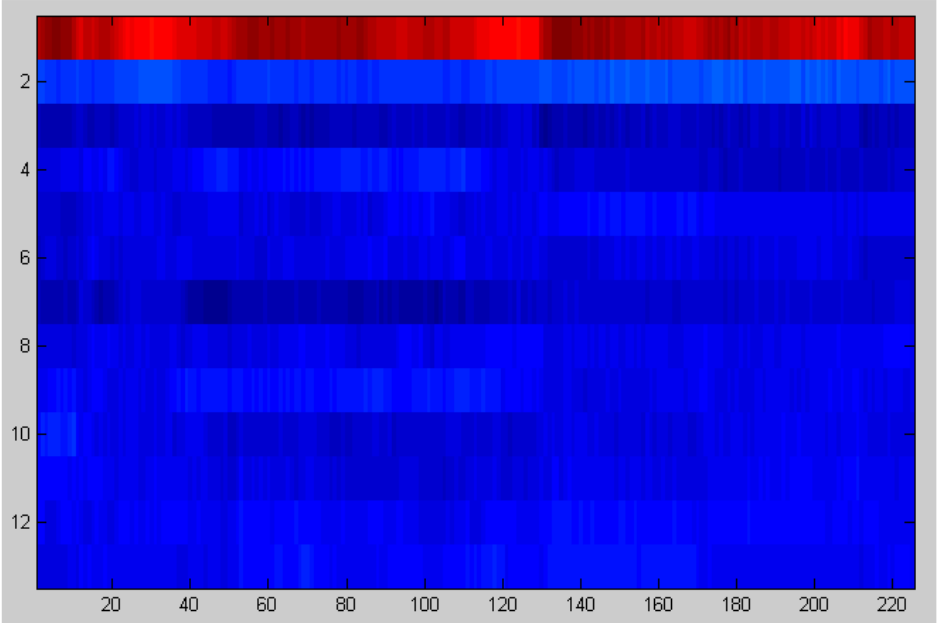
```
>> [cep, spec] = rastaplp(d(:,1), sr, 0, 12);
```

On observe que, pour l'intervalle temporel considéré, de 100.000 échantillons, on obtient 225 vecteurs de coefficients cepstra. En réalité, on peut considérer un seuil initial d'environ 1.500 échantillons avec un seul vecteur de coefficients, à partir duquel on obtient un nouveau vecteur pour chaque 440 échantillons (~10ms) de la forme d'onde.

```
>> size(cep)
ans =  13   225
```

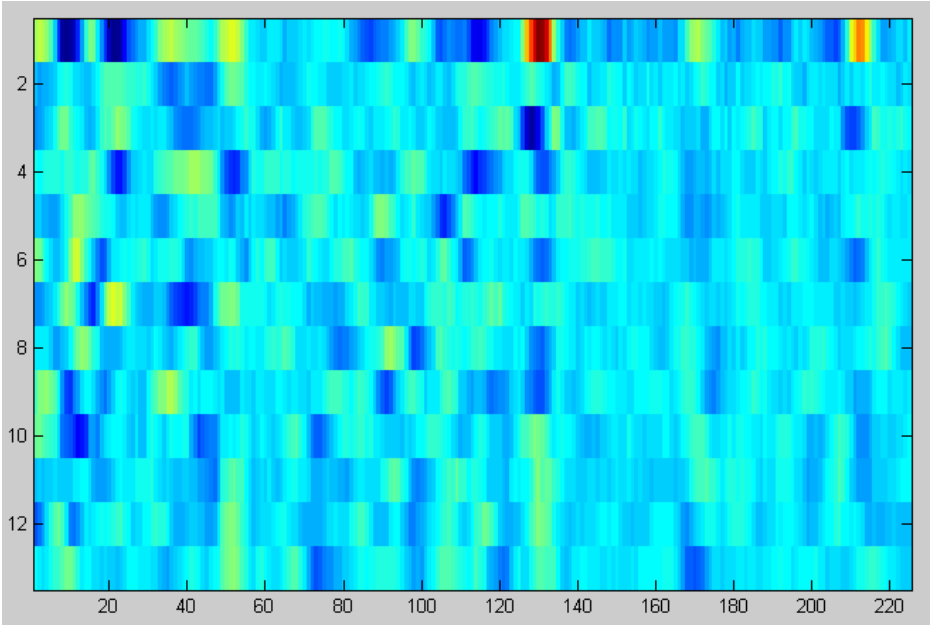
On peut visualiser les coefficients cepstra si l'on veut :

```
>> imagesc(cep)
```

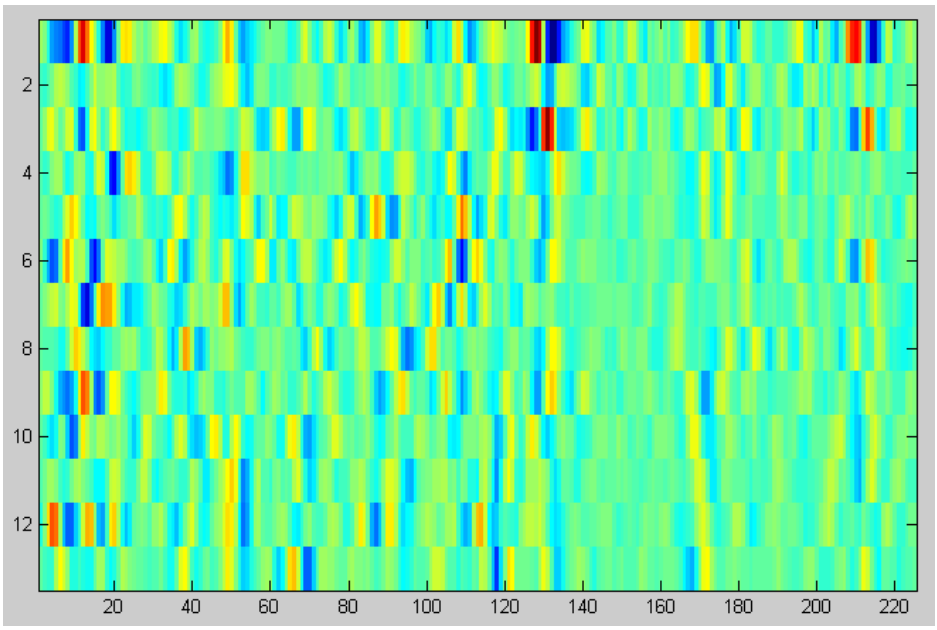


On peut aussi calculer et visualiser les vecteurs delta, avec les dérivés de premier et deuxième ordre des coefficients :

```
>> del=deltas(cep);
>> imagesc(del)
```



```
>> ddel = deltas(deltas(cep,5),5);
>> imagesc(ddel)
```

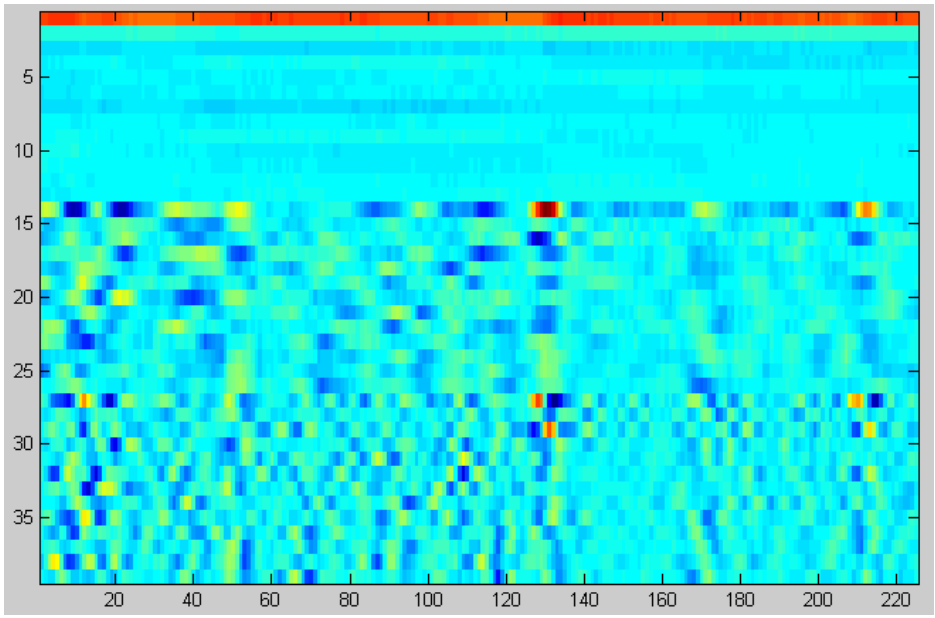


Finalement, on peut construire une matrice par la concaténation des trois types de vecteurs que l'on vient d'obtenir, qui donnera comme résultat des vecteurs de 39 composants qui seront ceux que l'on utilisera pour la classification des chansons :

```
>> M = [cep;del;ddel];
>> size(M)
ans =  39   225
```

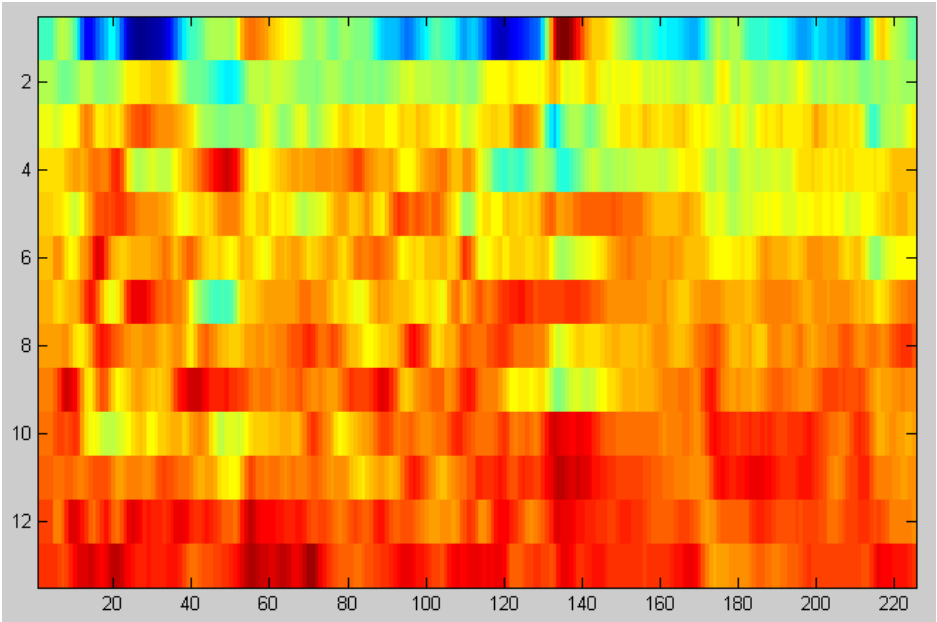
On peut aussi visualiser cette dernière matrice :

```
>> imagesc(M)
```

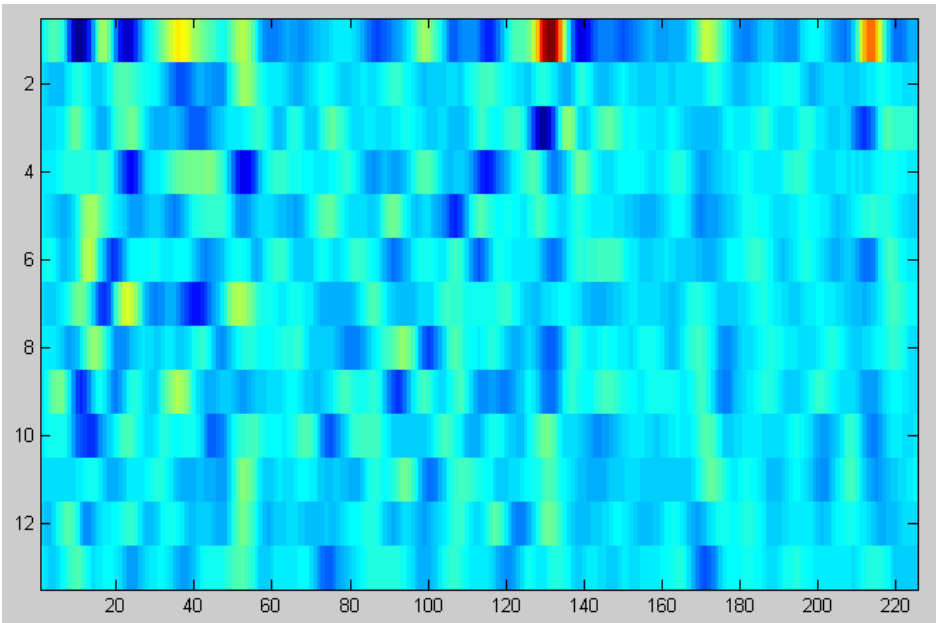


Pour avoir une idée de l'influence du filtrage RASTA, on peut répéter le même processus avec filtrage RASTA et comparer les résultats :

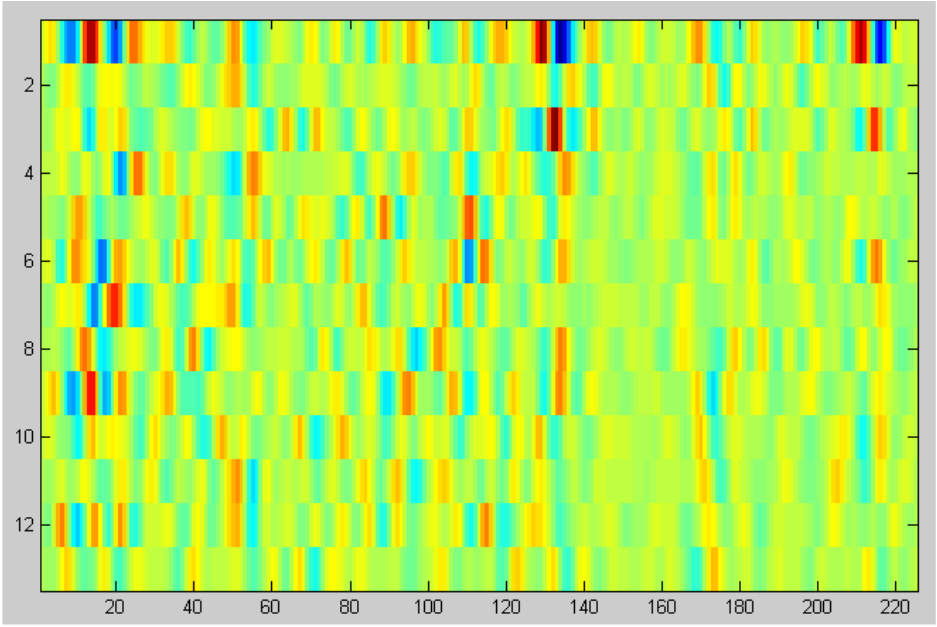
```
>> [cep2, spec2] = rastaplp(d(:,1), sr, 1, 12);
>> imagesc(cep2)
```




```
>> del2=deltas(cep2);
>> imagesc(del2)
```

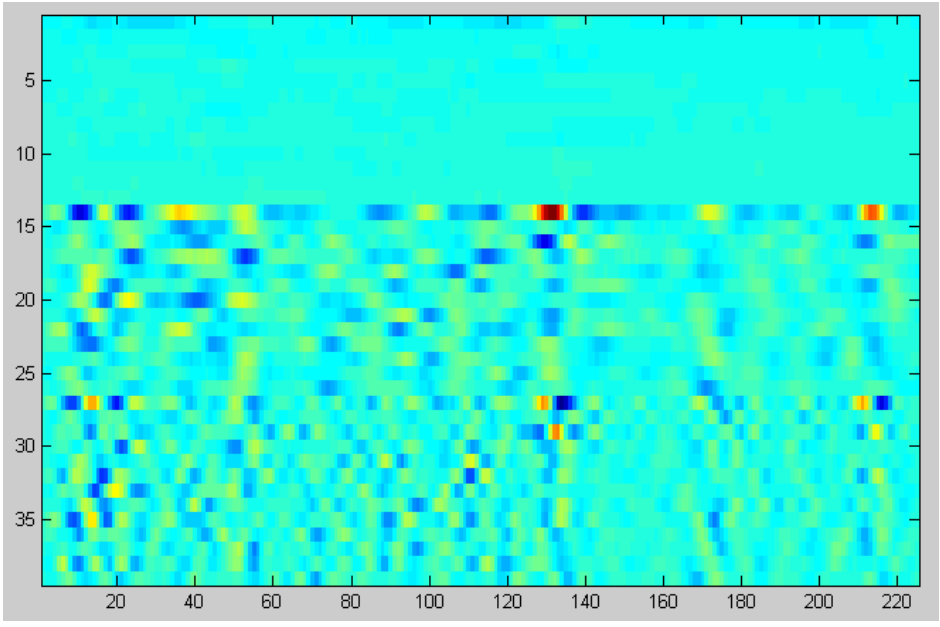


```
>> ddel2 = deltas(deltas(cep2,5),5);
>> imagesc(ddel2)
```



C'est ainsi que l'on arrive à une nouvelle matrice finale avec 225 vecteurs de caractéristiques à 39 composants :

```
>> M2 = [cep2;del2;ddel2];
>> size(M2)
ans =  39   225
>> imagesc(M2)
```



4 Les moteurs SVM

Une fois que j'ai été capable d'obtenir des vecteurs de *features* permettant de caractériser les chansons à classifier, je me suis intéressé aux différents moteurs svm freeware disponibles en ligne.

Initialement, j'ai décidé de travailler avec OSU-SVM, une interface pour Matlab du moteur LIBSVM. Après quelques simulations initiales, j'ai décidé d'utiliser aussi le moteur SMV-Light afin d'avoir une perspective plus riche sur les fonctionnalités offertes par chacun d'entre eux.

4.1 Le moteur LIBSVM et le front-end OSU-SVM pour Matlab

OSU-SVM est un toolbox pour Matlab développée par Junshui Ma, Yi Zhao et Stanley Ahalt à l'Ohio State University (d'où son nom) [6].

Le toolbox utilise les algorithmes du package LIBSVM de Chih-Jen Lin et Chih-Chung Chang [7]. Ces routines sont codifiées en C++ et le toolbox OSU-SVM s'en sert à travers des librairies MEX de Matlab, ce qui permet d'avoir une plus grande vitesse de calcul et une meilleure gestion de la mémoire. En plus, le toolbox est « self-contained », cet à dire, il ne faut pas utiliser des outils d'optimisation externes.

Les algorithmes mathématiques utilisés par le logiciel LIBSVM sont bien expliqués sur le site web [7] et plus concrètement sur la référence [8]. Il est aussi possible de télécharger une « guide pour débutants », avec des explications intuitives et très faciles à comprendre permettant de commencer rapidement à travailler avec le logiciel [9].

Apparemment, le package logiciel LIBSVM inclut des fonctions permettant d'utiliser les svm pour des problèmes de régression (SVR). On y trouve aussi l'implémentation de quelques algorithmes plus récents comme les v-SVM ou les « one-class SVM », bien que moi, je me suis limité au travail avec les C-SVM pour la classification.

En ce qui concerne le toolbox, elle est composée de quelques fonctions Matlab très faciles à utiliser, notamment :

- Des fonctions pour l'apprentissage svm. On y trouve une fonction spécifique pour chaque type de kernel :
  - LinearSVC.m : u'·v
  - PolySVC.m : $\gamma(u'\cdot v + w)^d$
  - RbfSVC.m) : $e^{-\gamma|u'-v|^2}$

  Elles prennent comme paramètres d'entrée une matrice avec les vecteurs de caractéristiques, un vecteur avec les étiquettes correspondantes, et quelques paramètres dont le plus importants sont :
  - C – qui permet de choisir le coût associé à des erreurs de classification (et qui permet, donc, de contrôler le « trade-off » entre erreurs de classification pendant l'apprentissage et marge de sécurité).
  - Paramètres spécifiques pour chaque type de kernel (d, γ et w).

- Une fonction pour la classification : SVMClass.m, qui prend comme paramètre d'entrée une matrice de vecteurs de caractéristiques et l'ensemble de paramètres caractérisant une svm. Elle donne comme sortie un vecteur d'étiquettes correspondant aux résultats de la classification.

- Une fonction pour le test : SVMTest.m, similaire à SVMClass, mais qui prend en plus comme entrée les bons étiquettes pour les données à classifier et qui calcule tout directement le taux de réussite et la matrice de confusion pour chaque processus de classification.

- Quelques fonctions pour le prétraitement des données, Normalize.m et Scale.m, pour normaliser les valeurs numériques des vecteurs de caractéristiques avant de les traiter avec le svm.

Pour avoir plus d'information concernant les paramètres d'entrée et de sortie de chacune de ces fonctions, ainsi que sur les autres fonctions du toolbox, le lecteur peut consulter la documentation de l'API disponible en ligne sur sourceforge [10].

Exemple d'utilisation

Imaginons que nous avons déjà une matrice « valeurs_ap » avec des vecteurs de caractéristiques comme ceux que l'on a obtenu dans la section précédente et un vecteur « etiq_ap » avec des étiquettes qui prennent la valeur 1 pour les chansons des Beatles et 2 pour les chansons de Brassens.

On peut faire l'apprentissage svm, avec un kernel RBF à l'aide de la commande RBFSvc :

```
>> [AlphaY, SVs, Bias, Parameters, nSV, nLabel] = RbfSVC(valeurs_ap, etiq_ap, Gamma,C);
```

Maintenant, on peut utiliser le svm que l'on vient de créer (défini par les variables à la sortie de l'expression ci-dessus) pour tester les résultats sur une base de test composée d'une nouvelle matrice de valeurs « valeurs_test » et des nouvelles étiquettes correspondantes :

```
>>[ClassRate, DecisionValue, Ns, ConfMatrix, PreLabels]=
                SVMTest(valeurs_test, etiq_test, AlphaY, SVs, Bias,Parameters, nSV, nLabel);
```

On peut aussi utiliser le svm pour faire des prédictions et classifier des nouveaux vecteurs de caractéristiques contenus, par exemple, dans une matrice « valeurs_generalisation » :

```
>> [Labels, DecisionValue]=
        SVMClass(valeurs_generalisation, AlphaY, SVs,Bias, Parameters, nSV, nLabel);
```

Cette dernière expression permet d'obtenir le vecteur d'étiquettes « Labels » correspondant aux vecteurs de caractéristiques « valeurs_generalisation ».

4.2 Le moteur SVM-Light

J'ai décidé d'utiliser SVM-Light après une conversation avec Matthieu Perrot, un autre étudiant du Mastère MVA, qui me l'a recommandé à cause de ses nombreux indicateurs pour la mesure de performances et pour l'estimation de la capacité de généralisation des svm générées.

SVM-Light [11] est un moteur svm développé par Thorsten Joachims (qui actuellement travaille à la Cornell University) quand il était dans le département d'Intelligence Artificielle de l'Université de Dortmund.

Le code est implémenté en C et est devenu un des moteurs les plus populaires et les plus utilisés actuellement. Il possède des nombreuses fonctionnalités et des caractéristiques très attractives pour l'utilisateur :

- Il permet d'utiliser des nombreux kernels prédéfinis, et même d'utiliser des kernels définis par l'utilisateur.

- Il implémente des algorithmes rapides pour l'optimisation en termes de temps de calcul et gestion de mémoire [12].

- Il fourni tout un ensemble de paramètres qui permettent d'évaluer les performances des svm après chaque processus d'apprentissage, et plus concrètement :
    - Il montre une estimation de la dimension de Vapnik-Chervonenkis.
    - Il calcule la précision sur la base d'apprentissage selon une procédure « *leave-one out* ».
    - Il calcule des estimations Xi-alpha pour le taux d'erreur, la précision, et le « recall ».

- Il permet d'aller au-delà de la classification pour aborder des problèmes de régression et classement (« *ranking* »). Il permet de travailler avec des svm « transductives» et il admet une variante appelée SVM-Struct pour les problèmes multi variés et structurés.

Les estimateurs des performances

Dans son article de 1999 « *Estimating the generalization Performance of an SVM Efficiently* » [13], Thorsten Joachims fait une étude sérieux et rigoureux des principaux outils permettant d'estimer les performances d'une svm.

Le premier de ces estimateurs que l'on peut considérer est celui du taux d'erreur. Il s'agit tout simplement de la proportion entre le nombre de échantillons bien classifiés et le nombre d'échantillons total.

On trouve aussi deux autres estimateurs issues du domaine de la classification de textes qui peuvent trouver des applications très intéressantes dans un cadre plus général : la précision et le « recall ».

Le paramètre appelé « précision » dans SVM-Light est une mesure de la probabilité de que, dans un contexte de classification avec deux classes possibles (d'étiquettes 1 et -1, par exemple), un élément qui ait été classifié dans la classe 1 ait été bien classifié (cet à dire, parmi tous les éléments qui ont été classifiés comme 1, on cherche la proportion d'éléments qui appartient vraiment à la classe 1) :

$$Prec(h) = \frac{\Pr(h(\vec{x}) = 1, y = 1)}{\Pr(h(\vec{x}) = 1, y = 1) + \Pr(h(\vec{x}) = 1, y = -1)}$$

Le paramètre appelé « recall » dans SVM-Light est une mesure de la probabilité de que, dans un contexte de classification avec deux classes possibles (d'étiquettes 1 et -1), un élément appartenant à la classe 1 ait été classifié dans cette classe (cet à dire, parmi tous les éléments qui devraient avoir été classifiés comme 1, on cherche la proportion d'éléments qui en effet ont été classifiés de cette manière) :

$$Rec(h) = \frac{\Pr(h(\vec{x}) = 1, y = 1)}{\Pr(h(\vec{x}) = 1, y = 1) + \Pr(h(\vec{x}) = -1, y = 1)}$$

Comme on a vu pendant le cours de M. Azencott pour le mastère MVA, la dimension de Vapnik-Chervonenkis est très utile pour établir une borne supérieure pour la différence entre l'erreur ou risque asymptotique et le risque empirique.

Cette borne, trouvé par Vapnik et Chervonenkis et 1979 [13], est telle que pour un espace $h_{\mathcal{L}}$ de dimension VC d_H, on peut affirmer avec probabilité 1-η :

$$Err^n(h_{\mathcal{L}}) \leq Err^n_{emp}(h_{\mathcal{L}}) + 2\sqrt{\frac{d_{\mathcal{H}}(ln\frac{2n}{d_{\mathcal{H}}} + 1) - ln\frac{\eta}{4}}{n}}$$

Joachims utilise ces idées dans son logiciel pour calculer trois estimateurs bornés, qu'il appelle estimateurs Xi-alpha ($\xi\alpha$) pour le taux d'erreur, la précision et le *recall* :

- Pour l'erreur, il utilise aussi l'idée de la validation croisée (« *cross-validation* ») et plus concrètement celle du « *Leave-One-Out* »(LOO). De cette manière, chaque fois que l'on fait un apprentissage avec SVM-Light, le logiciel calcule l'erreur sur la base d'apprentissage en utilisant tous les éléments de cette base sauf un pour créer une sous - base avec laquelle arriver à avoir une prédiction de la valeur de l'élément laissé de cote. Après parcourir tous les éléments de la base d'apprentissage en laissant chacun d'entre eux de coté une fois, on arrive à obtenir une estimation adéquate des capacités de prédiction sur les éléments de la base de test. Cette estimation est combinée avec les idées autours de la borne supérieure de Vapnik-Chervonenkis pour offrir une estimation de la borne supérieure finale pour l'erreur après chaque processus d'apprentissage (ce qu'il appelle « *Xi-alpha estimate of the error* »).

- Pour la précision et le recall, on trouve aussi des estimations avec des bornes inférieures (« *Xi-alpha estimate of the precision* » et « *Xi-alpha estimate of the recall* ») après chaque apprentissage.

Pour une description plus minutieuse et détaillée des différents algorithmes, théorèmes et concepts mathématiques associés à ces indicateurs, le lecteur intéressé pourra consulter l'article original de Joachims [13].

Exemple d'utilisation

En réalité, le logiciel est formé par un seul couple de fichiers : svm_learn et svm_classify. Il est possible de télécharger des versions qui marchent sur les systèmes opératifs les plus usuels : Windows, Linux, Solaris, etc…

Les différentes options et paramètres d'entrée pour ces deux programmes sont décrites sur le site web de Joachims [11]. On doit leur fournir les donnés d'entrée pour l'apprentissage et la classification dans des fichiers selon une syntaxe spéciale expliquée aussi sur le même site. Le svm résultant d'un processus d'apprentissage, est enregistré sur un fichier et peut ainsi être utilisé ultérieurement pour des taches de test et classification.

Imaginons, par exemple, que nous avons un fichier « learning_data » avec les données correspondants aux vecteurs de *features* et ses étiquettes associées. On veut appeler « my_svm » le fichier qui contiendra le svm résultant du processus d'apprentissage.
Si l'on veut utiliser un kernel RBF avec gamma=0.1 et C=10, il suffira, donc, d'écrire :

```
[system prompt]> svm_learn -t 2 -c 1 -g 0.1 learning_data my_svm
```

Et on obtiendra un feedback par écran de la forme :

```
Scanning examples...done
Reading examples into memory...100..200..[]..22600..OK. (22640
examples read)
Optimizing..... Checking optimality of inactive variables...done.
Number of inactive variables = 8045
done. (9109 iterations)
Optimization finished (67 misclassified, maxdiff=0.00099).
Runtime in cpu-seconds: -2041.15
Number of SV: 16787 (including 3043 at upper bound)
L1 loss: loss=727.53502
Norm of weight vector: |w|=98.56306
Norm of longest example vector: |x|=1.00000
Estimated VCdim of classifier: VCdim<=19430.35177
Computing XiAlpha-estimates...done
Runtime for XiAlpha-estimates in cpu-seconds: 0.06
XiAlpha-estimate of the error: error<=50.87% (rho=1.00,depth=0)
XiAlpha-estimate of the recall: recall=>46.49% (rho=1.00,depth=0)
XiAlpha-estimate of the precision: precision=>49.08%
(rho=1.00,depth=0)
Number of kernel evaluations: 1099093750
Writing model file...done
```

Après, on peut vouloir utiliser le svm généré pour tester les performances de généralisation sur une base de test contenue, par exemple, dans le fichier « data_test » :

```
[system prompt]> svm_classify data_test my_svm
```

Et on obtient un retour de la forme :

```
Reading model...OK. (16787 support vectors read)
Classifying test examples..100..200..[]..13500..done
Runtime (without IO) in cpu-seconds: 419.95
Accuracy on test set: 74.29% (10074 correct, 3486 incorrect, 13560
total)
Precision/recall on test set: 67.23%/94.79%
```

Pour automatiser la création des fichiers avec les bases de test et d'apprentissage, j'ai utilisé une interface de SVM-Light pour Matlab, développée par Anton Schwaighofer et téléchargeable depuis son site web [14].

En réalité, il propose plusieurs fonctions Matlab dont j'ai utilisé uniquement swmlwrite.m et swmlread.m pour transformer des matrices Matlab en fichiers de données avec la syntaxe adéquate pour SVM-Light et vice-versa.

Pour construire, par exemple, un fichier de nom « dataset » lisible par SVM-Light avec une matrice Matlab de vecteurs de caractéristiques « valeurs » et ses étiquettes correspondantes « etiq », il suffit d'écrire :

```
>> svmlwrite('dataset',valeurs,etiq);
```

On peut également récupérer les valeurs des étiquettes résultantes d'une classification et enregistrées sur un fichier SVM-Light avec la fonction Matlab svmlread :

```
>> [etiq] = svmlread('resultats_classif');
```

5 Simulations et résultats

5.1 OSU-SVM et LIBSVM

Pour les premières simulations que j'ai réalisé pour ce projet, j'ai utilisé LIBSVM à travers son interface OSU-SVM à Matlab.

J'ai construit une base d'apprentissage avec des chansons de « The Beatles » et de George Brassens pour lesquelles j'ai obtenu des ensembles de vecteurs de caractéristiques selon les procédures expliqués dans les sections précédentes.

Les auteurs de la guide d'utilisation de LIBSVM [9] recommandent, en général d'utiliser un kernel de type RBF et de faire varier les paramètres C et gamma pour observer les variations associées dans les performances de classification.

En principe, on obtiendra les vecteurs de caractéristiques selon une approche PLP d'ordre 12 et sans filtrage RASTA. Dans la sous-section suivante dédiée à SVM-Light on fera quelques comparaisons des résultats avec et sans RASTA, mais pour le moment on utilisera uniquement PLP.

Puisque les temps de calcul associés à l'utilisation de grandes bases d'apprentissage sont assez longs, j'ai décidé d'utiliser de bases plus petites au début pour avoir une idée plutôt intuitive de l'évolution des performances avec la variation des deux paramètres en question.

C'est pourquoi j'ai commencé par travailler avec une base de cinq chansons pour chacune des deux classes, choisies au hasard :

Base d'apprentissage	
George Brassens	The Beatles
- *La mauvaise réputation* - *Quand les cons sont braves* - *Méchante avec des jolis seins* - *Le progrès* - *Les copains d'abord*	- *Yellow Submarine* - *Yesterday* - *Help* - *All you need is love* - *Let it be*

Pour chacune de ces chansons, j'ai considéré un intervalle de 106 échantillons sur la forme d'onde (intervalles du type [1000001 2000000]), ce qui correspond à un peu plus de 20 secondes de musique puisqu'elles étaient échantillonnées à 44 Khz.

En termes de vecteurs de caractéristiques, ceci impliquait 2266 vecteurs par chanson et donc un total de 22660 vecteurs en total. Les simulations ont montré que le fait de normaliser les coefficients des vecteurs de caracterisitiques n'apporte pas des avantages significatives pour la classification, et donc j'ai travaillé avec les vecteurs sans normaliser.

Comme base de test, j'ai utilisé 15 chansons différentes pour chaque classe, sur lesquelles j'ai pris des intervalles de 400000 échantillons (du type [1000001 1400000], ce qui implique 905 vecteurs de caractéristiques par chanson). Plus concrètement, les chansons choisies pour cette base de test ont été :

Base de test	
George Brassens	The Beatles
- *Le gorille*	- *Pennylane*
- *Entre l'Espagne et l'Italie*	- *She loves you*
- *La cane de Jeanne*	- *She's got a ticket to ride*
- *J'ai rendez-vous avec vous*	- *Paperback Writer*
- *Le testament*	- *Strawberry*
- *Saturne*	- *Revolution*
- *Les sabots d'Helene*	- *Please, please me*
- *Il n'y a pas d'amour hereux*	- *Obladi, oblada*
- *Le parapluie*	- *Love me do*
- *L'orage*	- *Hey, Jude !*
- *La mauvaise herbe*	- *A hard day's night*
- *Le bistrot*	- *Get back*
- *Chanson pour l'auvergnat*	- *Eleanor Rigby*
- *Les amoureux des bancs publics*	- *Can't buy me love*
- *Le 22 septembre*	- *Lady Madonna*

Pour chaque chanson, je calculais le pourcentage des vecteurs de caractéristiques classés dans la classe « Brassens » et le pourcentage « Beatles ».

Cet ainsi que j'ai pu rapidement constater que, pour des valeurs petites de C et pour des valeurs grandes de gamma, les résultats étaient assez mauvais. Le pourcentage global de réussite se trouvait toujours au-dessus du 90% pour Brassens, même pour les chansons de « The Beatles », ce qui veut dire que presque tous les vecteurs de caractéristiques étaient classés dans le même groupe pour chaque chanson, indépendamment de l'auteur. Pour ces valeurs des paramètres on n'arrivait pas à séparer les deux classes d'une manière efficace.

Par contre, j'ai observé que, en prenant des valeurs plus grandes pour C et plus petites pour gamma, les résultats amélioraient. En plus, le temps de calcul pour l'apprentissage devenait de plus en plus court et le nombre total de vecteurs de support associé aux svm correspondants décroissait aussi.

Puisque les calculs étaient très lents, j'ai décidé de considérer des paires de paramètres C et gamma avec des C croissantes et des gammas decroissantes simultanément, ce qui a donné des résultats en terme de taux de réussite moyen de la forme :

	$C= 2^1$ $\gamma = 2^{-1}$	$C= 2^2$ $\gamma = 2^{-2}$	$C= 2^3$ $\gamma = 2^{-3}$	$C= 2^4$ $\gamma = 2^{-4}$	$C= 2^5$ $\gamma = 2^{-5}$	$C= 2^6$ $\gamma = 2^{-6}$	$C= 2^7$ $\gamma = 2^{-7}$	$C= 2^8$ $\gamma = 2^{-8}$	$C= 2^9$ $\Gamma = 2^{-9}$	$C= 2^{10}$ $\gamma = 2^{-10}$
%	57,06	59,41	62,01	64,67	67,11	69,09	70,70	71,90	72,85	73,65

En effet, on constate que ce sont les valeurs élevées pour C et petites pour gamma qui donnent les meilleurs résultats.

Je suis conscient du fait que cette approche n'est pas la plus adéquate pour étudier l'évolution des performances en fonction des différents paramètres, et qu'il serait préférable de travailler, au moins, avec une grille bidimensionnelle dans laquelle toutes les combinaisons possibles entre des valeurs de C et gamma pourraient être testés. Cela reste comme un possible amélioration pour le futur. Dans la pratique, moi, j'avais à ma disposition un seul ordinateur avec licence Matlab sur lequel travailler et les longs temps associés à ces simulations m'ont empêché de calculer tous les résultats pour cette grille hypothétique. De toute façon, j'ai calculé une grille de ce type pour le moteur SVM-Light comme on verra dans la section suivante, et donc, les conclusions peuvent être facilement extrapolées au cas de OSU-SVM et LIBSVM.

Ceci dit, on peut, vouloir étudier plus en détail les resultats pour le cas $C=2^{10}$ et $gamma=2^{-10}$, puisque cela semble être le choix de paramètres le plus performante parmi toutes celles évaluées.

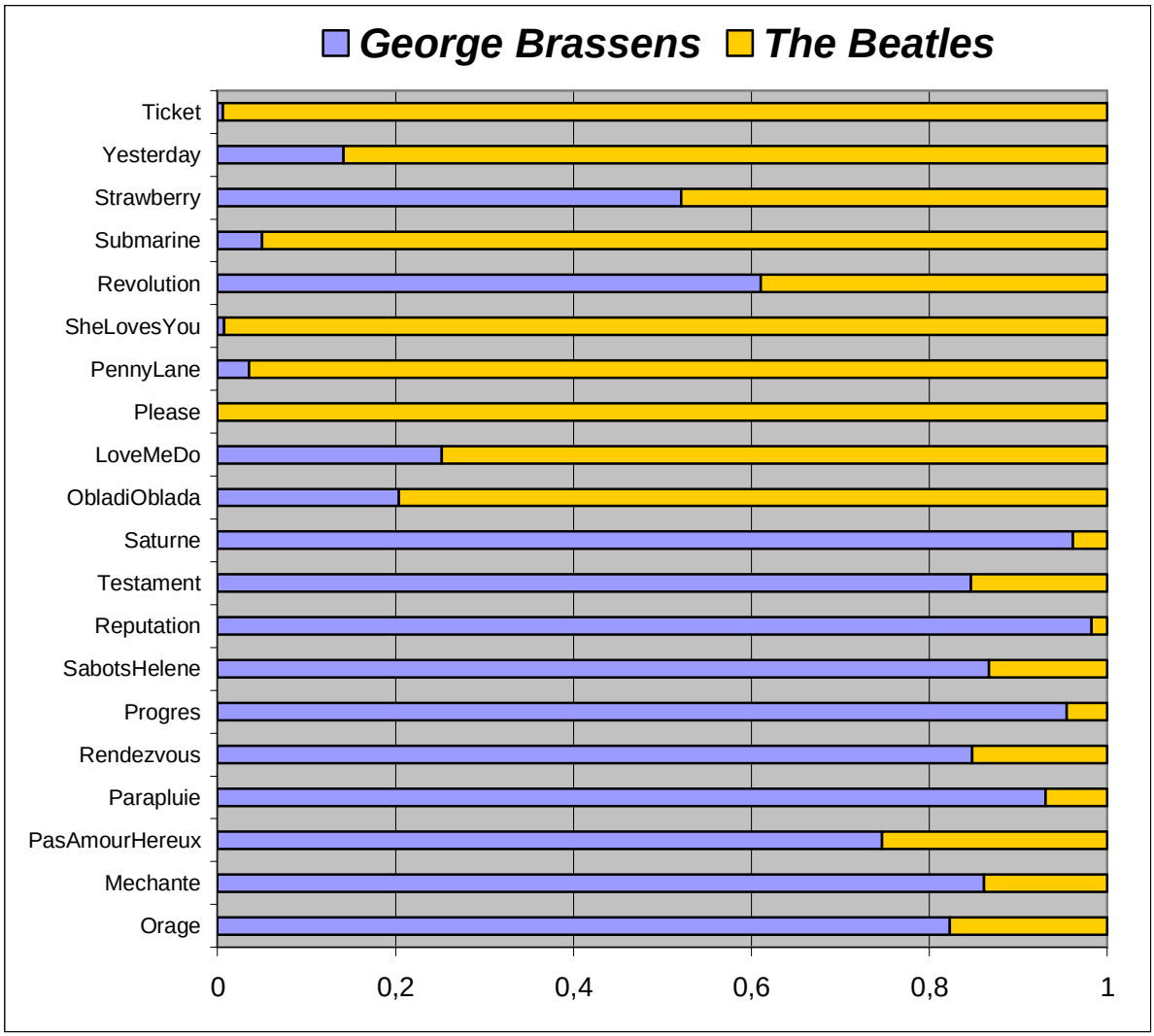
Pour aborder cette étude, on va considérer une nouvelle base d'apprentissage avec 10 chansons, choisies à nouveau au hasard, de la forme :

Base d'apprentissage	
George Brassens	The Beatles
- *L'Orage* - *Méchante avec des jolis seins* - *Il n'y a pas d'amour heureux* - *Le parapluie* - *J'ai rendez-vous avec vous* - *Le progrès* - *Les sabots d'Hélène* - *La mauvaise réputation* - *Le testament* - *Saturne*	- *Obladi Oblada* - *Love me do* - *Please, please me* - *Penny Lane* - *She loves you* - *Revolution* - *Yellow submarine* - *Strawberry Fields* - *Yesterday* - *She's got a ticket to ride*

Afin d'éviter que les temps d'apprentissage croissent énormément, on pendra des intervalles plus petits pour chaque chanson. Puisque l'on a doublé le nombre de chansons, on peut, par exemple, penser à prendre des intervalles avec une demi-longueur par rapport à ceux de l'exemple précédant. Cet ainsi que l'on considérera des intervalles de 500.000 échantillons, du type [1000001 15000000] sur la forme d'onde (ce qui générera des 1133 vecteurs de caractéristiques pour chaque chanson et un total de 22660 vecteurs à nouveau).

On essaiera d'estimer les capacités de généralisation de notre classificateur, en utilisant une procédure de validation croisée *« v-fold cross validation »* avec v=5, c'est-à-dire, on fera 5 paquets de 2 chansons dans l'ensemble de 10 chansons constituant la base d'apprentissage et on classifiera chacun de ces paquets avec un classificateur ayant appris sur la base des autres quatre paquets pour chaque cas.

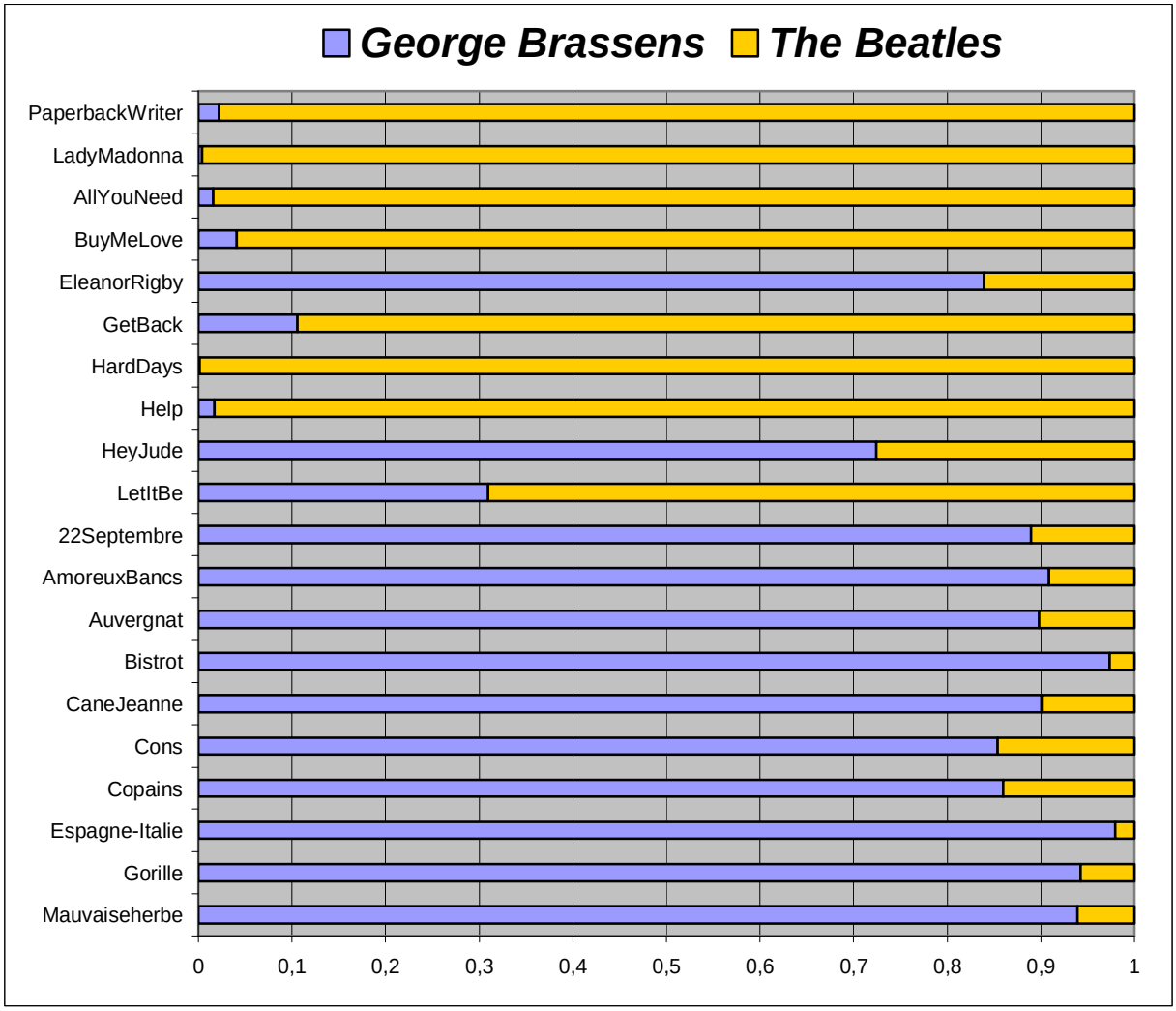
Le taux de réussite global estimé par cette procédure est du 84,97% et les résultats concrets pour chaque chanson se montrent dans la figure ci-jointe.



Si, maintenant, on veut utiliser toutes les 20 chansons de la base d'apprentissage ensemble pour créer un nouvel classificateur, on peut évaluer ses performances réelles de généralisation sur une vraie base de test composée de chansons différentes :

Base de test	
George Brassens	The Beatles
- *Mauvaise Herbe* - *Le gorille* - *Entre l'Espagne et l'Italie* - *Les copains d'abord* - *Quand les cons sont braves* - *La cane de Jeanne* - *Le bistrot* - *Chanson pour l'auvergnat* - *Les amoureux des bancs publics* - *Le 22 septembre*	- *Let it be* - *Hey, Jude !* - *Help !* - *It's been a hard day's night* - *Get back* - *Eleanor Rigby* - *Can't buy me love* - *All you need is love* - *Lady Madonna* - *Paperback writer*

On obtient ainsi un taux de réussite global de 85,26 %, et des résultats concrets pour chaque chanson de la forme:



On observe que la valeur estimée (84,97%) et la valeur réelle (85,26%) pour le taux de réussite global sont très proches. Cela semble indiquer que les estimations de performances calculées sur la base d'apprentissage suivant l'approche de la validation croisée avaient été bien faites.

5.2 SVM-Light

Après avoir travaillé avec OSU-SVM et LIBSVM, j'ai décidé de faire aussi quelques simulations sur SVM-Light.

J'ai utilisé tout d'abord les fonctions de l'interface sur Matlab dont on a parlé dans la section précédente pour transformer la même base d'apprentissage et la même base de test que l'on vient de voir pour le dernier exemple (avec contenant les mêmes deux ensembles de 20 chansons qui se montrent dans les tables ci-dessus) dans un fichier de données lisibles par SVM-Light.

Une fois que j'ai construit ces deux fichiers de données, j'ai utilisé directement les commandes svm_learn et svm_classify et j'ai regardé les résultats concernant les différents indicateurs de performances.

Il faut dire que, par opposition au cas de OSU-SVM/LIBSVM dans lequel j'avais à ma disposition un seul ordinateur avec une licence Matlab, le fait de pouvoir exécuter directement le logiciel SVM-Light sur des machines Solaris m'a permit de me connecter simultanément à plusieurs d'entre elles à travers le réseau de l'ENST Paris, grâce à l'application telnet. Cet ainsi que j'ai pu faire beaucoup plus de simulations jusqu'à obtenir plusieurs grilles caractérisant les paramètres d'intérêt, que je montrerai ci-dessous.

Afin de continuer dans la ligne adoptée pour OSU-SVM/LIBSVM, j'ai décidé de travailler avec des kernels RBF, pour lesquels j'ai fait varier les paramètres C et gamma. On peut considérer que ces paramètres ont une signification similaire dans les deux moteurs svm, même si les algorithmes utilisés dans leurs implémentations ne sont pas tout à fait identiques.

En ce qui concerne l'obtention des vecteurs de caractéristiques, on commencera par travailler, à nouveau, avec un modèle PLP d'ordre 12 sans filtrage RASTA.

La première idée plutôt intuitive pour évaluer la variation des performances avec ces deux paramètres a été de construire la table ci-jointe, en parcourant les valeurs 0.1, 1 et 10 pour C et gamma.

Pour chaque paire de valeurs C et gamma, on y trouve des estimations et indicateurs de performance selon le schéma suivant:

	C	
γ	Nombre de vecteurs de support Estimation Xi-alpha de l'erreur Estimation Xi-alpha de la précision Estimation Xi-alpha du recall	Dimension de Vapnik-Chervonenkis Réussite mesurée sur la base de test Précision mesurée sur la base de test Recall mesuré sur la base de test

Et voici la table en question :

	0.1		1		10	
0.1	18587 ≤30,96% ≥94,49% ≥62,62%	≤ 1083 65,38% 59,55% 95,86%	16787 ≤50,87% ≥46,49% ≥49,08%	≤ 19430 74,29% 67,23% 94,79%	16808 ≤52,11% ≥43,90% ≥47,71%	≤ 23783 74,87% 67,77% 94,84%
1	22640 ≤100% ≥0.00% ≥0.00%	≤ 457 50,01% 100% 0,03%	22640 ≤99.98% ≥0.04% ≥0.04%	≤ 44942 52,41% 72,93% 7,67%	22640 ≤99.98% ≥0.04% ≥0.04%	≤ 44951 52,48% 72,70% 7,94%
10	22640 ≤100% ≥0.00% ≥0.00%	≤ 453 50% NaN 0.00%	22640 ≤100% ≥0.00% ≥0.00%	≤ 45278 54,97% 62% 100%	22640 ≤100% ≥0.00% ≥0.00%	≤ 45281 54,97% 52,62% 100%

On observe que, pour des valeurs petites de C et pour des valeurs grandes de gamma, le svm n'arrive pas à apprendre ; on obtient des résultats très mauvais, comparables à ceux associés à un choix au hasard ou bien au fait de choisir toujours la même classe.

Il semble donc, logique, d'élargir cette première grille initiale dans la direction des valeurs grandes pour C et petites pour gamma, jusqu'à arriver à une nouvelle table de la forme :

	1		10		100	
0.05	12741	12325	11834	24597	11580	28843
	≤29,94% ≥74,11% ≥68,55%	79,88% 73,46% 93,57%	≤28,42% ≥75,06% ≥70,18%	80,43% 74,17% 93,36%	≤28,11% ≥75,40% ≥70,46%	80,32% 74,03% 93,30%
0.01	7401	5662	5159	35005	3978	217625
	≤29,41% ≥70,69% ≥70,55%	82,97% 77,95% 91,95%	≤21,30% ≥78,22% ≥78,11%	84,51% 80,07% 91,90%	≤16,74% ≥82,69% ≥83,63%	83,99% 80,00% 90,63%
0.005	7634	5108	5085	30616	3961	242607
	≤32,41% ≥67,65% ≥67,57%	83% 78,11% 91,81%	≤21,99% ≥77,84% ≥78,11%	84,4% 79,99% 91,76%	≤17,34% ≥82,31% ≥82,89%	84,79% 80,85% 91,17%
0.001	9921	2806	6462	14173	4521	78606
	≤41,04% ≥58,92% ≥58,97%	82,06% 77,16% 91,09%	≤28,43% ≥71,53% ≥71,59%	84,09% 79,32% 92,24%	≤19,95% ≥80,04% ≥80,06%	84,44% 80,23% 91,40%

	500		1000	
0.05	11580	28843	11580	28843
	≤28,11% ≥75,40% ≥70,46%	80,32% 74,07% 93,30%	≤28,11% 75,40% 70,46%	80,32% 74,07% 93,30%
0.01	3631	545002	3545	702007
	≤15,58% ≥83,90% ≥84,79%	82,86% 79,15% 89,23%	≤15,18% ≥84,31% ≥85,18%	82,93% 79,22% 89,28%
0.005	3401	980519	3228	1589257
	≤14,94% ≥84,44% ≥85,49%	84,60% 80,77% 90,83%	≤14,17% ≥85,39% ≥86,16%	84,24% 80,46% 90,44%
0.001	3955	340252	3772	751056
	≤17,46% ≥82,48% ≥82,58%	84,78% 80,70% 91,42%	≤16,66% ≥83,31% ≥83,36%	84,77% 80,69% 91,42%

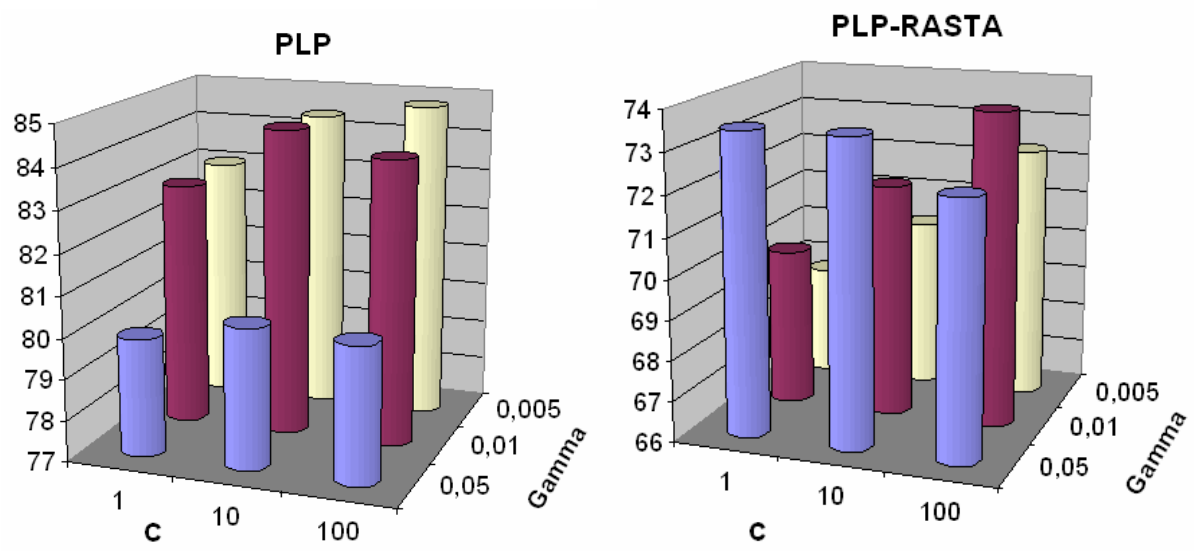
Dans ces deux tables, on arrive à trouver des résultats très intéressants, avec des estimations Xi-alpha pour l'erreur au-dessous de 15% et avec taux de réussite proches du 85% mesurés sur la base de test.

D'un autre coté, jusqu'à présent, on a travaillé seulement avec des vecteurs de caractéristiques obtenus par PLP d'ordre 12 et sans filtrage RASTA, mais on peut s'intéresser à évaluer l'influence de ce type de filtrage sur les performances de classification.

Dans la table ci-dessous on observe quelques résultats pour des simulations faites avec les mêmes bases d'apprentissage et test, mais avec le filtrage RASTA activé pour l'obtention des « *feature vectors* » :

	1		10		100	
0.05	14282	11732	11709	57675	9736	158130
	≤44,25% ≥57,76% ≥55,52%	73,45% 70,3% 81,21%	≤37,16% ≥65,61% ≥62,16%	73.50% 70,31% 81,37%	≤31,71% ≥70,90% ≥67,38%	72,31% 69,46% 79,63%
0.01	13758	5763	11041	57946	8755	581959
	≤59,39% ≥40,60% ≥40,60%	69,90% 70,33% 68,82%	≤48,21% ≥51,83% ≥51,79%	71,78% 70,80% 74,14%	≤38,43% ≥61,84% ≥61,51%	73,76% 72,47% 76,62%
0.005	15215	4436	12449	42107	9736	158130
	≤66,56% ≥33,43% ≥33,44%	68,75% 70,93% 62,92%	≤54,79% ≥45,37% ≥45,23%	70,22% 70,34% 69,94%	≤72,31% ≥69,46% ≥79,63%	72,31% 69,46% 79,63%

On observe que les performances sont clairement inférieures à celles que l'on avait obtenues sans RASTA :



6 Discussion

6.1 Comparaison des moteurs svm

Chacun des deux moteurs que j'ai utilisé dans mon projet a été bien utile pour tester la faisabilité des objectifs initiaux concernant la construction d'un classificateur capable de faire la distinction entre les chansons d'artistes différents avec un certain degré de robustesse.

Il faut aussi dire que les deux moteurs peuvent bien être utilisés de manière combinée. Dans mon cas, par exemple, OSU-SVM a été très utile pour visualiser les résultats concrets de la classification de chaque chanson depuis Matlab, tandis que SVM-Light m'a permis de faire un test plus exhaustif de l'influence des différents paramètres sur les performances de classification. Puisque les paramètres qu'ils prennent, bien que pas identiques, sont assez semblables dans leur signification, on peut bien extrapoler les résultats obtenus avec chacun des moteurs au domaine de simulation concernant l'autre.

De manière résumée, et toujours à partir de ma propre expérience, je essayerai de présenter les points forts et les points faibles de chacun des deux moteurs dans la table suivante :

OSU-SVM / LIBSVM
Points forts
- Interface très amicale (« user-friendly ») sur Matlab. Puissante et facile à utiliser. Elle utilise des fonctions de type MEX pour optimiser la performance en terme de temps de calcul et gestion mémoire. - Il permet de travailler très facilement avec plusieurs classes. Il permet même d'obtenir des matrices de confusion de manière directe à partir d'une base de test. - Très bien documenté. Sur le site web [7], on peut trouver un document qui explique en détail les différents algorithmes utilisés [8] et une guide rapide d'utilisation plus simple pour « débutants » [9].
Points faibles
- Il serait souhaitable d'avoir plus d'indicateurs et d'estimateurs pour la précision et les capacités de généralisation associées à chaque processus d'apprentissage et de classification (par exemple, on aimerait bien avoir une estimation de la dimension de Vapnik-Chervonenkis).

SVM-Light
Points forts
- Information très détaillée autour de chaque processus d'apprentissage, avec des nombreux indicateurs et estimateurs pour les performances, tels que la dimension VC, les estimations « leave-one-out » pour l'erreur sur la base d'apprentissage, les estimateurs Xi-alpha et des mesures réelles sur les bases de test concernant l'erreur, la précision et le recall. - Code très compacte et facilement portable d'un système à un autre. Le logiciel consiste uniquement à deux fichiers exécutables qui permettent de faire l'apprentissage et la classification.

- Le logiciel et fortement parametrisable et il offre des nombreux choix à l'utilisateur pour faire l'apprentissage. Il est même possible d'utiliser directement des kernels définis par l'utilisateur. - Les algorithmes et le code ont été optimisés du point de vue temps de calcul et gestion de mémoire et donc il offre des très bonnes performances pour le travail avec des grandes ensembles de vecteurs et avec des vecteurs de grande dimension.
Points faibles
- La documentation est un peut faible. Pour bien comprendre le fonctionnement du code il faut regarder des articles individuels de Joachims, ou bien acheter le livre qu'il a écrit récemment. Il serait souhaitable d'avoir quelque sorte de « guide pour l'utilisateur » expliquant plus en détail l'effet et la signification de chacun des paramètres. - L'interface sur Matlab que j'ai utilisé était trop lente, ce qui m'a obligé à générer les fichiers de données avec Matlab et puis exécuter SVM-Light hors Matlab. J'ai pu observer une différence énorme en terme de temps de calcul. Il serait donc, envisageable, de travailler sur une meilleure intégration du logiciel SVM-Light sur Matlab pour arriver à avoir simultanément « le mieux des deux mondes ».

Comme conclusion, je pense que OSU-SVM/LIBSVM constituent une option plus intéressante pour les utilisateur qui veulent travailler sur Matlab et qui n'a pas besoin de connaître des aspects tels que la dimension VC ou les estimations Xi-alpha pour leur travail. Pour les utilisateurs plus concernés par les aspects mathématiques autours le processus d'apprentissage, SVM-Light constitue une option plus complète en leur offrant plus de fonctionnalités qu'ils trouveront, bien sûr, très utiles.

6.2 Evaluation des résultats obtenus

Après les nombreuses simulations faites pour le projet, je suis arrivé à obtenir des taux de réussite d'environ 85% pour le problème de la discrimination entre les chansons de « The Beatles » et George Brassens.

Ces résultats ont été obtenus pour des vecteurs de caractéristiques calculés à travers une technique PLP d'ordre 12 et sans filtrage RASTA. Les simulations faites à ce respect montrent que le filtrage RASTA n'apporte pas aucune amélioration pour le processus de classification. Au contraire, les résultats obtenus avec PLP-RASTA sont toujours au-dessus de ceux obtenus avec PLP tout seul. Comme on a expliqué dans la section correspondante, le but habituel du filtrage RASTA est de compenser les perturbations et le bruit introduit par le canal de communication, à travers l'élimination de certaines fréquences, qui permette de mieux se concentrer sur la reconnaissance de la parole dans des applications du type « speech recognition ». Dans note cas, par contre, le fait de travailler avec des enregistrements de haute qualité, sans bruit de canal, et le fait d'aborder un problème de reconnaissance de contenus musicaux (différent à celui de la reconnaissance de « speech »), semblent rendre plutôt indésirable l'utilisation des techniques RASTA.

En ce qui concerne la transformation non-lineaire des données, j'ai utilisé des fonctions RBF. On a observé que les meilleures performances étaient obtenues pour des valeurs petites de gamma et pour des valeurs élèves de la fonction de coût C.

Les estimations des performances faites à travers de la technique de « *v-fold cross validation* » pour OSU-SVM/LIBSVM et à travers les estimateurs Xi-alpha pour SVM-Light ont été très proches des capacités réelles de généralisation mesurés sur des bases de test.

En général, si l'on adopte un critère de décision avec un seuil de 50% sur les vecteurs de caractéristiques obtenus à partir d'un fragment de chanson, on obtient toujours des taux d'identification et réussite très élevées pour toutes les chansons de Brassens, tandis que pour les Beatles on classifie bien la plupart, mais on arrive parfois à mal classer certaines chansons (peut être à cause du fait que ces chansons s'éloignent un peu du style habituel du groupe britannique ?).

En tout cas, à mon avis, ce serait très difficile, voire impossible, même pour un expert humain de bien classer toutes les frames temporels pour chaque chanson, puisqu'il doit en avoir certaines qui se ressemblent énormément pour les deux artistes (on peut penser, par exemple, à des moments proches du silence ou à des moments dans lesquels on n'entend que des instruments et des notes très similaires). C'est pourquoi je crois que les taux de réussite obtenus, autour du 85%, sont assez satisfaisants.

6.3 Difficultés rencontrées et possibles améliorations

Sans doute, la principale difficulté rencontrée pendant la réalisation de ce projet a été le temps de calcul associée à certains processus d'apprentissage. En effet, il semble que le choix des paramètres pour l'apprentissage a une grande influence sur les performances du classificateur final, mais aussi sur le temps de calcul.

Pour OSU-SVM/LIBSVM, par exemple, un apprentissage fait sur la même base de vecteurs de caractéristiques peut prendre 30 minutes (pour des valeurs élevées de C et petites de gamma) ou plus de 4 heures (si l'on fait un choix de paramètres à l'inverse).

Comme j'ai déjà mentionné, le fait d'avoir un seul ordinateur à ma disposition avec Matlab (qui, en plus, n'était pas spécialement puissant: AMD Duron 1100Mhz, 256 MB RAM) m'a empêché d'arriver à faire tous les test pour OSU-SVM/LIBSVM que j'aurai eu envie de faire. Il serait donc, souhaitable, d'enrichir le travail fait ici avec la construction de grilles sur OSU-SVM pour toutes les combinaisons possibles de paramètres C et gamma montrés dans la section de résultats, afin de pouvoir les comparer à celles obtenues pour SVM-Light. Il serait aussi intéressant de tester d'autres choix pour le kernel, comme par exemple celle des kernels polynomiaux.

Il serait aussi envisageable de faire plus de simulations pour tester les différentes possibilités concernant l'obtention des vecteurs de caractéristiques. Ici, j'ai testé avec PLP d'ordre 12 et avec et sans filtrage RASTA. On pourrait, par exemple, étudier l'influence de l'ordre de PLP sur les performances de classification. On pourrait, aussi, évaluer les performances d'autres techniques pour l'extraction de *features* très utilisées

aujourd'hui dans le domaine de la reconnaissance de signaux audio, comme les coefficients cepstra en escale de Mel (MFCC).

Une autre possibilité serait celle d'évaluer la robustesse de la classification face à la possible présence de bruit. Aussi, il serait souhaitable d'essayer de mieux comprendre l'influence du choix de la longueur de l'intervalle considéré pour chaque chanson. Y a-t-il des intervalles minimaux au-dessous desquels on n'arrive pas à obtenir des bonnes classifications ? Est-il mieux de considérer beaucoup d'intervalles très petits appartenant à beaucoup de chansons, ou est-il plus intéressant de prendre des intervalles plus longues avec un nombre plus réduite de chansons ?

Au cours de nos simulations, on a travaillé au début avec des intervalles de 10^6 échantillons prises sur 10 chansons et on a considéré après des intervalles de $5 \cdot 10^5$ échantillons prises sur 20 chansons. Les résultats semblaient être beaucoup mieux pour le deuxième cas, mais il y en aura peut-être des limites pour cette tendance que l'on pourrait essayer de déterminer, soit de manière empirique, soit à l'aide d'outils mathématiques comme la dimension de Vapnik-Chervonenkis.

6.4 Valorisation finale du projet et du travail accompli

Comme je viens juste de montrer, il reste beaucoup de choses à faire pour avoir une vision plus riche du problème traité dans mon projet. Il s'agit d'un domaine passionnant qui, en plus, peut offrir des nombreuses applications pratiques face à l'avenir, tels que des moteurs de recherche type Audio-google ou des reproducteurs de musique intelligents, pour en citer seulement un couple.

En tout cas, comme j'avais déjà mentionné au début de ce rapport, mon objectif ici n'était pas de développer un outil super - performant, ni de me placer même près de l'état de l'art dans le domaine, sinon plutôt de présenter, tout simplement, un exemple d'application des svm pour un problème réel, à travers duquel on a pu constater l'énorme intérêt de ces outils dans le domaine de la reconnaissance de formes et l'apprentissage statistique.

Pour moi, la réalisation de ce projet a été très enrichissante du point de vue pédagogique puis qu'elle m'a permis de mieux comprendre les idées et les concepts que l'on a traités pendant le cours du Mastère MVA. J'aimerais bien que, face à l'avenir, d'autres gens puissent aussi profiter de la valeur pédagogique du matériel ici présenté. La possibilité existe … il suffira, donc, de « *Let it be* ». ☺

7 Références

- Notes du cours « Introduction aux théories de l'apprentissage » de Robert Azencott pour le Mastère de Recherche en Mathématiques Appliquées « *Mathématiques, vision, apprentissage* » à l'ENS Cachan. Année 2004/05.

Articles

[3] *« Perceptually based linear predictive analysis of speech (PLP) »*
Hermansky, H.; Hanson, B.; Wakita, H.;
Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP '85. , Volume: 10 , Apr 1985 Pages:509 – 512
[Document accessible à travers de IEEEXPLORE : http://ieeexplore.ieee.org]

[4] « RASTA processing of speech »,
H. Hermansky and N. Morgan
IEEE Transactions on Speech and Audio Processing,
vol. 2, num. 4, pp. 578-589, Oct, 1984
http://www.bme.ogi.edu/~hynek/publications/rasta.pdf

[12] T. Joachims, 11 in: *« Making large-Scale SVM Learning Practical. Advances in Kernel Methods - Support Vector Learning »*,
de B. Schölkopf and C. Burges and A. Smola (ed.), MIT Press,1999.
http://www.joachims.org/publications/joachims_99a.pdf

[13] *« Estimating the Generalization Performance of a SVM Efficiently »*
T. Joachims
Proceedings of the International Conference on Machine Learning (ICML), Morgan Kaufman, 2000.
http://www.cs.cornell.edu/People/tj/publications/joachims_00a.pdf

[8] *« LIBSVM: a Library for Support Vector Machines »*
Chih-Chung Chang and Chih-Jen Lin.
December, 2004
http://www.csie.ntu.edu.tw/~cjlin/papers/libsvm.pdf

[9] *« A Practical Guide to Support Vector Classification »*
Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin
Department of Computer Science and Information Engineering
National Taiwan University
http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf

Sites web

[1] *« Feature Extraction »*
Johan Schalkwyk
Center for Spoken Language Understanding
Oregon Health and Science University
http://cslu.cse.ogi.edu/toolkit/old/old/version2.0a/documentation/csluc/node5.html

[2] *« A Brief Introduction to Speech Analysis and Recognition. An Internet Tutorial »*
Oscar Mayora Ibarra and Francesco Curatelli
Dipartimento di Ingegneria Biofisica ed Elettronica
Università degli Studi di Genova
http://www.mor.itesm.mx/~omayora/Tutorial/tutorial.html

Logiciels en ligne

[5] PLP and RASTA in Matlab
http://www.ee.columbia.edu/~dpwe/resources/matlab/rastamat/

[6] OSU-SVM Classifier Matlab Toolbox
http://www.ece.osu.edu/~maj/osu_svm/

[10] OSU-SVM API documentation
http://svm.sourceforge.net/docs.shtml

[7] LIBSVM -- A Library for Support Vector Machines
http://www.csie.ntu.edu.tw/~cjlin/libsvm/

[11] SVM-Light
http://svmlight.joachims.org/

[14] Matlab inferface for SVM-Light
http://www.cis.tugraz.at/igi/aschwaig/software.html