Université de Montréal

# Algorithmes de recommandation musicale

par
François Maillet

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté à la Faculté des arts et des sciences
en vue de l'obtention du grade de Maître ès sciences (M.Sc.)
en informatique

décembre, 2009

Université de Montréal
Faculté des arts et des sciences

Ce mémoire intitulé:

**Algorithmes de recommandation musicale**

présenté par:

François Maillet

a été évalué par un jury composé des personnes suivantes:

| | |
|---|---|
| Pascal Vincent, | président-rapporteur |
| Douglas Eck, | directeur de recherche |
| Fabian Bastin, | membre du jury |

## RÉSUMÉ

Ce mémoire est composé de trois articles qui s'unissent sous le thème de la recommandation musicale à grande échelle.

Nous présentons d'abord une méthode pour effectuer des recommandations musicales en récoltant des étiquettes (*tags*) décrivant les items et en utilisant cette *aura* textuelle pour déterminer leur similarité. En plus d'effectuer des recommandations qui sont transparentes et personnalisables, notre méthode, basée sur le contenu, n'est pas victime des problèmes dont souffrent les systèmes de filtrage collaboratif, comme le **problème du démarrage à froid** (*cold start problem*).

Nous présentons ensuite un algorithme d'apprentissage automatique qui applique des étiquettes à des chansons à partir d'attributs extraits de leur fichier audio. L'ensemble de données que nous utilisons est construit à partir d'une très grande quantité de données sociales provenant du site Last.fm.

Nous présentons finalement un algorithme de génération automatique de liste d'écoute personnalisable qui apprend un espace de similarité musical à partir d'attributs audio extraits de chansons jouées dans des listes d'écoute de stations de radio commerciale. En plus d'utiliser cet espace de similarité, notre système prend aussi en compte un nuage d'étiquettes que l'utilisateur est en mesure de manipuler, ce qui lui permet de décrire de manière abstraite la sorte de musique qu'il désire écouter.



**Mots clés: apprentissage automatique, recommandation musicale, similarité musicale, application automatique d'étiquettes, génération de liste d'écoute, traitement de signal.**

## ABSTRACT

This thesis is composed of three papers which unite under the general theme of large-scale music recommendation.

The first paper presents a recommendation technique that works by collecting text descriptions of items and using this textual aura to compute the similarity between them using techniques drawn from information retrieval. We show how this representation can be used to explain the similarities between items using terms from the textual aura and further how it can be used to steer the recommender. Because our system is content-based, it is not victim of the usual problems associated with collaborative filtering recommenders like the cold start problem.

The second paper presents a machine learning model which automatically applies tags to music. The model uses features extracted from the audio files and was trained on a very large data set constructed with social data from the online community Last.fm.

The third paper presents an approach to generating steerable playlists. We first demonstrate a method for learning song transition probabilities from audio features extracted from songs played in professional radio station playlists. We then show that by using this learnt similarity function as a prior, we are able to generate steerable playlists by choosing the next song to play not simply based on that prior, but on a tag cloud that the user is able to manipulate to express the high-level characteristics of the music he wishes to listen to.



**Keywords: machine learning, music recommendation, music similarity, automatic tagging of music, playlist generation, signal processing.**

## TABLE DES MATIÈRES

## LISTE DES TABLEAUX

## LISTE DES FIGURES

## LISTE DES ANNEXES

## LISTE DES SIGLES

| | |
|---|---|
| AA | Apprentissage automatique |
| CBR | Case-based reasoning |
| DFT | Discrete Fourier transform |
| FC | Filtrage collaboratif |
| GMM | Gaussian mixture model |
| Hz | Hertz |
| iid | Indépendantes et identiquement distribuées |
| MFCC | Mel-frequency cepstral coefficients |
| MIC | Modulation d'impulsion codée |
| MLP | Multi-layer perceptron |
| MP3 | MPEG-1/2 audio layer 3 |
| SdA | Stacked denoising autoencoders |
| TF $\times$ IDF | Term frequency-inverse document frequency |

## NOTATION

Dans ce document, la notation suivante sera utilisée, à moins d'indication contraire. Les vecteurs sont représentés par une lettre minuscule telle $\mathbf{x}$ et sont considérés des vecteurs colonnes. On peut représenter les éléments d'un vecteur $\mathbf{x}$ de dimension $d$ comme $\mathbf{x} = (x_0, x_1, ..., x_d)^T \in \mathbb{R}^d$.

On représente les matrices par une majuscule telle $\mathbf{M}$.

| | |
|---|---|
| $\mathscr{D}$ | Ensemble de données d'entraînement |
| $\mathscr{D}_{test}$ | Ensemble de données de test |
| $\mathbb{E}_{x\sim\mathscr{P}}$ | Espérance par rapport aux observations $x$ issues de la distribution $\mathscr{P}$ |
| $\mathbb{R}^d$ | Ensemble des nombres réels de dimensionalité $d$ |
| $C(\cdot)$ | Fonction de coût |
| $C(f\|\mathscr{D})$ | Coût de la fonction $f$ sur l'ensemble de données $\mathscr{D}$ |
| $f_\Theta$ | Fonction $f$ paramétrisée par les paramètres $\Theta$ |
| $\phi$ | Fonction de base non linéaire, représentant implicitement $\phi(\mathbf{x})$ |
| $\mathbf{x}^{(i)}$ | $i$-ème point de données $\in \mathbb{R}^d$, issue d'un ensemble $\mathscr{D}$ |
| $\mathrm{t}^{(i)}$ | $i$-ème cible, issue d'un ensemble $\mathscr{D}$ |
| $h^{(i)}$ | $i$-ème modèle faible |
| $\alpha^{(i)}$ | Poids associé à $h^{(i)}$ |
| $\sigma(\cdot)$ | Fonction sigmoïde |
| $\|\|\mathbf{a}\|\|$ | Norme euclidienne du vecteur $\mathbf{a}$ |
| $I(\text{condition})$ | Fonction indicatrice, vaut 1 si condition est vrai, 0 sinon |
| $\frac{\partial}{\partial x}f$ | Dérivée de la fonction $f$ par rapport à $x$ |

*À mes parents, qui ont toujours cru en moi et m'ont toujours supporté inconditionnellement depuis mon plus jeune âge, surtout dans les moments les plus difficiles. Sans vous, rien n'aurait été possible.*

## REMERCIEMENTS

Je tiens tout d'abord à remercier mon directeur de recherche, Douglas Eck, pour m'avoir donné la chance de poursuivre des études supérieures dans ce domaine fantastique, ainsi que pour son appui financier.

Je veux remercier mes collègues membres du laboratoire GAMME et LISA, particulièrement, Sean Wood, Simon Lemieux, Thierry Bertin-Mahieux, Guillaume Desjardins et Hugo Larochelle, pour leur aide, les discussions, les longues soirées et toutes les aventures partagées.

Je souhaite aussi remercier tous mes anciens collègues de *Sun Labs*, Stephen Green, Jeffrey Alexander et particulièrement Paul Lamere, pour la confiance qu'il m'a témoignée.

Je tiens à souligner le travail d'Olivier Breuleux et James Bergstra du laboratoire LISA dans le développement de Theano[1], sur lequel l'implémentation de plusieurs algorithmes de ce mémoire repose.

Je tiens à mentionner que les travaux dont il est question dans ce mémoire auraient été extrêmement difficiles à réaliser sans les services web gratuits des sites *Last.fm*[51, 53] ainsi que *Yes.com*[97]. L'accès à de tels ensembles de données est essentiel à l'avancement de la recherche.

Je tiens finalement à mentionner que la recherche effectuée dans le cadre de mes études utilise presque uniquement des librairies et logiciels gratuits à source libre créés grâce au temps investi, dans bien des cas, par des bénévoles. Entre autres, *Python* et toutes les librairies associées comme *NumPy* et *matplotlib*, et aussi *SQLite*.

Dans le même ordre d'idée, *Wikipedia* a aussi été un outil aussi pratique qu'essentiel.

[1] Theano : `http://lgcm.iro.umontreal.ca`

# CHAPITRE 1

# INTRODUCTION

Nous assistons présentement à une véritable révolution du monde musical. L'ère du Web 2.0 a amené une démocratisation de tous les aspects de la musique qui a fondamentalement changé la relation qu'ont les gens avec elle.

En effet, comme le disque de vinyle a jadis dû céder sa place au disque compact, ce dernier est en train de se faire progressivement remplacer, comme médium de distribution de la musique commerciale, par les fichiers audio numériques tels le *MP3*. Les magasins en ligne comme *iTunes*[1] offrent maintenant plus de 10 millions de chansons numériques téléchargeables. D'autres compagnies comme *Spotify*[2] proposent un abonnement qui permet de choisir parmi plus de 6 millions de chansons et de les écouter instantanément par transmission de données en continu (*stream*). Les chansons n'étant pas préalablement téléchargées, l'accès à la totalité de leur catalogue est simple et rapide.

Contrairement aux magasins traditionnels qui ont une limite physique d'espace et donc une quantité finie de produits qu'ils peuvent proposer à leurs clients, les magasins en ligne n'ont aucune restriction quant à la taille du catalogue qu'ils peuvent offrir. Le jour où toute la musique commerciale jamais enregistrée sera accessible instantanément par l'entremise du web est beaucoup moins lointain que plusieurs peuvent se l'imaginer.

Le problème des utilisateurs n'est maintenant plus à savoir si une chanson précise est accessible, mais bien de déterminer ce qu'ils désirent écouter. De nouveaux outils sont essentiels pour permettre aux utilisateurs d'explorer efficacement ce nouvel espace et de profiter pleinement des nouvelles possibilités qu'il offre. Les nouveaux outils de recommandation musicale seront pour la musique ce que Google a été pour le web en général.

Le défi est de permettre aux utilisateurs d'explorer la **longue traîne** (*long tail*)[2]. En effet, la situation actuelle est que très peu de musique représente la très grande majorité des ventes. Selon les données de vente provenant de *Nielsen Soundscan 2007*, on constate que des 4 millions de chansons vendues en ligne cette année-là, seulement 1% des chansons représentent près de 80% des ventes[49]. Sur la figure 1.1[3], illustrant la longue traîne de manière conceptuelle, l'axe horizontal représente les artistes et l'axe vertical représente le volume de vente, ou d'écoutes. Les artistes populaires se trouvent dans la zone de gauche, où très peu d'entre eux représentent une très grande proportion des ventes. Les systèmes de recommandations actuels, au lieu d'aider les utilisateurs à explorer la longue traîne, les gardent dans le début de la courbe en recommandant

[1] `http://www.apple.com/itunes`
[2] `http://www.spotify.com`
[3] Image réalisée par Hay Kranen. Voir `http://en.wikipedia.org/wiki/File:Long_tail.svg`

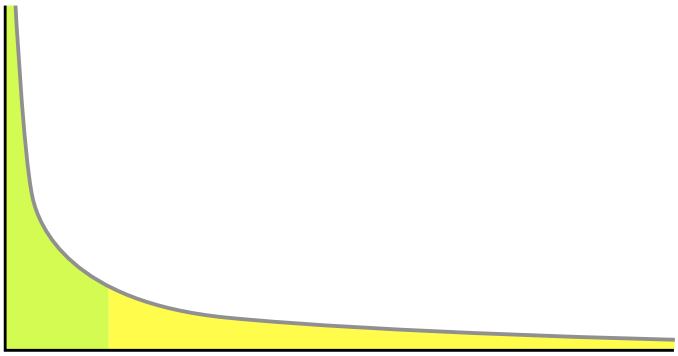

Figure 1.1 – Illustration conceptuelle de la longue traîne, représentée par une fonction suivant une loi de puissance, où l'axe horizontal représente divers artistes tandis que l'axe vertical représente le volume de vente. La figure représente la situation actuelle, où très peu d'artistes représentent la presque totalité des ventes de musique.

des artistes populaires[24]. Les magasins à grande surface étaient déjà très efficaces pour vendre ce type de musique. Sur le web, on se doit de faire mieux.

D'ici quelques années, la révolution sera terminée et nous aurons accès, où que nous soyons, à tous les enregistrements de toutes les chansons de tous les artistes commerciaux et amateurs. Des centaines de milliers de nouvelles chansons s'ajouteront chaque semaine. Avec tant de choix, ne devrions-nous pas tous être constamment en train de découvrir de la nouvelle musique spécifiquement choisie selon nos goûts ?

C'est dans ce contexte de profonds changements que ce mémoire se positionne. Il est construit autour de trois articles qui s'unissent sous le thème général de la recommandation musicale à grande échelle. Nous avons réalisé toute la recherche dont il est question dans ce mémoire sur des ensembles de données très grands et avons utilisé des techniques extensibles (*scalability*). Les modèles présentés sont donc applicables dans un contexte commercial réel.

## 1.1 Les étiquettes

L'ensemble des travaux réalisés dans le cadre de ce projet utilise les étiquettes (*tags*), qui sont un type de métadonnée. Les étiquettes sont une nouvelle manière de décrire, classer et chercher des documents. Elles sont devenues très populaires ces dernières années sur le web et sont appliquées à plusieurs types d'items, comme par exemple des marques pages, des photos ou de la musique. Contrairement au modèle classique d'organisation d'éléments qui consiste à leur assigner une seule catégorie (analogue à les placer dans un dossier), les étiquettes sont non hiérarchiques et non exclusives. On peut donc assigner plusieurs étiquettes à chaque élément, ce qui crée un espace beaucoup plus riche.

Le meilleur exemple de l'utilisation d'étiquettes pour décrire la musique est la communauté

musicale *Last.fm*[53], où des millions d'utilisateurs appliquent des étiquettes à la musique qu'ils écoutent. Les utilisateurs appliquent des étiquettes à leur musique pour différentes raisons[48] : pour aider la recherche, pour organiser, par but d'affirmation sociale, par exprimer une opinion, etc. Le tableau 1.1 présente les différentes catégories d'étiquettes utilisées sur Last.fm.

| Catégories d'étiquette | Fréquence | Exemples |
|---|---|---|
| Genre musical | 68% | heavy metal, punk |
| Localisation | 12% | French, Seattle, NYC |
| Atmosphère | 5% | chill, party |
| Opinion | 4% | love, favorite |
| Instrumentation | 4% | piano, female vocal |
| Style | 3% | political, humor |
| Varia | 2% | Coldplay, composers |
| Personnel | 1% | seen live, I own it |
| Organisationnel | 1% | check out |

Tableau 1.1 – Distribution de la sorte des 500 étiquettes les plus populaires sur le site Last.fm[53]. Tableau provenant de [48].

Plus d'information sur la collecte ainsi que sur l'usage des étiquettes dans le contexte de la recommandation sera présentée tout au long de ce mémoire. Le lecteur est aussi invité à consulter [48].

## 1.2 Aperçu des chapitres

Le reste de ce mémoire est organisé comme suit. Une introduction à l'apprentissage automatique et à l'extraction d'attributs à partir de fichiers audio est présentée au chapitre 2. Chacun des trois articles est ensuite présenté et est précédé par un chapitre le résumant.

Tout d'abord, l'article *Generating Transparent, Steerable Recommendations from Textual Descriptions of Items* est introduit au chapitre 3 et inclus au chapitre 4. Cet article présente un système de recommandation musicale extrêmement extensible qui utilise des étiquettes provenant de plusieurs sources, dont celles générées par *Autotagger* (chapitre 6). Les recommandations faites par ce système sont transparentes et personnalisables, de sorte que l'utilisateur peut comprendre pourquoi un artiste lui a été recommandé et peut ensuite donner de la rétroaction au système pour qu'il ajuste ses recommandations futures.

Ensuite, l'article *Autotagger : a model for predicting social tags from acoustic features on large music databases* est présenté au chapitre 5 et inclus au chapitre 6. Cet article traite d'un modèle d'apprentissage appliquant automatiquement des étiquettes à des chansons à partir de leur fichier audio. Le modèle fut entraîné à partir de données sociales provenant du site *Last.fm*

et utilise le modèle d'apprentissage Filterboost.

Finalement, l'article *Steerable Playlist Generation by Learning Song Similarity from Radio Station Playlists* est introduit au chapitre 7 et inclus au chapitre 8. Cet article introduit une méthode en deux étapes de génération de listes d'écoute personnalisables (*steerable playlists*). Un modèle d'apprentissage est tout d'abord utilisé pour apprendre, à partir de listes d'écoute de stations de radio commerciale, un espace de similarité musical à partir duquel on peut déterminer quelles chansons peuvent se suivre harmonieusement dans une liste d'écoute. Ensuite, nous utilisons les concepts de recommandations transparentes et personnalisables (chapitre 4) ainsi que des étiquettes générées par Autotagger (chapitre 6) pour permettre à l'utilisateur de personnaliser la génération de la liste d'écoute en utilisant un nuage d'étiquettes personnalisable (*steerable tag cloud*).

Nous concluons finalement ce mémoire au chapitre 9.

# CHAPITRE 2

# NOTIONS PRÉALABLES

## 2.1 Introduction à l'apprentissage automatique

La section suivante est consacrée aux bases de l'apprentissage automatique et présente des notions essentielles à la compréhension des modèles présentés dans ce mémoire. Elle est inspirée de l'excellent livre de Bishop [19] ainsi que des thèses de deux collèges[27, 50].

### 2.1.1 Qu'est-ce que l'apprentissage automatique ?

L'**apprentissage automatique** (*machine learning*), aussi appelé apprentissage statistique, est un type d'intelligence artificielle où un algorithme modifie son comportement automatiquement de manière à améliorer sa performance sur une tâche à partir d'un ensemble de données. Nous qualifions ce processus d'*apprentissage* étant donné que l'algorithme est optimisé à partir d'un ensemble d'observations et qu'il essaye d'en extraire les régularités statistiques.

Le type d'apprentissage dont il sera principalement question dans ce mémoire est l'**apprentissage supervisé**. Dans ce type d'apprentissage, l'objectif de l'algorithme est de prédire une cible t explicite et connue à partir des données d'entraînement. Les deux types de cibles les plus courantes sont soit des valeurs continues où $\mathrm{t} \in \mathbb{R}$ (problème de **régression**) ou des classes discrètes où $\mathrm{t} \in \{1, ..., N_C\}$ pour un problème à $N_C$ classes (problème de **classification**).

Considérons un exemple concret de classification. Plusieurs travaux (par exemple [12]) se sont penchés sur le problème de la classification automatique de chansons en différents genres musicaux (rock, jazz, etc). Cette tâche, relativement simple pour un humain, est quasi impossible à résoudre en utilisant un algorithme standard. Par contre, en utilisant l'apprentissage automatique, il devient possible d'entraîner un modèle qui aura une performance très raisonnable.

Un algorithme d'apprentissage utilisera un **ensemble d'entraînement** (*training set*) $\mathscr{D}$ composé de chansons dont on connait le genre musical. La phase de prétraitement des fichiers audio (voir section 2.3) permettra d'extraire, pour chaque chanson $i$, un vecteur d'attributs (*feature vector*) $\mathbf{x}^{(i)} \in \mathbb{R}^d$ de plus haut niveau, qui la caractérisent. Posons de plus la valeur de cible $\mathrm{t}^{(i)}$, qui encode le genre musical de la chanson $i$, et représente la cible que l'algorithme devra prédire. L'ensemble d'entrainement $\mathscr{D}$ peut donc être formellement défini comme $\mathscr{D} = \{(\mathbf{x}^{(i)}, \mathrm{t}^{(i)}) | \mathbf{x}^{(i)} \in X, \mathrm{t}^{(i)} \in Y\}_{i=1}^{I}$.

Mentionnons aussi un autre type d'apprentissage, soit l'**apprentissage non supervisé**. Contrairement à l'apprentissage supervisé, aucune cible t explicite n'est utilisée pendant l'entraînement, d'où le nom *non supervisé*. Ce type d'apprentissage inclus des techniques comme le **groupage**

(*clustering*), c'est-à-dire la recherche de groupes d'items similaires dans les données. Certains algorithmes non-supervisés sont aussi utilisés pour modéliser la distribution des données d'entrées (*density estimation*) ou encore pour effectuer une réduction de dimensionalité, soit transformer les vecteurs d'entrées $\mathbf{x}^{(i)} \in \mathbb{R}^d$ en vecteurs $\mathbf{x}'^{(i)} \in \mathbb{R}^{d'}$, où $d' < d$. La réduction de dimensionalité peut être utilisée pour des fins de visualisation (si $d' \leqslant 3$) ou encore pour obtenir une version plus compacte et moins bruitée des données originelles. Les vecteurs $\mathbf{x}'^{(i)}$ pourront ensuite potentiellement être utilisés comme données d'entrée à un algorithme supervisé.

### 2.1.2 Minimisation de l'erreur empirique

Notre ensemble d'entraînement $\mathscr{D}$ est un échantillon issu d'une distribution $\mathscr{P}$ qui est inconnue et que nous tentons de modéliser. L'algorithme d'apprentissage a comme objectif de trouver la fonction $f \in F$ qui estime le mieux la véritable distribution, où $F$ est l'ensemble de toutes les fonctions $f : X \to Y$ possibles.

Pour être en mesure de déterminer la performance de $f$, nous devons introduire le concept de **fonction de coût** ($C(\cdot)$), qui mesure l'erreur d'une fonction sur des paires $(\mathbf{x}^{(i)}, \mathrm{t}^{(i)})$.

Ce que nous voulons minimiser est le **risque réel** (ou **risque espéré**) de notre fonction $f$, c'est-à-dire son erreur sur la distribution $\mathscr{P}$. Considérons la fonction de coût 0-1 calculant le risque réel, nous voulons alors trouver :

$$C_{0-1}(f) = \mathbb{E}_{(\mathbf{x},\mathrm{t})\sim\mathscr{P}} I(\mathrm{t} \neq f(\mathbf{x})), \tag{2.1}$$

où $I$ est la fonction indicatrice.

Par contre, puisque nous ne connaissons pas la distribution $\mathscr{P}$, nous devons utiliser nos données $\mathscr{D}$, qui furent générées par le processus réel, comme intermédiaire et plutôt calculer l'**erreur empirique**, soit l'erreur sur les exemples que nous possédons. L'erreur empirique de $f$ sur l'ensemble de données $\mathscr{D}$ est donc défini par :

$$\hat{C}_{0-1}(f|\mathscr{D}) = \frac{1}{|\mathscr{D}|} \sum_{(\mathbf{x}^{(i)},\mathrm{t}^{(i)})\in\mathscr{D}} I(\mathrm{t} \neq f(\mathbf{x})). \tag{2.2}$$

L'erreur empirique est donc un estimateur du risque réel, calculé sur un échantillon.

### 2.1.3 La généralisation et les ensembles de données

L'objectif fondamental de l'apprentissage automatique est la **généralisation**. En effet, il est facile de construire un modèle qui apprendra *par coeur* l'ensemble d'entraînement, c'est-à-dire de trouver une fonction $f$ telle que $C(f|\mathscr{D}) \approx 0$. Mais comme nous l'avons mentionné, cet ensemble ne représente qu'un échantillon d'un processus inconnu et la véritable difficulté est de

trouver une fonction $f$ qui saura être performante sur des données de ce processus auxquelles elle n'a pas été exposée durant son entraînement.

Introduisons l'**ensemble de test** (*test set*) $\mathscr{D}_{test}$, qui est semblable à l'ensemble d'entraînement et contient des observations issues du même processus. Les données de l'ensemble de test sont par contre distinctes et ne sont utilisées que pour évaluer la performance de généralisation du modèle. Puisque $\mathscr{D}$ et $\mathscr{D}_{test}$ sont indépendants, on peut utiliser $\mathscr{D}$ pour trouver une fonction $f$ et l'évaluer de manière non baisée en calculant son erreur sur $\mathscr{D}_{test}$ avec l'équation 2.2.

Lorsqu'un modèle d'apprentissage tente de trouver la meilleure fonction $f_i$ pour résoudre une tâche, il les compare donc en calculant l'erreur empirique de chaque fonction sur l'ensemble de test.

Il existe aussi un troisième ensemble, l'**ensemble de validation**, qui est utilisé dans la situation où le modèle possède des **hyperparamètres**, c'est-à-dire des paramètres fixés à l'avance et qui ne sont pas appris à partir des données. Dans le cas des réseaux de neurones (voir section 2.2.2), un hyperparamètre est par exemple le nombre de neurones cachés à utiliser tandis que les paramètres appris seront le poids de chacun des neurones. Pour évaluer le modèle de manière non biaisée, on choisira les meilleurs hyperparamètres en calculant l'erreur sur l'ensemble de validation et l'on déterminera la performance de généralisation du modèle en calculant l'erreur sur l'ensemble de test.

### 2.1.4 Capacité et sur-apprentissage

Le concept de généralisation est fortement lié à la **capacité du modèle**. En effet, plus la capacité est élevée, plus le modèle sera en mesure d'apprendre par coeur, ou de faire un **sur-apprentissage** de l'ensemble d'entraînement, ce qui nuira à sa capacité à généraliser. Pour résoudre un problème complexe, le modèle aura par contre besoin d'avoir une capacité suffisante pour être en mesure de modéliser les données. La capacité du modèle doit donc être judicieusement réglée pour obtenir une bonne généralisation.

### 2.1.5 Apprentissage par descente de gradient

Nous avons parlé qu'un algorithme d'apprentissage voudra trouver la fonction $f$ qui minimise l'erreur empirique sur nos donnés d'entraînement. La méthode que nous utiliserons pour entraîner presque tous les modèles dont il sera question dans ce mémoire est la descente de gradient.

Considérons la fonction $f_\Theta$, où $\Theta$ représente l'ensemble des paramètres ajustables du modèle. On cherchera alors les paramètres $\hat{\Theta}$ qui minimisent l'erreur empirique $C$. Formellement, on a :

$$\hat{\Theta} = \arg\min_{\Theta} C(f_\Theta | \mathscr{D}). \tag{2.3}$$

La méthode que nous utiliserons pour déterminer les paramètres $\Theta$ est la **descente de gradient**. Essentiellement, cet algorithme optimise les paramètres $\Theta$ en calculant l'erreur du modèle $f_\Theta(\mathbf{x})$ et en modifiant chacun des paramètres dans la direction opposée au gradient. En effet, puisque le gradient d'une fonction pointe toujours dans la direction de la plus forte augmentation, on pourra minimiser cette fonction en suivant l'opposé du gradient. Pour chacun des paramètres $\theta_i \in \Theta$, on lui soustraira sa contribution au gradient, définie par $\frac{\partial}{\partial \theta_i} C(f_\Theta | \mathscr{D})$.

Il est important de noter que la fonction de coût $C_{0-1}$ (équation 2.2) n'est pas différentiable. Il n'est donc pas possible de l'utiliser comme fonction d'erreur que l'on optimisera par descente de gradient. En pratique, on devra donc utiliser d'autres fonctions d'erreur différentiables telles que l'**erreur quadratique** (équation 2.15) ou l'**erreur d'entropie croisée** (équation 2.10).

Notre fonction de coût globale représente la sommation de l'erreur pour chaque exemple de notre ensemble de données ($C = \sum_n C_n$). Effectuer la mise à jour des paramètres après avoir testé tous les exemples est appelé une **descente de gradient *batch***. Une autre approche, qui approxime le véritable gradient, est la **descente de gradient stochastique**, où on met à jour les paramètres du modèle après avoir testé chaque exemple. Il existe aussi un compromis entre ces deux approches, soit les *mini-batch*, où l'ensemble d'entraînement est divisé en plusieurs sous-ensembles. On procède alors au calcul du gradient ainsi qu'à la mise à jour des paramètres en considérant séquentiellement chaque sous-ensemble. Habituellement, pour de gros ensembles de données, la descente de gradient stochastique converge plus rapidement que l'approche *batch*.

Formellement, pour la descente de gradient stochastique, on aura que

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla C_n, \tag{2.4}$$

où $\tau$ est l'itération, $\eta$ est l'importance de la mise à jour des paramètres, appelée **taux d'apprentissage** et où le vecteur $\mathbf{w}$ contient les paramètres du modèle (plus de détails sur les paramètres à la section 2.1.6).

### 2.1.6 Notions préalables de classification

Un classifieur linéaire binaire trouve une **fonction discriminante** $g(\cdot)$ qui segmente l'espace de manière à séparer le mieux possible les points des deux classes $\{C_1, C_2\}$. La formulation la plus simple d'une telle fonction est :

$$g(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0 = w_0 + \sum_{i=1}^{d} w_i x_i, \tag{2.5}$$

où $\mathbf{w}$ est le **vecteur de poids** et $w_0$ est le **biais**. $\mathbf{w}$ représente la pente de la fonction de décision tandis que $w_0$ son décalage par rapport à l'origine.

Introduisons maintenant la **fonction de décision** $f(\cdot)$ qui est une fonction non linéaire défi-

nissant la véritable sortie du modèle. Concrètement, la fonction $g$ retourne un nombre réel et dans le cas de la classification, nous désirons obtenir une classe discrète $C_i$ comme sortie. Toujours dans un contexte de classification binaire, nous pouvons définir $f$ comme :

$$f(\mathbf{x}) = \begin{cases} C_2 & \text{si } g(\mathbf{x}) \geqslant 0, \\ C_1 & \text{si } g(\mathbf{x}) < 0. \end{cases} \tag{2.6}$$

### 2.1.7 Fonctions de bases

Considérons de plus des **fonctions de base** (*basis functions*) non linéaires qui ont pour but de transformer le vecteur d'attribut $\mathbf{x}$ pour le projeter dans un nouvel espace. En effet, un problème n'est peut-être pas linéairement séparable dans son espace d'origine, mais il le deviendra possiblement si on applique un vecteur de fonctions de base $\phi$ sur le vecteur d'attributs $\mathbf{x}$. Un classifieur qui opère sur $\phi(\mathbf{x})$ sera linéaire dans le nouvel espace et non linéaire par rapport à l'espace originel. L'utilisation de fonctions de base est facultative et dépend du problème. Effectuer un prétraitement des données, comme en effectuant l'extraction des attributs du fichier audio (voir section 2.3), peut par exemple être considéré comme l'application de fonctions de base. Certains modèles utiliseront des fonctions de base fixes tandis que d'autres les apprendront à partir des données. Apprendre les fonctions de bases augmente dramatiquement la capacité du modèle et sera nécessaire à la modélisation d'un espace fortement non linéaire.

## 2.2 Modèles d'apprentissage

La prochaine section couvre les modèles d'apprentissage qui sont utilisés dans les articles composant ce mémoire.

Nous traiterons tout d'abord de trois modèles dont l'apprentissage est à base de gradient, soit la régression logistique, les réseaux de neurones ainsi que les *stacked denoising autoencoders*, couverts de la section 2.2.1 à 2.2.3. Ils sont utilisés dans l'article présenté au chapitre 8. Une représentation graphique de ces trois modèles est présentée à la figure 2.1.

Nous couvrirons ensuite le *boosting* à la section 2.2.4, un meta-algorithme qui est construit à partir d'une série de modèles faibles, et qui est utilisé dans l'article décrit au chapitre 6.

### 2.2.1 La régression logistique

La régression logistique est, malgré son nom, un classifieur linéaire. Il s'agit du modèle le plus simple que nous utiliserons. Il s'agit d'un modèle probabiliste qui, dans le cas binaire, est défini comme suit :

$$f(\phi) = p(C_1|\phi) = \sigma(\mathbf{w}^T\phi + w_0), \tag{2.7}$$

où $\phi$ représente implicitement $\phi(\mathbf{x})$ et où $\sigma$ est la **fonction sigmoïde** définie par :

$$\sigma(a) = \frac{1}{1+\exp(-a)}. \tag{2.8}$$

La fonction $\sigma$ écrase toutes les valeurs qu'elle reçoit, ayant comme asymptotes les droites $y = 0$ et $y = 1$. On a donc que $\sigma : [-\inf, +\inf] \to [0,1]$. C'est pour cela que l'on peut interpréter la sortie comme une probabilité.

La fonction de décision de la régression logistique sera donc :

$$f(\phi) = \begin{cases} C_2 & \text{si } g(\phi) \geqslant 0.5, \\ C_1 & \text{si } g(\phi) < 0.5. \end{cases} \tag{2.9}$$

L'erreur que l'on voudrait optimiser durant l'entraînement est le nombre d'éléments mal classifiés, soit le coût $C_{0-1}$. Puisque ce coût n'est pas différentiable, nous minimiserons à la place la **log-vraisemblance négative** (*negative log-likelihood*), appelée l'**erreur d'entropie croisée**. Dans le cas binaire, ce coût est défini comme suit :

$$C_{ec}(f_\Theta(\mathbf{x})), \mathrm{t}) = -\mathrm{t}\log(f_\Theta(\mathbf{x})) - (1-\mathrm{t})\log(1-f_\Theta(\mathbf{x})), \tag{2.10}$$

où $\Theta$ contient les paramètres du modèles.

### 2.2.2 Les réseaux de neurones

Comme il a été mentionné préalablement, pour pouvoir séparer des données non linéairement séparables, la régression logistique doit utiliser des fonctions de base fixes qui doivent être déterminées au préalable. Cette manière de faire réduit beaucoup la capacité du modèle et une amélioration possible est d'apprendre les fonctions de bases, c'est-à-dire de les ajuster en fonction des données d'entraînement.

L'idée derrière les réseaux de neurones est d'utiliser une deuxième série de régresseurs logistique comme fonctions de base. On visualise cet emboîtement comme un réseau à plusieurs couches, comme le montre la figure 2.1 (centre).

Considérons le vecteur d'entrée $\mathbf{x} = (x_1, ..., x_d)$, où chaque $x_i$ correspond à un noeud de la couche d'entrée. En supposant un réseau pleinement connecté, on effectuera alors une série de $J$ combinaisons linéaires ayant la forme

$$a_j = \sum_{i=1}^{d} w_{ji}^{(1)} x_i + w_{j0}^{(1)}, \tag{2.11}$$

où $j = 1, ..., J$ et où l'indice $(1)$ signifie que nous sommes à la première couche du réseau. Dans la terminologie des réseaux de neurones, les valeurs $a_j$ sont appelées des **activations** et elles

sont passées dans une fonction non linéaire et différentiable appelée **fonction d'activation** $h(\cdot)$ donnant lieu à $z_j = h(a_j)$. Un choix fréquent pour la fonction $h$ est la fonction sigmoïde, définie à l'équation 2.8.

Nous avons mentionné que les réseaux de neurones sont similaires à la régression logistique, avec la différence qu'ils apprennent leurs fonctions de base à partir des données. Pour faire un parallèle, les valeurs dans $\mathbf{z}$, qui sont la sortie de la fonction d'activation dans le réseau de neurones, sont l'équivalent de la sortie des fonctions de bases $\phi$ dans l'équation 2.7, définissant la régression logistique. Dans un réseau de neurones, on appelle ces fonctions de bases apprises les *unités cachées* de la **couche cachée**. La similarité est évidente en regardant l'équation 2.12.

Une deuxième série de transformations ayant la même forme que l'équation 2.11 est ensuite appliquée sur $\mathbf{z}$ pour donner la sortie du réseau. Supposons un réseau ayant une couche cachée de $M$ unités, chaque noeud de sortie $k = 1, 2, ..., K$ est défini comme :

$$g_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)}\right) \tag{2.12}$$

$$= \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{d} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right). \tag{2.13}$$

Bien que la fonction sigmoïde ($\sigma$) est utilisée comme fonction d'activation de la couche de sortie à l'équation 2.13, notons que la fonction utilisée varie selon le type de problème. Pour la régression, il s'agira de l'identité, pour la classification binaire, on utilisera la fonction sigmoïde et pour la classification multiclasse, la fonction **softmax**, définie à l'équation 2.14, sera utilisée.

$$\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \tag{2.14}$$

L'algorithme de descente de gradient (voir section 2.1.5) utilisé pour entraîner les réseaux de neurones est la **rétropropagation du gradient** (*backpropagation*). En tirant avantage de la règle de dérivation en chaîne, il est possible de calculer la contribution de chaque paramètre $w \in \mathbf{W}$ à l'erreur, et donc d'ajuster leur valeur en conséquence.

### 2.2.3 Stacked denoising autoencoders

Les réseaux de neurones ne furent jusqu'à tout récemment pas utilisés avec plus d'une couche cachée. La performance des réseaux à plus d'une couche cachée était en effet décevante puisqu'ils semblaient tomber dans des minima locaux non optimaux et qu'une manière de les entraîner de façon efficace n'était pas connue.

De récents travaux [9, 42] ont cependant introduit une nouvelle stratégie, soit l'initialisation

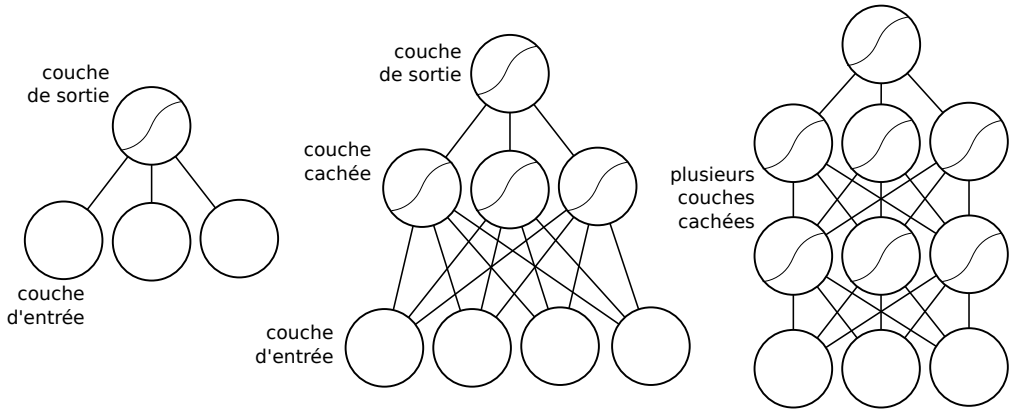


Figure 2.1 – Illustration de la régression logistique (gauche), d'un réseau de neurones (centre) et d'un réseau profond (droite), comme l'auto-encodeur débruiteur. Notons que la partie supérieure du réseau de neurones ainsi que du réseau profond est un régresseur logistique.

de chaque couche indépendamment de façon non supervisée. Ces avancées ont permis l'entraînement de réseaux profonds efficacement. Elles sont à la base du *stacked denoising autoencoder*[93] (SdA) qui est présenté dans cette section.

Nous présentons d'abord l'auto-encodeur de base, puis son extension *denoising*, qui est l'élément atomique du réseau profond que nous utilisons, le SdA.

#### 2.2.3.1 L'auto-encodeur de base

Un auto-encodeur est un réseau de neurones à une couche cachée utilisé pour apprendre une représentation compacte d'un ensemble de données. Supposons un vecteur d'entré $\mathbf{x} \in [0,1]^d$, le réseau tentera en effet de reproduire exactement le même vecteur en sortie en passant à travers une couche cachée $\mathbf{y} \in [0,1]^{d'}$. La clé est que la couche cachée aura une dimensionalité inférieure aux couches d'entrée et de sortie ($d' < d$) de sorte que le modèle devra apprendre comment conserver le maximum d'information possible dans cette représentation intermédiaire compacte.

L'auto-encodeur a la même architecture que le réseau de neurones (voir l'équation 2.12). Son entraînement représente par contre un problème de régression. Le coût que nous voulons minimiser est donc l'**erreur quadratique**, définie comme :

$$C(\mathbf{x}, \mathbf{z}) = ||\mathbf{x} - \mathbf{z}||^2, \tag{2.15}$$

où $\mathbf{z} \in [0,1]^d$ est la couche de sortie. Dans le contexte de l'auto-encodeur, l'erreur est appelée l'**erreur de reconstruction**, puisque l'on tente de reconstruire l'entrée.

#### 2.2.3.2 L'auto-encodeur débruiteur

L'auto-encodeur *débruiteur*[93] (**denoising autoencoder**) reprend l'idée de l'auto-encodeur et tente de rendre la représentation intermédiaire robuste à la destruction partielle de l'entrée. La motivation de cette approche est de tenter de forcer le modèle à apprendre une représentation encore plus générale des données, de sorte qu'elle soit extrêmement robuste au bruit.

La procédure d'entraînement consiste à utiliser un processus stochastique $\tilde{\mathbf{x}} \sim q_D(\tilde{\mathbf{x}}|\mathbf{x})$ qui prendra le vecteur d'entré $\mathbf{x} \in [0,1]^d$ et réglera au hasard $\nu d$ de ses éléments à 0, où $\nu$ est un hyperparamètre ajustable. L'auto-encodeur sera alors entraîné à reproduire $\mathbf{x}$ à partir de l'entrée incomplète $\tilde{\mathbf{x}}$, ce qui le forcera à véritablement trouver les régularités dans les données.

Les nouveaux vecteurs $\tilde{\mathbf{x}}$ sont générés de manière stochastique à chaque époque de l'entraînement. Comme notre objectif est de débruiter l'entrée plutôt que de la reconstruire exactement, la couche cachée n'a plus obligatoirement besoin d'être plus petite que l'entrée ($d' < d$) pour s'assurer que le modèle n'apprenne pas simplement l'identité, comme dans le cas de l'auto-encodeur de base. On pourra en effet avoir des couches cachées qui sont beaucoup plus grosses que l'entrée.

#### 2.2.3.3 Utilisation dans un réseau profond

Les auto-encodeurs *denoising* ont été utilisés avec succès comme éléments atomiques de réseaux de neurones profonds[93]. En empilant plusieurs auto-encodeurs, on obtient un SdA.

L'architecture consiste à utiliser la sortie de l'auto-encodeur de la $k$-ième couche comme entrée de l'auto-encodeur de la $(k+1)$-ième couche et de successivement les entraîner de manière non supervisé. Cet entraînement non supervisé sert d'initialisation au réseau profond et remplace l'initialisation aléatoire des poids généralement utilisée dans les réseaux de neurones. En procédant de cette manière, on initialise le réseau près d'une bonne solution, ce qui évite de tomber dans un mauvais minimum local lors de l'entraînement. Une fois que chaque couche a été initialisée, on exécute un entraînement supervisé à travers tout le réseau, comme on le ferait pour un réseau de neurones multicouche, en utilisant le véritable coût que l'on souhaite optimiser.

### 2.2.4 Boosting

Nous présentons maintenant deux algorithmes très différents de ceux à base de gradient présentés aux sections précédentes. En effet, tandis que les algorithmes précédents étaient basés sur l'optimisation de paramètres en suivant l'opposé du gradient d'une fonction d'erreur, l'idée derrière le *boosting* est de combiner une série de modèles très simples pour former un comité qui votera lorsque le modèle devra prendre une décision. À chaque itération, on choisit le modèle simple à ajouter en considérant sa performance sur les points de données que les modèles préalablement choisis ont eu le plus de difficulté à classifier correctement.

Les modèles de *boosting* que nous présentons sont AdaBoost ainsi qu'une de ses extension multi-classe, AdaBoost.MH, à la section 2.2.4.1. Nous regardons ensuite une version utilisant l'échantillonnage par rejet, FilterBoost, permettant de travailler avec de très gros ensembles de données, à la section 2.2.4.2.

#### 2.2.4.1 AdaBoost et AdaBoost.MH

**AdaBoost**[35], ou *adaptive boosting*, est un meta-algorithme qui combine plusieurs modèles *faibles* (*weak learners*) ensemble pour former un modèle dit *fort*. L'idée derrière cette méthode est de construire itérativement un ensemble de $J$ classifieurs faibles, ayant chacun un poids leur étant associé et de les faire *voter* lorsqu'on tente de classifier un vecteur d'entré. Un modèle fort entraîné sera donc un ensemble $\{(h^{(j)}, \alpha^{(j)})\}_{j=1}^{J}$, où $h^{(j)}$ représente un classifieur faible choisi à l'itération $j$ et $\alpha^{(j)}$ son poids associé. Le poids $\alpha^{(j)}$ est déterminé en considérant la performance du classifieur faible ; meilleur est sa performance, plus son vote aura d'importance.

Une des particularités d'AdaBoost est que l'ensemble de données n'est pas statique. En effet, un poids $w_n$, utilisé dans la fonction de coût, est associé à chaque élément de l'ensemble de données. Les poids sont d'abord initialisés à la valeur $1/N$, où $N$ est le nombre d'éléments dans l'ensemble. Par la suite, le poids de chaque élément est modifié en augmentant celui des éléments difficiles à classifier, c'est-à-dire ceux que les classifieurs précédents ont mal classifiés. En procédant ainsi, les classifieurs subséquents mettront plus d'effort à classifier ces points difficiles puisque leur contribution au coût sera plus grande. Le classifieur faible $j+1$ sera donc celui qui minimise la fonction de coût

$$h^{(j+1)} = \arg\min_{h' \in \mathscr{H}} \sum_{n=1}^{N} w_n I(\mathrm{t}^{(n)} \neq h'(\mathbf{x}^{(n)})), \tag{2.16}$$

où $\mathscr{H}$ est l'ensemble des classifieurs faibles considérés et où $\mathrm{t}^{(n)}$ est la véritable classe du point $\mathbf{x}^{(n)}$.

La sortie d'un modèle faible sera une classe telle que $h^{(j)}(\mathbf{x}) \in \{-1, 1\}$. La fonction discriminante après $i$ itérations, ou utilisant seulement les $i$ premiers classifieurs faibles, est donc :

$$g_i(\mathbf{x}) = \sum_{j=1}^{i} \alpha^{(j)} h^{(j)}(\mathbf{x}), \tag{2.17}$$

et la fonction de décision est quant à elle $f_i(\mathbf{x}) = \mathrm{signe}(g_i(\mathbf{x}))$.

**AdaBoost.MH**[77] est une extension multi-classe à AdaBoost qui utilise la stratégie **un contre tous** (*one-versus-all*). Étant donné un problème à $K$ classes, cette stratégie consiste à construire $K-1$ classifieurs, résolvant chacun un problème de classification binaire où il devra séparer les points de la classe $C_k$ des autres points.

#### 2.2.4.2 FilterBoost

**FilterBoost**[22] est une modification d'AdaBoost permettant de travailler avec des ensembles de données qui sont beaucoup trop gros pour entrer en mémoire. FilterBoost utilise en effet l'**échantillonnage par rejet** (*rejection sampling*) pour sélectionner un sous-ensemble de l'ensemble de données, appelé **minibatch**, à chaque itération.

L'idée est d'utiliser deux nouvelles entités, un *oracle* capable d'aller chercher des paires $(\mathbf{x}^{(n)}, \mathrm{t}^{(n)})$ de manière iid de notre ensemble de données, et un *filtre*, acceptant ou non les exemples que l'oracle génère. Tandis qu'AdaBoost ajustait la pondération des points de l'ensemble de données pour donner plus d'importance à ceux mal classifiés, FilterBoost inclura les exemples mal classifiés dans la *minibatch* avec une plus haute probabilité.

Quant on construit une *minibatch*, l'oracle choisi un exemple de l'ensemble de données, où la probabilité qu'il retourne un exemple positif ou négatif est de $1/2$. Cet exemple est ensuite passé au filtre qui l'accepte avec une probabilité

$$q_j(\mathbf{x}, \mathrm{t}) = \frac{1}{1 + \exp\left(\mathrm{t} \cdot g_j(\mathbf{x})\right)}, \tag{2.18}$$

où $g_j$ (équation 2.17) est la sortie du *booster* avec les $j$ classifieurs faibles déjà choisis, de sorte que les exemples mal classifiés aient plus de chances d'être inclus dans la *minibatch*.

**Algorithm 2.1** Pseudocode de $getEdge(f_k(\cdot))$

```
m ← 0, n ← 0, u ← 0,
for iter ← 1 to batchsize do
    (x, t) ← filter()
    n ← n + 1
    m ← m + I(f_k(x) = t)
    u ← m/n − 1/2
end for
return u
```

Une fois que $h^{(j+1)}$ a été choisi, on lui assigne son poids $\alpha^{(j+1)}$ en calculant son erreur sur une nouvelle *minibatch*. On utilise tout d'abord la fonction $\gamma = getEdge(f_{j+1}(\cdot))$, définie à l'algorithme 2.1, pour ensuite calculer $\alpha^{(j+1)}$ comme suit :

$$\alpha^{(j+1)} = \frac{1}{2} \ln\left(\frac{1/2 + \gamma}{1/2 - \gamma}\right). \tag{2.19}$$

Notons que nous avons traité de FilterBoost dans un contexte à deux classes. Des stratégies multiclasses analogues à celles utilisées pour AdaBoost ont été proposées, notamment dans [21].

## 2.3 Extraction d'attributs à partir de fichiers audio

### 2.3.1 La musique

La musique, le centre d'intérêt de nos travaux, est une forme d'art définie par *Le Robert*[75] comme « l'art de combiner des sons d'après des règles (variables selon les lieux et les époques), d'organiser une durée avec des éléments sonores. » La perception que nous en faisons est donc sous la forme de sons organisés[1]. On peut représenter la musique de diverses manières, par exemple en utilisant la théorie musicale, qui permet de décrire le fonctionnement de la musique occidentale en termes de rythme, mélodie et harmonie. Bien que des études se penchent sur l'utilisation de données symboliques liées à la théorie musicale, nous utilisons une approche basée sur l'analyse du signal musical lui-même, donc du son.

### 2.3.2 Le son

D'un point de vue physique, le son est une onde, c'est-à-dire une oscillation de la pression, qui nous est généralement transmise par l'air ambiant. Le son est donc une superposition d'ondes sonores de différentes fréquences ayant diverses caractéristiques comme l'amplitude et la phase. La fréquence représente la *hauteur* du son. Plus la fréquence est élevée, plus le son sera aigu. L'unité de mesure utilisée pour la fréquence est le hertz (Hz), qui représente le nombre d'oscillations que fait l'onde par seconde. Par exemple, la note *la* au piano a une fréquence de 440Hz. L'amplitude représente la force ou le volume du son tandis que la phase est le décalage temporel de l'onde.

Le son est un signal analogique qui doit être quantifié pour pouvoir être représenté dans un ordinateur. La représentation numérique non compressée qui est utilisée s'appelle la **modulation d'impulsion codée** (MIC, ou en anglais *pulse code modulation*). La MIC effectue un échantillonnage rapide et régulier du signal analogique de manière à en construire une approximation, comme le montre la figure 2.2. Les CD audio sont par exemple encodés avec la MIC et leur échantillonnage est fait à une fréquence de 44 100Hz, ou 44,1kHz.

Ce choix d'échantillonnage est loin d'être arbitraire. En effet, le théorème d'échantillonnage de Nyquist-Shannon nous dit que l'échantillonnage d'un signal doit être au moins le double de la fréquence maximale qu'il contient, sans quoi il ne pourra être fidèlement reproduit. Puisque les humains peuvent entendre des sons allant jusqu'à environ 20 kHz, un échantillonnage minimum de 40 kHz était nécessaire pour pouvoir représenter toutes les fréquences audibles par l'oreille humaine.

Tandis que certains attributs utilisent le signal dans le domaine temporel, c'est-à-dire dans sa représentation telle que donnée par la MIC, plusieurs autres travaillent sur sa forme fréquentielle.

[1] L'expression "organized sound" nous vient d'Edgard Varèse.

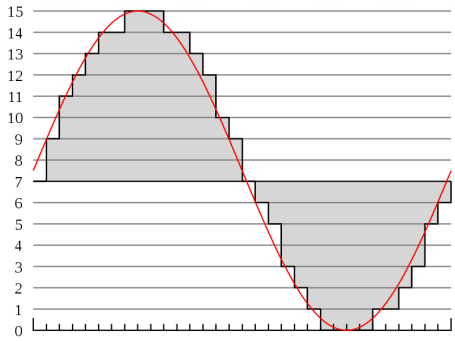


Figure 2.2 – Échantillonnage et quantification d'un signal analogique par une MIC 4-bit. La fonction continue représente le signal originel tandis que la fonction en escalier représente sa quantification. Plus la fréquence d'échantillonnage est élevée, plus les marches seront petites et plus la représentation du signal sera fidèle. Image réalisée par Ktims (http ://fr.wikipedia.org/wiki/Fichier :Pcm.svg)

En effet, on peut utiliser la transformée de Fourier discrète (*discrete Fourier transform* ou DFT) pour décomposer un signal numérique temporel en ses composantes sinusoïdales, et donc le faire passer dans le domaine des fréquences. La DFT retourne un vecteur complexe qui contient la quantité d'énergie et la phase présentes à chaque fréquence. La représentation visuelle habituelle de ce vecteur est appelée un spectrogramme. Un exemple est présenté à la figure 6.2c. Pour un traitement complet du traitement de signal ainsi que la transformée de Fourier, le lecteur est invité à consulter [36].

En effectuant des opérations supplémentaires sur le signal, soit dans le domaine temporel ou des fréquences, on peut obtenir plusieurs attributs par exemple, les MFCC (*Mel-frequency cepstral coefficients*) qui décrivent le timbre, ou les coefficients d'autocorrélation, qui décrivent le rythme. Plus d'informations sur les attributs généralement utilisés sont présentées dans [4].

Une fois les attributs extraits des fichiers audio, ils sont généralement concaténés ensemble pour former un vecteur par chanson qui sera la représentation utilisée pour réaliser l'apprentissage.

# CHAPITRE 3

# APERÇU DU PREMIER ARTICLE

**Generating Transparent, Steerable Recommendations from Textual Descriptions of Items.**
Stephen Green, Paul Lamere, Jeffrey Alexander et François Maillet
*Proceedings of the 3rd ACM Conference on Recommender Systems*, New York, USA, 2009

## 3.1 Contexte

Il existe plusieurs types de systèmes de recommandation musicale[24], soit le filtrage collaboratif, la recommandation basée sur le contenu, basée sur le contexte ou les méthodes hybrides.

L'article inclut dans le prochain chapitre présente une méthode pour effectuer des recommandations musicales à partir d'étiquettes décrivant les artistes et les chansons. Nous considérons qu'il s'agit d'un modèle hybride étant donné que les étiquettes que nous utilisons proviennent à la fois d'un contexte social ainsi que d'un modèle d'apprentissage appliquant automatiquement des étiquettes en se basant sur le contenu musical.

Pour mieux illustrer les avantages d'une telle méthode, nous présentons d'abord une très brève introduction au filtrage collaboratif.

### 3.1.1 Recommandations par filtrage collaboratif

Le **filtrage collaboratif** (FC)[76] consiste à effectuer des recommandations en se basant sur les données d'interactions qu'ont les utilisateurs avec les items. On peut s'intéresser à deux types d'interactions, soit des interactions passives ou actives. Un exemple d'interaction active est un système de classement où les utilisateurs peuvent assigner un score, par exemple de 1 à 5, aux items auxquels ils accèdent. Un exemple d'interaction passive est simplement la fréquence d'accès aux différents items. Les utilisateurs de Last.fm peuvent en effet installer un logiciel, le *Last.fm Scrobbler*[1], qui envoie automatiquement le titre des chansons qu'ils écoutent aux serveurs de Last.fm.

Une fois que l'on a récolté une grande quantité de données d'interaction, on peut construire une représentation des utilisateurs et des items basée sur ces données et déterminer quels utilisateurs et items sont similaires entre eux. La représentation d'un utilisateur serait donc un vecteur contenant le nombre de fois qu'il a écouté chaque artiste, tandis que la représentation d'un artiste serait le nombre de fois qu'il a été écouté par chaque utilisateur. Cela nous permet d'offrir des recommandations du type : *Les utilisateurs qui ont aimé l'artiste X ont aussi aimé l'artiste Y.*

[1] Last.fm Scrobbler : `http://www.last.fm/download`

Ce type de système est performant en présence d'une très grande quantité de données. Des compagnies comme Amazon ou Netflix, qui possèdent une quantité astronomique de données, peuvent donc utiliser un système de FC et obtiendront de très bons résultats. Par exemple, l'équipe qui gagna le *Netflix Price*[10] utilisa une approche purement basée sur le FC[44].

Par contre, quand on possède une quantité de données insuffisante, la performance en souffre. Il est extrêmement difficile d'amorcer un nouveau système et il est aussi difficile d'introduire un nouvel item dans un système déjà en fonction. Puisqu'aucun utilisateur n'a accédé à un nouvel item, il ne sera similaire à aucun autre item et ne sera pas recommandé. Dans un contexte où on découvre énormément d'items via les recommandations des sites, si un item inconnu n'est pas recommandé, il ne sera pas découvert très efficacement et restera donc inconnu. Cela est appelé le **problème du démarrage à froid** (*cold start problem*).

De plus, ces systèmes sont vulnérables au bias de popularité (*popularity bias*), aussi appelé l'effet *Harry Potter* ou *Coldplay*[49]. Certains items très populaires (ie. : les livres de Harry Potter) sont achetés avec beaucoup d'autres items qui ne leur sont pas nécessairement reliés. Dû au très grand nombre de transactions incluant ces items très populaires, ils deviennent similaires à presque tous les autres items aux yeux du système de recommandation. Ce dernier les recommande alors d'une manière qui semble aléatoire et mine la confiance des utilisateurs face au système.

### 3.1.2 Recommandations par méthode hybride

Le modèle que nous présentons utilise une *aura* textuelle, communément appelée un **nuage d'étiquettes** (*tag cloud*), comme représentation des items. Il s'agit d'une liste pondérée d'étiquettes où celles ayant un plus gros poids représentent de manière plus importante l'item. La représentation visuelle d'un nuage d'étiquettes habituellement utilisée est présentée à la figure 4.1.

Les étiquettes que le système utilise proviennent de plusieurs sources, par exemple du site Last.fm, de l'article Wikipedia décrivant chaque artiste ainsi que d'un algorithme d'application automatique d'étiquettes appelé *Autotagger*, que nous couvrirons au chapitre 5. Grâce à *Autotagger*, notre méthode peut recommander des artistes complètement inconnus puisque l'on peut inférer les étiquettes les plus probables à partir de l'audio de leurs chansons, réglant ainsi le ***cold start problem***.

Un système basé sur le contenu est aussi fondamentalement immunisé contre le bias de popularité puisque les données d'interactions ne sont pas prises en considération pour effectuer les recommandations.

## 3.2 Contributions

Cet article propose une approche unifiée qui possède plusieurs avantages sur les méthodes basées sur le FC, en plus de ceux énoncés à la section précédente. Notre système offre en effet des recommandations transparentes, personnalisables et est construit pour être extensible.

Pour déterminer quels items sont similaires entre eux, nous comparons leur nuage d'étiquettes. Les nuages sont tout d'abord repondérés en utilisant une méthode issue du traitement de langue naturelle appelée TF×IDF (*term frequency-inverse document frequency*). Considérons les étiquettes comme des mots $t$ associés à des documents $d \in D$ (les artistes). La fréquence d'un mot (*term-frequency*) est définie comme :

$$\mathrm{tf}_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}, \tag{3.1}$$

où $n_{i,j}$ est le nombre d'occurrences du mot $t_i$ dans le document $d_j$. Dans notre cas, il s'agira du poids de l'étiquette dans les données de Last.fm. Le dénominateur représente la sommation des occurrences de tous les mots dans un document $d_j$.

La fréquence inverse de document (*inverse document frequency*) représente l'importance relative d'un mot dans la collection. Par exemple, l'étiquette *rock* est extrêmement fréquente et n'est donc pas très distinctive. On voudra donc diminuer son importance. On déterminera la fréquence inverse de document comme suit :

$$\mathrm{idf}_i = \log\left(\frac{|D|}{1 + \sum_{d \in D} I(t_i \in d)}\right), \tag{3.2}$$

où $\sum_{d \in D} I(t_i \in d)$ représente le nombre de documents contenant le mot $t_i$. On peut alors calculer TF×IDF avec l'équation suivante :

$$(\mathrm{tf} - \mathrm{idf})_{i,j} = \mathrm{tf}_{i,j} \times \mathrm{idf}_i. \tag{3.3}$$

Essentiellement, cette repondération donne plus d'importance aux étiquettes distinctives pour chaque artiste, c'est-à-dire celles qui les différencient des autres. En effet, elle augmente l'importance des étiquettes ayant une haute fréquence pour peu d'artistes tout en ayant une basse fréquence pour le reste des artistes.

Nous pouvons ensuite comparer les nuages en utilisant la similarité cosinus, définie à l'équation 6.7.

### 3.2.1 Transparence

Puisque les recommandations sont effectuées en comparant des nuages d'étiquettes, le système peut *expliquer* à l'utilisateur pourquoi un item est recommandé. En effet, on utilise un

nuage d'étiquettes de similarité (*overlap tagcloud*), où les étiquettes communes entre les deux items sont incluses et où leur taille est proportionnelle à leur degré de similarité. Un nuage d'étiquettes différentiel (*difference tagcloud*) est aussi inclus au système. Ce nuage met en évidence ce qui différencie deux items. Ces deux nuages sont présentés à la figure 3.1.

Un utilisateur peut donc comprendre pourquoi un item lui est recommandé. On dira de ce système qu'il est transparent. Comme il est expliqué dans [82], un système transparent est très important puisque cela augmente la confiance de l'utilisateur et sa satisfaction envers le système.

### 3.2.2 Personnalisation

Notre système est aussi personnalisable (*steerable*), de sorte que l'utilisateur peut indiquer au système de recommandation le type de musique qui l'intéresse présentement. L'interface utilisateur que nous avons développée pour le *Music Explaura*[80] (voir figure 3.2) permet de construire un nuage d'étiquettes personnalisable en lui ajoutant des étiquettes et en leur assignant une taille. Il est aussi possible de marquer certaines étiquettes comme étant collantes (ce qui oblige les items recommandés à avoir cette étiquette) ou de leur donner un poids négatif (ce qui agit comme un filtre sur les recommandations).

Un exemple simple de l'utilité d'un tel système est d'ajouter toutes les étiquettes des *Beatles* dans un nuage personnalisable et d'ajouter ensuite une étiquette comme *voix féminine*. On se fera alors recommander de la musique qui ressemble à celle des *Beatles*, mais avec une chanteuse au lieu d'un chanteur.

En combinant la personnalisation à la transparence, l'utilisateur peut comprendre pourquoi les items sont recommandés et modifier ce qui est à l'origine des recommandations pour orienter le système dans la direction de ce qui l'intéresse vraiment.

## 3.3 Commentaires

La contribution de l'auteur à cet article fut réalisée dans le cadre de deux stages à *Sun Microsystems Laboratories*. Il a travaillé principalement sur le *Music Explaura*[80] ainsi qu'à développer le concept des recommandations personnalisables.

Le système est propulsé par le *Aura Datastore*, aussi développé chez Sun, qui est un système distribué hautement extensible. *Aura* permet de calculer la similarité entre un très grand nombre de vecteurs d'étiquettes extrêmement rapidement.

Malgré que le sondage auprès d'usagers ait relevé un peu de confusion vis-à-vis l'interface du nuage d'étiquettes personnalisable, aucune autre approche n'a été proposée dans la littérature pour permettre à l'utilisateur de donner de la rétroaction au système avec autant de flexibilité et de puissance. Nous avons aussi réutilisé ce concept dans un autre article inclus dans ce mémoire


Figure 3.1 – a) Nuage d'étiquettes de similarité pour Jimi Hendrix and Eric Clapton. Les étiquettes communes sont incluses dans le nuage et leur taille est proportionnelle à leur degré de similarité. Leur ordre est aléatoire. b) Nuage d'étiquettes différentiel pour Jimi Hendrix et Eric Clapton. Les étiquettes de Hendrix apparaissent en premier, avec celles qui le différencient le plus de Clapton ayant une police plus grosse. Plus la police des étiquettes rapetisse, plus les étiquettes sont communes aux deux. Quand elle recommence à grossir, on tombe dans les étiquettes qui différencient Clapton de Hendrix.

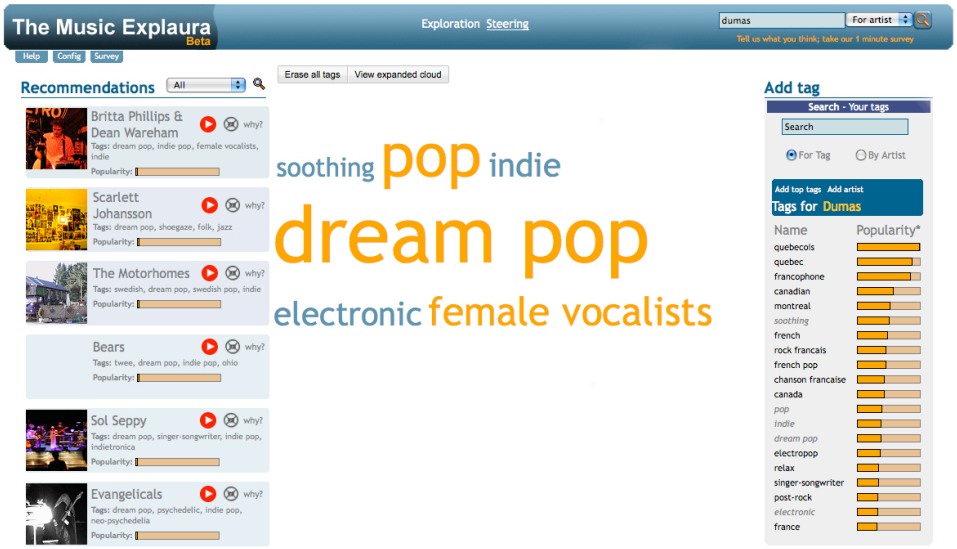


Figure 3.2 – Nuage d'étiquettes personnalisable du *Music Explaura*[80]. Les utilisateurs peuvent ajouter des étiquettes à partir du panneau latéral de droite. Le centre constitue leur nuage personnalisable, où ils peuvent modifier la taille des étiquettes. Ils obtiennent des recommandations sur le panneau latéral de gauche.

et qui permet de donner les mêmes possibilités à l'utilisateur pour la génération de listes d'écoute (voir chapitre 7).

La seule autre interface de recommandation personnalisable dont nous avons connaissance est un prototype de Last.fm appelé *Multi-tag search*[52]. L'interface, relativement plus simple que la nôtre, ne permet par contre pas d'assigner différents poids aux étiquettes et ne met pas à jour les recommandations en temps réel dès qu'une modification est apportée.

# CHAPITRE 4

# GENERATING TRANSPARENT, STEERABLE RECOMMENDATIONS FROM TEXTUAL DESCRIPTIONS OF ITEMS

## 4.1 Abstract

We propose a recommendation technique that works by collecting text descriptions of items and using this *textual aura* to compute the similarity between items using techniques drawn from information retrieval. We show how this representation can be used to explain the similarities between items using terms from the textual aura and further how it can be used to steer the recommender. We describe a system that demonstrates these techniques and we'll detail some preliminary experiments aimed at evaluating the quality of the recommendations and the effectiveness of the explanations of item similarity.

## 4.2 Introduction

One of the problems faced by current recommender systems is explaining why a particular item was recommended for a particular user. Tintarev and Masthoff [82] provide a good overview of why it is desirable to provide explanations in a recommender system. Among other things, they point out that good explanations can help inspire trust in a recommender, increase user satisfaction, and make it easier for users to find what they want.

If users are unsatisfied with the recommendations generated by a particular system, often their only way to change how recommendations are generated in the future is to provide thumbs-up or thumbs-down ratings to the system. Unfortunately, it is not usually apparent to the user how these gestures will affect future recommendations.

Finally, many current recommender systems simply present a static list of recommended items in response to a user viewing a particular item. Our aim is to move towards exploratory interfaces that use recommendation techniques to help users find new items that they might like and have the exploration of the space part of the user experience.

In this paper, we'll describe a system that builds a *textual aura* for items and then uses that aura to recommend similar items using textual similarity metrics taken from text information retrieval. In addition to providing a useful similarity metric, the textual aura can be used to explain to users why particular items were recommended based on the relationships between the textual auras of similar items. The aura can also be used to *steer* the recommender, allowing users to explore how changes in the textual aura create different sets of recommended items.

### 4.3 Related Work

Tintarev and Masthoff's [82] survey of explanations in recommender systems provides an excellent description of the aims of explanations and what makes a good explanation. We will adopt their terminology wherever possible. Zanardi and Capra [98] exploit tags and recommender system techniques to provide a social ranking algorithm for tagged items. Their approach uses straight tag frequency, rather than more well-accepted term weighting measures.

Vig *et al*. [92] use social tags to generate descriptions of the recommendations generated by a collaborative filtering recommender. Although they explicitly forego "keyword-style" approaches to generating recommendations, we believe that it is worthwhile to try techniques from information retrieval that are known to provide good results.

Wetzker *et al*. [96] use Probabilistic Latent Semantic Analysis (PLSA) to combine social tags and attention data in a hybrid recommender. While PLSA is a standard information retrieval technique, the dimensionality reduction that it generates leads to a representation for items that is difficult to use for explanations.

### 4.4 The Textual Aura

The key aspect of our approach is to use an item representation that consists of text about the item. For some item types like books or blog posts, this could include the actual text of the item, but for most items, the representation will mainly consist of text crawled from the Web. In our representation, each item in the system is considered to be a document. The "text" of the document is simply the conglomeration of whatever text we can find that pertains to the item. Because we may collect text about the item from a number of sources, we divide an item's document into fields, where each field contains text from a particular source.

The representation we use for an item is a variant of the standard vector space representation used in document retrieval systems. Rather than using a single vector to represent a document, the system keeps track of separate vectors for each of the fields that make up the document. This allows us to calculate document similarity on any single field or any combination of fields.

Once one is treating items as documents in this fashion, it becomes easy to do other things like treat individual social tags as documents whose words are the items to which the social tag has been applied. Similarly, a user can be modeled as a document whose words are the social tags that the user has applied to items, allowing us to compute user-user and user-item similarity.

An advantage of a vector space representation is that there are well known techniques for dealing with a number of phenomena that are typically encountered when dealing with tags. For example, the issues of tag quality and tag redundancy encountered by Vig *et al*. [92] are usually handled quite well using the standard vector space model for document retrieval. Another

advantage is that there are also well-known techniques for building and scaling systems that use such a representation.

Our current work has focused on recommending musical artists. The data sources that we use to represent a musical artist include social tags that have been applied to the artist at Last.fm[53] and the Wikipedia entry for the artist. Figure 4.1 shows a portion of the textual aura for Jimi Hendrix derived from the tags applied to him at Last.fm as a tag cloud. Unlike typical tag clouds, where the size of a tag in the cloud is proportional to the frequency of occurrence of the tag, here the size of a tag is proportional to it's term weight in the vector space representation for the artist.

By tying the size of a tag in the cloud to the term weight, we are giving the user an indication of what kind of artists we will find when we look for similar artists. In this case, we will tend to find a *guitar virtuoso*, who plays *blues* influenced *psychedelic rock*, as these are the largest tags and therefore the tags with the most influence over the similarity calculation.

### 4.4.1 Generating Recommendations

We can use the textual aura in a number of recommendation scenarios. We can find similar artists based on a seed artist's aura, or we can find artists that a user might like based on tags that they have applied or based on the combined aura of their favorite artists. We allow users to save tag clouds and use them to generate recommendations over time.

Since the implementations of these recommendation scenarios are based on similar techniques, we'll illustrate the general approach by explaining the specific case of finding artists that are similar to a seed artist. First, we retrieve the document vector for the seed item. This may be a vector based on a single field of the item's document or a composite vector that incorporates information for a number of fields.

In any case, for each of the terms that occurs in the vector for the seed item, we retrieve the list of items that have that term in their textual aura. As each list is processed, we accumulate a per-item score that measures the similarity to the seed item. This gives us the set of items that


Figure 4.1: The textual aura for Jimi Hendrix

have a non-zero similarity to the seed item. Once this set of similar items has been built, we can select the top $n$ most-similar items. This selection may be done in the presence of item filters that can be used to impose restrictions on the set of items that will ultimately be recommended to a user.

For example, we can derive a popularity metric from the listener counts provided by Last.fm and use this metric to show a user artists that everyone else is listening to or that hardly anyone else is listening to.

### 4.4.2 Explanations and Transparency

Given the auras for two items, it is straightforward to determine how each of the terms in the textual aura for a seed item contributes to the similarity between the seed item and a recommended item. For items $A$ and $B$, and the vectors used to compute their similarity, we can build three new tag clouds that describe both how the items are related and how they differ.

We can build an *overlap* tag cloud from the terms that occur in the vectors for both items. The tags in this cloud will be sized according to the proportion of the similarity between $A$ and $B$ contributed by the term. This explanation of the similarity between two items is directly related to the mechanism by which the similarity was computed. We can also build two *difference* clouds that show the terms in $A$ and $B$ that are not in the set of overlapping terms. The difference cloud can be used to indicate to a user how a recommended item differs from the seed item, which may lead them into an attempt to steer the recommendation towards a particular aspect.

### 4.4.3 Steerable Recommendations

In addition to providing a straightforward way of generating explanations of the recommendations made, our approach offers an equally straightforward way of *steering* the recommendations.

We allow the user to directly manipulate the document vector associated with an item and use their manipulations to interactively generate new sets of recommended items on the fly. If the vector for an item is presented as a tag cloud, then the user can manipulate this representation by dragging tags so that the tags increase or decrease in size. As with any other recommendation, we can provide an explanation like the ones described above that detail why a particular item was recommended given a particular cloud.

The user can also make particular tags *sticky* or *negative*. A sticky tag is one that must occur in the textual aura of a recommended artist. A sticky tag will still contribute some portion of the similarity score, since it appears in both the seed item and in the recommended item. A negative tag is one that must not occur in the textual aura of a recommended artist. Negative tags are used as a filter to remove artists whose aura contains the tag from the results of the similarity

computation.

The overall effect of the steerability provided to the users is that they can begin to answer questions like “Can I find an artist like Britney Spears but with an indie vibe?” or “Can I find an artist like The Beatles, but who is recording in this decade?”

### 4.4.4 The Music Explaura

As part of the AURA project in Sun Labs, we have developed the Music Explaura [80], an interface for finding new musical artists that incorporates all of the techniques described above. Users can start out by searching for a known artist, by searching for an artist with a particular tag in their aura, or by searching for a tag and seeing similar tags.

The aim is to provide as much context as possible for the user. In addition to the tag cloud for the artist (based on Last.fm tags by default) and the top 15 most-similar artists to the seed artist, we display a portion of the artist’s biography from Wikipedia, videos crawled from YouTube, photos crawled from Flickr, albums crawled from Amazon, and events crawled from Upcoming.

## 4.5 Evaluating Recommendations

To get a better understanding of how well the aura-based recommendations perform, we conducted a web-based user survey that allowed us to compare the user reactions to recommendations generated by a number of different recommenders. We compared two of our research algorithms, our aura-based recommender and a more traditional collaborative filtering recommender, to nine commercial recommenders. The aura-based system used a data set [47] consisting of about 7 million tags (with 100,000 unique tags) that had been applied to 21,000 artists. The CF-based system generated item-item recommendations based on the listening habits of 12,000 Last.fm listeners. The nine commercial recommenders evaluated consist of seven CF-based recommenders, one expert-based recommender and one hybrid (combining CF with content-based recommendation). We also included the recommendations of five professional critics from the music review site Pitchfork [73].

To evaluate the recommenders we chose the simple recommendation task of finding artists that are similar to a single seed artist. This was the only recommendation scenario that was supported by all recommenders in the survey. We chose five seed artists: The Beatles, Emerson Lake and Palmer, Deerhoof, Miles Davis and Arcade Fire. For each recommender in the study, we retrieved the top eight most similar artists. We then conducted a Web-based survey to rank the quality of each recommended artist. The survey asked each participant in the survey to indicate how well a given recommended artist answers the question “If you like the seed artist you might like *X*.” Participants could answer “Excellent” “Good” “Don’t Know” “Fair” or “Poor”. Two hundred individuals participated in the survey, contributing a total of over ten

| System | Average Rating | Relative Precision | Novelty |
|---|---|---|---|
| Sun aura | 4.02 | 0.49 | 0.31 |
| CF-10M | 3.68 | -0.02 | 0.24 |
| CF-2M | 3.50 | -0.16 | 0.28 |
| Sun-CF-12K | 3.48 | -0.38 | 0.26 |
| CF-1M | 3.26 | -0.47 | 0.23 |
| CF-100K | 2.59 | -1.01 | 0.32 |
| Expert | 2.06 | -1.17 | 0.58 |
| Music Critic | 2.00 | -1.26 | 0.49 |
| CF-1M | 1.82 | -1.18 | 0.38 |
| Hyb-1M | 0.89 | - 3.31 | 0.57 |
| CF-10K | 0.82 | -4.32 | 0.47 |
| CF-10K | -2.39 | -13.18 | 0.48 |

Table 4.1: Survey Results. CF-*X* represents a collaborative filtering-based system with an estimated number of users *X*

thousand recommendation rankings. We used the survey rankings to calculate three scores for each recommender:

**Average Rating** The average score for all recommendations based on the point assignment of 5, 1, 0, -1 and -5 respectively for each Excellent, Good, Don't Know, Fair and Poor rating. This score provides a ranking of the overall quality of the recommendations.

**Relative Precision** The average score for all recommendations based on the point assignment of 1, 1, 0, -1 and -25 respectively for each Excellent, Good, Don't Know, Fair and Poor rating. This score provides a ranking for a recommender's ability to reject poor recommendations.

**Novelty** The fraction of recommendations that are unique to a recommender. For instance, a value of 0.31 indicates that 31 percent of recommendations were unique to the particular recommender.

Table 4.1 shows the results of the survey. Some observations about the results: CF-based systems with larger numbers of users tend to have higher average ratings and relative precision. Somewhat surprisingly, human-based recommenders (Expert and Music Critic) do not rate as well as larger CF-based recommenders, but do tend give much more novel recommendations. Poorly rated recommenders tend to have a higher novelty score. The aura-based recommender provides good average rating and relative precision results while still providing somewhat novel recommendations.

It is important not to draw too many conclusions from this exploratory evaluation. The recommendation task was a simple one using popular artists, the number of participants was relatively small and the participants were self-selected. Nevertheless, the survey does confirm that the aura-based approach to recommendation is a viable approach yielding results that are competitive with current commercial systems.

### 4.5.1 The Music Explaura

In order to gain preliminary insight into the effectiveness of our approach to transparent, steerable recommendations, we conducted a small scale qualitative usability evaluation using The Music Explaura, contrasting it against other music recommendation sites.

We asked users to evaluate The Explaura's recommendations with respect to the familiarity and accuracy of the recommendations, their satisfaction with the recommendations, how novel the recommended artists were, the transparency and trustworthiness of the recommender, and the steerability of the recommendations.

Ten participants were recruited from the student population of Bentley University and musician forums on craigslist. The participants were all web-savvy, regular Internet users.

The Explaura interface presents a number of interaction paradigms very different from familiar interaction conventions, creating a pronounced learning curve for new users. Users' ratings of different sites' recommendations appeared to largely correspond to how much they agreed with the recommendation of artists that were familiar to them. Included in this assessment of the quality of the recommendations was often an assessment of rank.

Ten of ten users agreed, though not always immediately (some required prompting to investigate the similarity tag clouds) that they understood why Explaura recommended the items it did, and that the list of recommendations made sense to them.

It was clear that the value of the tag clouds as an explanation is highly vulnerable to any perception of inaccuracy or redundancy in the tag cloud. Also, sparse tag clouds produce more confusion than understanding. Almost none of the participants immediately grasped the meaning of the tag clouds. The general first impression is of confusion. Furthermore, no one expected to be able to manipulate the cloud.

Despite these problems, the users continually expressed the desire to limit and redefine the scope of an exploration. When the concept is presented, virtually all users express surprise, interest, and pleasure at the idea that they can do something with the results.

## 4.6 Future Work and Conlusions

The current system uses the traditional bag-of-words model for the tags. While this has provided some worthwhile results, it seems clear that we should be clustering terms like *canada*

and *canadian* so that their influence can be combined when generating recommendations and explanations. At the very least, combining such terms should lead to less confusion for users.

We're interested in generating more language-like descriptions of the similarities and differences between items. It seems like it should be possible to use the term weightings along with language resources like WordNet to generate Pandora-like descriptions of the similarity tag clouds, which may make them more approachable for new users.

Our ultimate aim is to provide hybrid recommendations that include the influence of the textual aura as well as that of collaborative filtering approaches. An obvious problem here is how to decide which approach should have more influence for any given user or item.

The textual aura provides a simple representation for items that can produce novel recommendations while providing a clear path to a Web-scale recommender system. The initial evaluation of the quality of the recommendations provided by an aura-based recommender provides strong evidence that the technique has merit, if we can solve some of the user interface problems.

Our initial usability study for the Music Explaura showed that the tag cloud representation for the artist can be confusing at first viewing. While the users expressed a real desire to explore the recommendation space via interaction with the system, the current execution needs further usability testing and interface design refinement to enhance user acceptance of the model.

**Acknowledgments**

Thanks to Professor Terry Skelton of Bentley University.

# CHAPITRE 5

# APERÇU DU DEUXIÈME ARTICLE

**Autotagger : a model for predicting social tags from acoustic features on large music databases.**
Thierry Bertin-Mahieux, Douglas Eck, François Maillet et Paul Lamere
*Journal of New Music Research, special issue : "From genres to tags : Music Information Retrieval in the era of folksonomies*.", volume 37(2), pages 151 à 165, 2008

## 5.1 Contexte

Comme il a été préalablement mentionné, l'utilisation d'étiquettes dans la description d'artistes et de chansons est devenue très populaire ces dernières années sur Internet. Les étiquettes sont d'abord et avant tout utilisées par les utilisateurs, mais l'espace qu'elles engendrent est aussi utilisé par divers algorithmes. En effet, plusieurs systèmes de recommandation musicale utilisent les étiquettes appliquées aux divers artistes comme une mesure de similarité entre eux (voir chapitre 4).

Il existe divers moyens[84] de récolter des étiquettes. Entre autres, des sondages, des jeux en ligne, des communautés musicales comme Last.fm ou en appliquant des techniques de traitement du langage naturel sur des documents web reliés aux artistes. Une autre méthode, celle que nous décrivons ici, consiste à utiliser un modèle d'apprentissage pour générer automatiquement des étiquettes à partir des fichiers sonores.

La motivation première de l'utilisation d'un modèle d'apprentissage est de minimiser l'impact du **problème du démarrage à froid** (voir section 3.1.1), c'est-à-dire le manque d'étiquettes pour les artistes méconnus ou nouveaux. En effet, un artiste qui n'est pas connu ne sera pas beaucoup écouté et donc peu d'utilisateurs lui auront appliqué des étiquettes. De la même manière, les documents web parlant de cet artiste seront rares.

Une approche alternative utilisée en industrie et qui s'apparente à un sondage est d'utiliser un groupe d'experts, comme le fait le *Music Genome Project*, sur lequel la radio internet Pandora[1] est basée[84]. Chaque chanson accessible dans leur système est écoutée par des experts en musique qui doivent remplir un questionnaire et assigner entre 150 à 500 étiquettes aux chansons. Dans le contexte où la quantité de nouvelles chansons accessibles sur la toile augmente de manière exponentielle et que la tendance ne fera que s'accélérer, une telle approche est humainement impraticable si l'on souhaite véritablement pouvoir exposer nos utilisateurs aux artistes

[1]Radio internet Pandora : `http://www.pandora.com`

moins populaires se trouvant dans la queue de la longue traîne ; il y a simplement trop de musique à écouter.

## 5.2 Contributions

Dans cet article, nous décrivons *Autotagger*, un algorithme d'application automatique d'étiquettes à des chansons à partir de leur fichier audio en utilisant l'algorithme d'apprentissage *Filterboost* (voir section 2.2.4.2). Le problème est formulé comme une tâche de classification où nous entrainons 360 modèles, chacun associé à une étiquette particulière et pouvant déterminer si cette étiquette s'applique à une chanson donnée ou non. Les expériences sont réalisées sur une base de données d'environ 100,000 chansons.

L'avantage notable de notre approche est son extensibilité. En effet, le modèle présenté est une extension de [29], qui utilise AdaBoost. En tirant avantage de l'échantillonnage par rejet ainsi que d'une stratégie d'apprentissage *en ligne*, Filterboost peut fonctionner avec des bases de données extrêmement grosses.

### 5.2.1 Corrélations entre les étiquettes

De plus, nous explorons deux façons de tirer avantage des corrélations entre les étiquettes provenant de Last.fm pour améliorer la qualité des étiquettes que l'on génère. Puisque nous entrainons un modèle différent pour chaque étiquette, chaque modèle ne peut pas utiliser l'information provenant des autres étiquettes. Par exemple, une chanson étiquetée *alternative rock* a probablement de très fortes chances d'être aussi étiqueté *indie rock* et *rock*. En considérant chaque étiquette comme indépendante, nous rendons la tâche d'apprentissage plus difficile.

La première manière que nous explorons est d'entrainer un deuxième ensemble de classifieurs (*second-stage learners*) en utilisant la sortie des premiers comme attributs d'entrée. L'apprentissage du premier niveau se fait donc à partir de caractéristiques audio, tandis que le deuxième niveau se fait en utilisant les 360 probabilités générées par les classifieurs du premier niveau.

La deuxième approche que nous avons testée consiste à calculer la corrélation empirique qui existe entre les étiquettes sociales utilisées pour l'entrainement. Nous avons ensuite ajusté la prédiction du modèle pour chaque étiquette en la combinant avec celles des autres étiquettes, proportionnellement à leur corrélation dans les données sociales .

Malgré qu'elle fut légère, nous avons observé une amélioration de la performance sur les mesures de performance utilisées quand ces ajustements sont faits.

### 5.2.2 Comparaison à l'approche GMM

Nous comparons finalement notre modèle à une approche basée sur un mélange de gaussiennes hiérarchique (*hierarchical gaussian mixture model (GMM)*) en utilisant un ensemble de données commun, soit l'ensemble *Computer Audition Lab 500-Song* (CAL500)[85, 86].

Un des objectifs est d'évaluer la différence entre leur approche qui est basée sur *peu de données fiables* et la nôtre basée sur *beaucoup de données bruitées*. En effet, l'ensemble de données *CAL500* est beaucoup plus petit que les données sociales avec lesquelles nous avons travaillé. Il a été construit en payant un groupe de 66 étudiants pour qu'ils appliquent des étiquettes sur 500 chansons. Un total de 1,708 étiquettes furent appliquées dans *CAL500*, contre plus de 7 millions dans notre ensemble de données issue de Last.fm. Les étiquettes de *CAL500* utilisent aussi un vocabulaire prédéterminé, ce qui contraste avec nos données qui sont issues d'un vocabulaire libre.

Les résultats que nous avons obtenus montrent que notre modèle semble mieux fonctionner avec beaucoup de données. En effet, l'approche GMM est supérieure avec les données originelles tandis que nous avons obtenu de meilleurs résultats qu'eux en augmentant artificiellement l'ensemble d'entraînement, comme il est expliqué à la section 6.7.2.

Compte tenu des expériences que nous avons réalisées, il n'est donc pas possible de déterminer quel algorithme est supérieur puisqu'il faudrait tester le modèle GMM sur un ensemble de données d'une taille beaucoup plus importante. À ce jour, ces expériences n'ont pas encore été réalisées.

## 5.3 Commentaires

La contribution de l'auteur à cet article fut principalement d'effectuer la comparaison avec le modèle *GMM* en utilisant l'ensemble de données *CAL500*, présentée à la section 6.7.

Suite à la publication de l'article, notre modèle fut soumis[17] et évalué au concours MIREX[81] 2008 qui utilisait l'ensemble de données *Majorminer*[66]. Le concours de l'année 2009 a aussi utilisé cet ensemble de données ce qui permet une comparaison des modèles soumis lors de ces deux années. Les résultats des trois meilleurs modèles de chaque année sont présentés au tableau 5.1.

Notre modèle arriva en troisième position en 2008 et sa performance fut en deçà de la plupart des algorithmes soumis en 2009. Tout comme l'ensemble de données *CAL500*, *Majorminor* est beaucoup plus petit que l'ensemble originel sur lequel nous avons effectué nos tests. Nous pensons donc que les mêmes problèmes existent, et qu'une évaluation des algorithmes sur un ensemble de données de taille réelle est nécessaire.

Effectivement, il est aussi intéressant de noter que la plupart des modèles ayant mieux performé qu'*Autotagger* utilisent une machine à vecteurs de support (*SVM*) comme modèle d'ap-

prentissage. L'apprentissage de ce classifieur peut être extrêmement couteux en temps de calcul. Il n'est donc pas nécessairement facile d'utiliser ce type de modèle avec de gros ensembles de données, ce qui peut être un problème majeur dès qu'on tente de les appliquer hors d'un contexte académique.

Deux approches soumises lors de l'édition 2009 sont dignes de mention. Premièrement, [59] utilisent des SVM et AdaBoost ensemble, de manière à former un meta-algorithme et obtenir un classifieur très fort. Deuxièmement, l'idée d'entraîner un deuxième ensemble de classifieurs à partir de la sortie d'un premier ensemble fut repris par [70] en utilisant des SVM au lieu de Filterboost, comme nous avons essayé.

| | Modèle | F-Score (Tag) | AUC-ROC (Tag) |
|---|---|---|---|
| 2009 | Lo, Wang & Wang[59] | 0.311 | 0.807 |
| | Tzanetakis[70] | 0.293 | 0.786 |
| | Burred & Peeters[23] | 0.290 | 0.761 |
| 2008 | GMM[86] | 0.28 | 0.77 |
| | Mandel & Ellis[67] | 0.26 | 0.77 |
| | *Autotagger* | 0.19 | 0.74 |

Tableau 5.1 – Comparaison de la performance d'Autotagger aux autres modèles soumis à la tache *Audio Tag Classification (Majorminor)* du concours MIREX08-09. Les trois meilleurs algorithmes de chaque année sont présentés. Les métriques sont expliquées à la section 6.7.3. Le nombre de modèles soumis fut de 11 en 2008 et de 12 en 2009.

# CHAPITRE 6

# AUTOTAGGER : A MODEL FOR PREDICTING SOCIAL TAGS FROM ACOUSTIC FEATURES ON LARGE MUSIC DATABASES

## 6.1 Abstract

Social tags are user-generated keywords associated with some resource on the Web. In the case of music, social tags have become an important component of "Web 2.0" recommender systems, allowing users to generate playlists based on use-dependent terms such as *chill* or *jogging* that have been applied to particular songs. In this paper, we propose a method for predicting these social tags directly from MP3 files. Using a set of 360 classifiers trained using the online ensemble learning algorithm FilterBoost, we map audio features onto social tags collected from the Web. The resulting automatic tags (or *autotags*) furnish information about music that is otherwise untagged or poorly tagged, allowing for insertion of previously unheard music into a social recommender. This avoids the "cold-start problem" common in such systems. Autotags can also be used to smooth the tag space from which similarities and recommendations are made by providing a set of comparable baseline tags for all tracks in a recommender system. Because the words we learn are the same as those used by people who label their music collections, it is easy to integrate our predictions into existing similarity and prediction methods based on web data.

## 6.2 Introduction

Social tags are a key part of "Web 2.0" technologies and have become an important source of information for recommendation. In the domain of music, Web sites such as Last.fm[53] use social tags to help their users find artists and music (Lamere [48]). In this paper, we propose a method for predicting social tags using audio feature extraction and supervised learning. These automatically-generated tags (or "autotags") can provide information about music for which good, descriptive social tags are lacking. Using traditional information retrieval techniques a music recommender can use these autotags (combined with any available listener-applied tags) to predict artist or song similarity. The tags can also serve to smooth the tag space from which similarities and recommendations are made by providing a set of comparable baseline tags for all artists or songs in a recommender.

This paper presents "Autotagger", a machine learning model for automatically applying text labels to audio. The model is trained using social tags, although it is constructed to work with any training data that fits in a classification framework. This work is an extension of Eck et al. [29, 30]

which proposed an AdaBoost-based model for predicting autotags from audio features. The main contributions of this paper are as follows. First, we extend the model from [29] to sample data from an arbitrarily large pool of audio files. This is achieved by replacing the AdaBoost batch learning algorithm with the FilterBoost online learning algorithm. Second, we explore two ways to take advantage of correlations among the tags collected from Last.fm in order to improve the quality of our automatically-generated tags. Finally we compare our approach to another approach on a new data set. All experimental results in this paper are new and make use of 360 autotags trained on a data set of approximately 100,000 MP3s.

This paper is organized as follows: in 6.3, we describe social tags in more depth, including a description of how social tags can be used to avoid problems found in traditional collaborative filtering systems, as well as a description of the tag set we built for these experiments. In 6.4, we describe previous approaches to automatic tagging of music and related tasks. In 6.5 we present our algorithm for autotagging songs based on labelled data collected from the Internet. In Sections 6.6 through 6.8, we present a series of experiments exploring the ability for the model to predict social tags and artist similarity. Finally, in 6.9, we describe our conclusions and future work.

### 6.3 Using social tags for recommendation

As the amount of online music grows, automatic music recommendation becomes an increasingly important tool for music listeners to find music that they will like. Automatic music recommenders commonly use collaborative filtering (CF) techniques to recommend music based on the listening behaviours of other music listeners. These CF recommenders (CFRs) harness the "wisdom of the crowds" to recommend music. Even though CFRs generate good recommendations there are still some problems with this approach. A significant issue for CFRs is the *cold-start* problem. A recommender needs a significant amount of data before it can generate good recommendations. For new music, or music by an unknown artist with few listeners, a CF recommender cannot generate good recommendations. Another issue is the *lack of transparency* in recommendations (Herlocker et al. [40]). A CF recommender cannot tell a listener why an artist was recommended beyond the superficial description: "people who listen to X also listen to Y."

Music listening occurs in many contexts. A music listener may enjoy a certain style of music when working, a different style of music when exercising and a third style when relaxing. A typical CF recommender does not take the listening context into account when recommending music. Ideally, a music listener should be able to request a music recommendation for new music that takes into account the style of the music and the listening context. Since a CF recommender bases its recommendations on listener behaviour, it cannot satisfy a music recommendation re-

quest such as “recommend new music with female vocals, edgy guitar with an indie vibe that is suitable for jogging.”

An alternative style of recommendation that addresses many of the shortcomings of a CF recommender is to recommend music based upon the similarity of “social tags” that have been applied to the music. Social tags are free text labels that music listeners apply to songs, albums or artists. Typically, users are motivated to tag as a way to organize their own personal music collection. A user may tag a number of songs as *mellow* some songs as *energetic* some songs as *guitar* and some songs as *punk*. Typically, a music listener will use tags to help organize their listening. A listener may play their *mellow* songs while relaxing, and their *energetic* artists while they exercise.

The real strength of a tagging system is seen when the tags of many users are aggregated. When the tags created by thousands of different listeners are combined, a rich and complex view of the song or artist emerges. Table 6.1 shows the top 21 tags and frequencies of tags applied to the band “The Shins”. Users have applied tags associated with the genre (*Indie*, *Pop*, etc.), with the mood (*mellow*, *chill*), opinion (*favorite*, *love*), style (*singer-songwriter*) and context (*Garden State*). From these tags and their frequencies we learn much more about “The Shins” than we would from a traditional single genre assignment such as “Indie Rock”.

Using standard information retrieval techniques, we can compute the similarity of artists or songs based on the co-occurrence of tags. A recommender based upon the social tags addresses some of the issues seen in traditional CFRs. Recommendations are transparent — they can be explained in terms of tags. Recommendations can be sensitive to the listening context. A recommender based on social tags is able to cross the semantic gap, and allow a listener to request a recommendation based upon a textual description of the music. The recommender can satisfy a request to “recommend music with female vocals, edgy guitar with an indie vibe that is suitable for jogging”. However, a social-tag-based recommender still suffers from the cold-start problem that plagues traditional CFRs. A new artist or song will have insufficient social tags to make good recommendations.

In this paper, we investigate the automatic generation of tags with properties similar to those assigned by social taggers. Specifically, we introduce a machine learning algorithm that takes as input acoustic features and predicts social tags mined from the web (in our case, Last.fm). The model can then be used to tag new or otherwise untagged music, thus providing a partial solution to the cold-start problem.

For this research, we extracted tags and tag frequencies from the social music website Last.fm using the Audioscrobbler web services [51] during the spring of 2007. The data consists of over 7 million artist-level tags applied to 280,000 artists. 122,000 of the tags are unique. Table 6.2 shows the distribution of the types of tags for the 500 most frequently applied tags. The majority of tags describe audio content. Genre, mood and instrumentation account for 77% of

the tags. This bodes well for using the tags to predict audio similarity as well as using audio to predict social tags. However, there are numerous issues that can make working with tags difficult. Taggers are inconsistent in applying tags, using synonyms such as *favorite*, *favourite* and *favorites*. Taggers use personal tags that have little use when aggregated (*i own it*, *seen live*). Tags can be ambiguous; *love* can mean a romantic song or it can mean that the tagger loves the song. Taggers can be malicious, purposely mistagging items (presumably there is some thrill in hearing lounge singer Barry Manilow included in a death metal playlist). Taggers can purposely mistag items in an attempt to increase or decrease the popularity of an item. Although these issues make working with tags difficult, they are not impossible to overcome. Some strategies to deal with these are described in Guy and Tonkin's article [39].

A more difficult issue is the uneven coverage and sparseness of tags for unknown songs or artists. Since tags are applied by listeners, it is not surprising that popular artists are tagged much more frequently than non-popular artists. In the data we collected from Last.fm, "The Beatles" are tagged 30 times more often than "The Monkees". This sparseness is particularly problematic for new artists. A new artist has few listeners, and therefore, few tags. A music recommender that uses social tags to make recommendations will have difficulties recommending new music because of the tag sparseness. This cold-start problem is a significant issue that we need to address if we are to use social tags to help recommend new music.

Overcoming the cold-start problem is the primary motivation for this area of research. For new music or sparsely tagged music, we predict social tags directly from the audio and apply these automatically generated tags (called *autotags*) in lieu of traditionally applied social tags. By automatically tagging new music in this fashion, we can reduce or eliminate much of the cold-start problem. More generally, we are able to use these autotags as part of a music recommender to recommend music from the "long tail" of the distribution [43] over popular artists and thus introduce listeners to new or relatively unknown music.

Given that our approach needs social tag data to learn from, it is not a complete solution for the cold-start problem. For a new social recommender having no user data at all, it would be necessary to obtain some initial training data from an external source. Given that many useful sources such as Audioscrobbler are freely available only for non-commercial use, this may be impossible or may require a licensing agreement.

### 6.4 Previous Work and Background

In this section we discuss previous work on music classification and music similarity. In 6.4.1 we carry out an overview of the existing work in genre recognition. Then, in 6.4.2, we discuss issues relating to measuring similarity, focusing on challenges in obtaining ground truth. Finally we provide details about the similarity distance measures used in many of our experiments.

| Tag | Freq | Tag | Freq | Tag | Freq |
|---|---|---|---|---|---|
| Indie | 2375 | The Shins | 190 | Punk | 49 |
| Indie rock | 1138 | Favorites | 138 | Chill | 45 |
| Indie pop | 841 | Emo | 113 | Singer-songwriter | 41 |
| Alternative | 653 | Mellow | 85 | Garden State | 39 |
| Rock | 512 | Folk | 85 | Favorite | 37 |
| Seen Live | 298 | Alternative rock | 83 | Electronic | 36 |
| Pop | 231 | Acoustic | 54 | Love | 35 |

Table 6.1: Top 21 tags applied to *The Shins* for a sample of tags taken from Last.fm.

| Tag Type | Frequency | Examples |
|---|---|---|
| Genre | 68% | heavy metal, punk |
| Locale | 12% | French, Seattle, NYC |
| Mood | 5% | chill, party |
| Opinion | 4% | love, favorite |
| Instrumentation | 4% | piano, female vocal |
| Style | 3% | political, humor |
| Misc | 3% | Coldplay, composers |
| Personal | 1% | seen live, I own it |

Table 6.2: Distribution of tag types for the Last.fm tag sample.

### 6.4.1 Music Classification and Similarity

A wide range of algorithms have been applied to music classification tasks. Lambrou et al. [46], and Logan and Salomon [62] used minimum distance and K-nearest neighbours. Tzanetakis and Cook [89] used Gaussian mixtures. West and Cox [95] classify individual audio frames by Gaussian mixtures, Linear Discriminant Analysis (LDA), and regression trees. Ahrendt and Meng [1] classify 1.2s segments using multiclass logistic regression. In Bergstra et al. [15], logistic regression was used to predict restricted "canonical" genre from the less-constrained noisy genre labels obtained from the FreeDb web service.

Several classifiers have been built around Support Vector Machines (SVMs). Li et al. [58] reported improved performance on the same data set as [89] using both SVM and LDA. Mandel and Ellis [64] used an SVM with a kernel based on the symmetric KL divergence between songs. Their model performed particularly well at MIREX[81] 2005, winning the Artist Recognition contest and performing well in the Genre Recognition contest. While SVMs are known to perform very well on small data sets, the quadratic training time makes it difficult to apply them to large music databases. This motivates research on applying equally well-performing but more time-efficient algorithms to music classification.

The ensemble learning algorithm AdaBoost was used in Bergstra et al. [14] to predict musical genre from audio features. One contribution of this work was the determination of the optimal audio segmentation size for a number of commonly-used audio features and classifiers. This model won the MIREX 2005 genre contest [13] and performed well in the MIREX 2005 artist recognition contest. A similar boosting approach was used in Turnbull et al. [87] to perform musical boundary detection. As mentioned in 6.2, AdaBoost was the algorithm used in Eck et al. [29]. FilterBoost, an online version of AdaBoost which uses rejection sampling to sample an arbitrarily large data set, is used in the current work. See 6.5.2 for details.

Though tasks like genre classification are of academic interest, we argue in our analysis of user tagging behaviour 6.3 that such winner-take all annotation is of limited real-world value. A similar argument is made in McKay and Fujinaga [69]. For a full treatment on issues related to social tags and recommendation see Lamere's article [48].

### 6.4.2 Collecting Ground-Truth Data for Measuring Music Similarity

Measuring music similarity is of fundamental importance for music recommendation. Our approach as introduced in [29] is to use distance between vectors of autotags as a similarity measure. Though our machine learning strategies differ, this approach is similar to that of Barrington et al. [8] which used distance between semantic concept models (similar to our autotags) as a proxy for acoustic similarity. Their approach performed well at MIREX in 2007, finishing third in the similarity task out of 12 with no significant difference among the top four participants. See

6.7 for a comparison of our approach and that of Barrington.

As has long been acknowledged (Ellis et al. [31]), one of the biggest challenges in predicting any attribute about music is obtaining "ground truth" for comparison. For tasks like genre prediction or social tag prediction, obtaining ground truth is challenging but manageable. (For genre prediction an ontology such as provided by AllMusic can be used; for social tag prediction, data mining can be used). The task is more complicated when it comes to predicting the similarity between two songs or artists.

What all researchers want, it is safe to say, is a massive collection of error-free human-generated similarity rankings among all of the songs and artists in the world, in other words a *large and clean* set of ground-truth rankings that could be used both to train and to evaluate models. Though no such huge, pristine similarity data set exists, it is currently possible to obtain either small datasets which are relatively noise-free or large datasets which may contain significant noise.

In general *small and clean* approaches take advantage of a well-defined data collection process wherein explicit similarity rankings are collected from listeners. One option is to use a controlled setting such as a psychology laboratory. For example, Turnbull et al. [85] paid subjects to provide judgements about the genre, emotion and instrumentation for a set of 500 songs. Another increasingly-popular option is to use an online game similar to the now-famous ESP game for images (von Ahn and Dabbish [94]) where points are awarded for describing an image using the same words as another user. Variations of the ESP game for music can be seen in [55, 65, 88].

If one of these games explodes in popularity it has great potential for generating exactly the kinds of large and clean datasets we find useful. In the meantime, large dataset collection techniques are done via data mining of public web resources and thus are not driven by elicited similarity judgements. Our approach uses such a *large and noisy* data collection technique: the word distributions used to train our autotag classifiers come from the Last.fm website, which does nothing to ensure that users consider music similarity when generating tags. Thus it is possible that the word vectors we generate will be noisy in proportion to the noise encountered in our training data. Our belief is that in the context of music similarity a large, noisy dataset will give us a better idea of listener preferences than will a small, clean one. This motivated the construction of our model, which uses classification techniques that scale well to large high-dimensional datasets but which do not in general perform as well on small datasets as do some other more-computationally expensive generative models. We will return this issue of *small and clean* versus *large and noisy* in section 6.7.

### 6.4.3 Measuring Similarity

In our experiments we use three measures to evaluate our ability to predict music similarity. The first, *TopN*, compares two ranked lists: a target "ground truth" list $A$ and our predicted list $B$. This measure is introduced in Berenzweig et al. [11], and is intended to place emphasis on how well our list predicts the top few items of the target list. Let $k_j$ be the position in list $B$ of the $j$th element from list $A$. $\alpha_r = 0.5^{1/3}$, and $\alpha_c = 0.5^{2/3}$, as in [11]. The result is a value between 0 (dissimilar) and 1 (identical top $N$),

$$s_i = \frac{\sum_{j=1}^{N} \alpha_r^j \alpha_c^{k_j}}{\sum_{l=1}^{N} (\alpha_r \times \alpha_c)^l} \tag{6.1}$$

For the results produced below, we look at the top $N = 10$ elements in the lists.

Our second measure is Kendall's *Tau*, a classic measure in collaborative filtering which measures the number of discordant pairs in 2 lists. Let $R_A(i)$ be the rank of the element $i$ in list $A$, if $i$ is not explicitly present, $R_A(i) = length(A) + 1$. Let $C$ be the number of concordant pairs of elements $(i, j)$, e.g. $R_A(i) > R_A(j)$ and $R_B(i) > R_B(j)$. In a similar way, $D$ is the number of discordant pairs. We use Kendall's *Tau*'s definition from Herlocker et al. [41]. We also define $T_A$ and $T_B$ the number of ties in list $A$ and $B$. In our case, it's the number of pairs of artists that are in $A$ but not in $B$, because they end up having the same position $R_B = length(B) + 1$, and reciprocally. Kendall's *Tau* value is defined as:

$$\tau = \frac{C - D}{sqrt((C + D + T_A)(C + D + T_B))} \tag{6.2}$$

Unless otherwise noted, we analyzed the top 50 predicted values for the target and predicted lists.

Finally, we compute what we call the *TopBucket*, which is simply the percentage of common elements in the top $N$ of 2 ranked lists. Here we compare the top 20 predicted values unless otherwise noted.

## 6.5 Autotagger: an Automatic Tagging Algorithm using FilterBoost

We now describe a machine learning model which uses the *meta-learning* algorithm FilterBoost to predict tags from acoustic features. This model is an extension of a previous model [29], the primary difference being the use of FilterBoost in place of AdaBoost. FilterBoost is best suited for very large amounts of data. See figure 6.1 for an overview.

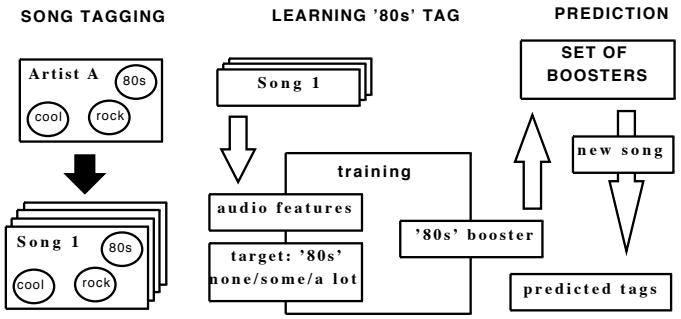


Figure 6.1: Overview of our model. A booster is learnt for every tag, and can then be used to "autotag" new songs.

### 6.5.1 Acoustic Feature Extraction

The features we use include 20 Mel-Frequency Cepstral Coefficients, 176 autocorrelation coefficients of an onset trace sampled at $100Hz$ and computed for lags spanning from 250msec to 2000msec at 10ms intervals, and 85 spectrogram coefficients sampled by constant-Q (or log-scaled) frequency. See previous work [14] or Gold and Morgan [36] for more detailed descriptions of these standard acoustic features.

The audio features described above and shown in figure 6.2 are calculated over short windows of audio ($\sim$100ms with 25ms overlap). This yields too many features per song for our purposes. To address this, we create "aggregate" features by computing individual means and standard deviations (i.e., independent Gaussians) of these features over 5s windows of feature data. When fixing hyperparameters for these experiments, we also tried a combination of 5s and 10s features, but saw no real improvement in results. For reasons of computational efficiency we used random sampling to retain a maximum of 12 aggregate features per song, corresponding to 1 minute of audio data.

### 6.5.2 AdaBoost and FilterBoost

AdaBoost [35] is an *ensemble* (or *meta-learning*) method that constructs a classifier in an iterative fashion. It was originally designed for binary classification, and it was later extended to multi-class classification using several different strategies.

In each iteration $t$, the algorithm calls a simple learning algorithm (the *weak learner*) that returns a classifier $h^{(t)}$ and computes its coefficient $\alpha^{(t)}$. The input of the weak classifier is a $d$-dimensional observation vector $x$ containing the features described in 6.5.1, and the output of $h^{(t)}$ is a binary vector $x \in \{-1,1\}^k$ over the $k$ classes. If $h_\ell^{(t)} = 1$ the weak learner "votes for" class $\ell$ whereas $h_\ell^{(t)} = -1$ means that it "votes against" class $\ell$. After $T$ iterations, the algorithm

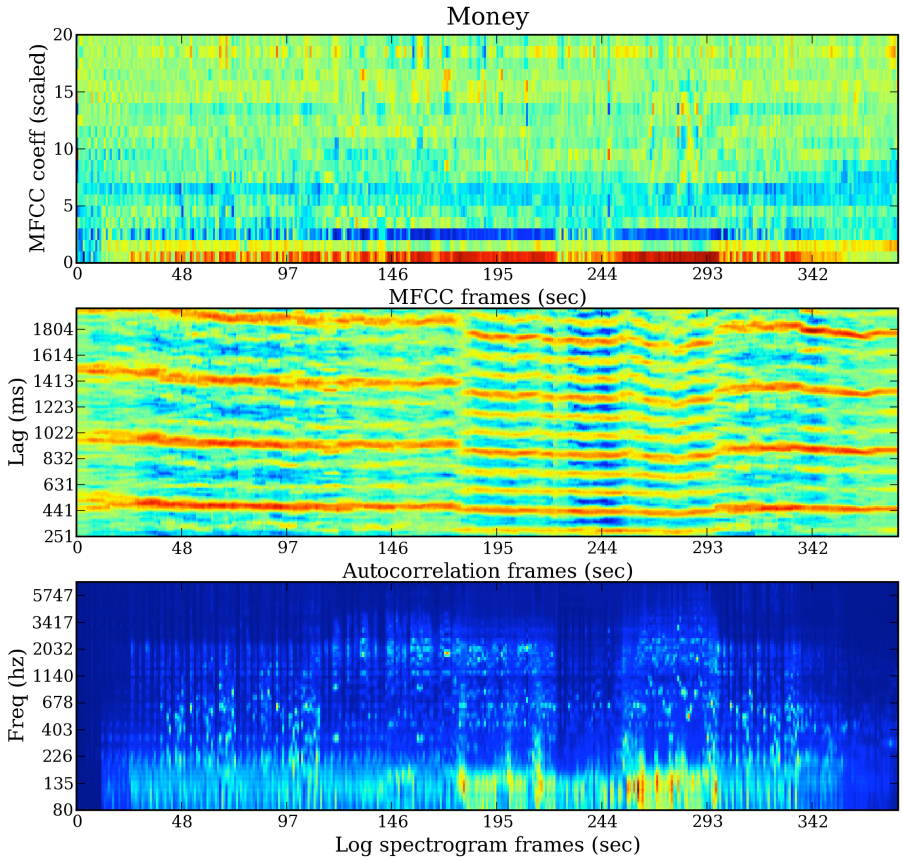


Figure 6.2: Acoustic features for "Money" by Pink Floyd.

outputs a vector-valued discriminant function

$$g(x) = \sum_{t=1}^{T} \alpha^{(t)} h^{(t)}(x) \tag{6.3}$$

Assuming that the feature vector values are ordered beforehand, the cost of the weak learning is $O(nkd)$ ($n$ number of examples), so the whole algorithm runs in $O\big(nd(kT + \log n)\big)$ time. Thus, though boosting may need relatively more weak learners to achieve the same performance on a large data set than a small one, the computation time for a single weak learner scales linearly with the number of training examples. Thus AdaBoost has the potential to scale well to very large data sets.

FilterBoost [22] is an extension to AdaBoost which provides a mechanism for doing rejection sampling, thus allowing the model to work efficiently with large data sets by choosing training examples based on their similarity. The addition of rejection sampling makes it possible to use

FilterBoost with data sets which are too large or too redundant to be used efficiently in a batch learning context. This is the case for industrial music databases containing a million or more tracks and thus tens or hundreds of millions of audio segments.

Data is presumed to be drawn from an infinitely large source called an "oracle". The filter receives a sample $(x,l)$ from the oracle at iteration $t+1$ and accepts it with a probability:

$$q_t(x,l) = \frac{1}{1+e^{l g_t(x)}} \tag{6.4}$$

$l$ being the true class of $x$, $l \in \{-1,+1\}$, and $g(x)$ the output of the booster between $-1$ and 1. Sampling continues until a small data set (usually 3000 examples in our experiments) is constructed, at which time we select the best weak learner $h^{(t+1)}(x)$ on this set, and then evaluate $h^{(t+1)}(x)$ weight by sampling again from the oracle (see [22] for details).

It is also possible to select more than one weak learner in each round, using the classical AdaBoost weighting method on the small data set created. Conceptually, all the weak learners selected in a single pass can be considered as one single learner by grouping them. In our experiments, we choose 50 weak learners per round.

Regardless of whether AdaBoost or FilterBoost is employed, when no "a priori" knowledge is available for the problem domain, small decision trees or, in the extreme case, *decision stumps* (decision trees with two leaves) are often used as weak learners. In all experiments reported here, decision stumps were used. In earlier experiments we also tried decision trees without any significant improvement. Because decision stumps depend on only one feature, when the number of iterations $T$ is much less than the number of features $d$, then the booster acts as an implicit feature extractor that selects the $T$ most relevant features to the classification problem. Even if $T > d$, one can use the coefficients $\alpha^{(t)}$ to order the features by their relevance. Because a feature is selected based on its ability to minimize empirical error, we can use the model to eliminate useless feature sets by looking at the order in which those features are selected. We used this property of the model to discard many candidate features such as chromagrams (which map spectral energy onto the 12 notes of the Western musical scale) because the weak learners associated with those features were selected very late by the booster.

### 6.5.3 Generating Autotags using Booster Outputs

Each booster is trained using individual audio segments as input (Figure 6.1). However we wish to make predictions on the level of tracks and artists. In order to do so we need to integrate segment-level predictions into track and artist level predictions. One way to do this is to take the mean value of the hard discriminant function $\mathbf{sign}(g(x))$ for all segments. Instead we take the mean or median of the raw discriminant function (i.e., the sum of the weak learner predictions) $g(x)$ for all segments. So that the magnitudes of the weak learner predictions are

more comparable we normalize the sum of the weak learner weights $\alpha$ to be 1.0. This yields a song-level prediction scaled between 0 and 1 where .5 is interpreted as incertitude.

This normalization is useful in that it allows us to use and compare all words in our vocabulary. Since FilterBoost is a large-margin classifier, it will try to push audio segments as far away from the boundary as possible. The effect of this will be that tags that are easy-to-learn will have larger $\alpha$ values than hard-to-learn tags. Lacking normalization, difficult-to-learn words tend not to have any impact at all because the booster confidences are so low in comparison to other tags. The undesirable side-effect of our approach is that the impact of very poorly learned tags is no longer attenuated by low learner confidence. In a production system it would be important to filter out such tags so they do not contribute undue noise. Though we used no such filtering for these experiments, it can be easily implemented by discarding tags which fall below a threshold out-of-sample classification rate.

### 6.5.4 Second-stage learning and correlation reweighting

As discussed above, each social tag is learned independently. This simplifies our training process in that it allows us to continually update the boosters for a large set of tags, thus spreading out the computation load over time. Furthermore it allows us to easily add and subtract individual tags from our set of relevant tags as the social tagging data changes over time. If the tag-level models were dependant on one another this would be difficult or impossible. It is clear, however, that an assumption of statistical independence among tags is false. For example, a track labelled "alternative rock" is also likely to be labelled "indie rock" and "rock". By ignoring these interdependencies, we make the task of learning individual tags more difficult. We test two techniques for addressing this issue.

Our first approach uses a second set of boosted classifiers. These "second-stage" learners are trained using the autotag predictions from the first stage. In other words, each second-stage booster predicts a single social tag using a weighted mixture of acoustically-driven autotags. Since the input includes the results from the first stage of learning, convergence is fast.

In our second approach we calculate the empirical correlation among the social tagging data obtained from Last.fm. We then adjust our predictions for a tag (whether from the single-stage or two-stage approach) by mixing predictions from other tags proportional to correlation. For example, suppose the tags *rock* and *electric guitar* are highly correlated in the Last.fm data, the prediction $p$ for the tag *rock* will become $p'(\text{rock}) = p(\text{rock}) + \lambda\, p(\text{electric guitar})$, where $\lambda$ is proportional to their correlation. The correlation matrix is computed once for the entire Last.fm data set and applied uniformly to all autotags for all songs.

### 6.5.5 Generating Labelled Datasets for Classification from Audioscrobbler

We extracted tags and tag frequencies for nearly 100,000 artists from the social music website Last.fm using the Audioscrobbler web service [51]. From these tags, we selected the 360 most popular tags. Those tags are listed in the appendix I. Because it was impossible to obtain a sufficient number of song-level tags, only artist tags were used. This is admittedly a questionable practice, especially for artists whose work spans many different styles. For example, the Beatles are currently the number four artist for the tag "psychedelic" yet only a few Beatles songs fit that description. Currently there are many more tags applied to artists than to tracks. As more track-level tags become available we will take advantage of them.

Intuitively, automatic labelling would be a regression task where a learner would try to predict tag frequencies for artists or songs. However, because tags are sparse (many artist are not tagged at all; others like Radiohead are heavily tagged) this proves to be too difficult using our current Last.fm data set. Instead, we chose to treat the task as a classification one. Specifically, for each tag we try to predict if a particular artist would or would not be labelled with a particular tag. The measure we use for deciding how well a tag applies to an artist is:

$$\text{weight} = \frac{\text{\# times this tag was applied to this artist}}{\text{\# times any tags was applied to this artist}} \tag{6.5}$$

In our previous work [30], class labels were generated by dividing tags into three equal-sized sets (e.g. *no rock*, *some rock* or *a lot of rock*). With this strategy, hundreds or even thousands of artists appeared as positive examples. In our current work, we chose to select *positive examples* from only the top 10 artist for any given tag. The remaining artists which received enough tags to make Audioscrobbler's top 1000 list for that tag are treated as *ignore examples* which are not used for learning but which are used to test model performance. The set of *negative examples* are drawn by randomly sampling from all artists in our music collection which did not make the top 1000 list for a tag. With this strategy in many instances such as "rock" a booster is trained on only a tiny proportion of the valid positive artists suggested by Last.fm, resulting in a "rock" autotag that will certainly fail to find some salient characteristics of the genre, having never seen a large number of positive examples. This is in keeping with our goal to generate a large set of autotags which each succeeds at modeling a relatively narrow, well-defined subspace.

Regardless of the specific strategy we use for generating datasets, the set of positive examples for a tag will always be much smaller than the set of negative examples. This extreme imbalance suggests that we should preserve as many positive examples as possible, thus motivating our decision to use all top 10 artist songs for training. In addition, when training we sample equally between positive and negative training examples, thus artificially balancing the sets.

All of the music used in these experiments is labelled using the free MusicBrainz[1] service. The MusicBrainz track, album and artist ids used in our experiments are available on request.

## 6.6 Predicting Social Tags

In our first experiment, we measure booster predictions on the positive list, the ignore list and randomly-selected examples from the negative list. Recall that the positive list for a tag $T$ is made of the songs for the 10 artists whose weight (equation 6.5) for that tag is the highest. The ignore list is made of all the songs for artists with high weight for that tag, but not enough to be in the top 10. Negative examples are drawn from the rest of the database. For a tag like *rock*, we have 10 positive artists, about 900 ignored artists and 3000 negative ones. Results are presented in tables 6.3 and 6.4. Table 6.3 displays success rate as a percentage of correctly classified songs. In parentheses we also show classification rates for second-stage boosting as discussed in section 6.5.4. Song-level predictions were obtained by taking the median for all segment-level predictions for that song. The results for the positive list can be seen as measuring training error, but the two other measures give an idea of how well we generalize: we did not train on any tracks from the ignore list, and we randomly sampled songs from 120K negative ones during training, so there is little risk of overfitting on 200 random songs. Table 6.4 provides examples of normalized booster outputs. In both tables, we also provide the results for selected genres, instrument and mood-related terms. Again, parentheses show results for second-stage learning. From a computational point of view, we train 2000 weak learners per word, as we did in earlier work [30]. However, FilterBoost computes them in a couple of hours instead of 1 or 2 days previously.

### 6.6.1 Second-stage learning

Second-stage learning results were obtained by training FilterBoost for 500 iterations. As there were only 360 autotag values present in the input vector, the booster could capture the influence of every autotag if necessary. The classification results in parentheses 6.3 show improved performance for positive and negative examples but degraded performance for the ignore list. The mean normalized booster outputs in 6.4 suggest that the second-stage boosting is separating more strongly the positive and negative classes. For more results on our second-stage learning, see 6.7 and 6.8.

[1] `www.musicbrainz.org`

| Boosters | Positive (10 artists) | Ignore (100 songs) | Negative (200 songs) |
|---|---|---|---|
| main genres (rock, pop, Hip-Hop, metal, jazz, dance, Classical, country, blues, reggae) | 89.1% (90.1%) | 80.6% (78.9%) | 80.0% (79.1%) |
| instruments (piano, guitar, saxophone, trumpet) | 87.0% (88.7%) | 61.0% (60.3%) | 82.4% (83.6%) |
| mood (happy, sad, romantic, Mellow) | 87.8% (89.1%) | 67.4% (66.8%) | 79.0% (81.1%) |
| **all** | **88.2**% (90.5%) | **60.0**% (57.2%) | **81.4**% (84.1%) |

Table 6.3: Song classification results, percentage of the songs that are considered positive (for positive and ignore examples) or negative for negative examples. Values in parentheses are for second-stage boosters.

| Boosters | Positive (10 artists) | Ignore (100 songs) | Negative (200 songs) |
|---|---|---|---|
| main genres (rock, pop, Hip-Hop, metal, jazz, dance, Classical, country, blues, reggae) | 0.540 (0.572) | 0.528 (0.544) | 0.463 (0.438) |
| instruments (piano, guitar, saxophone, trumpet) | 0.538 (0.570) | 0.507 (0.520) | 0.470 (0.438) |
| mood (happy, sad, romantic, Mellow) | 0.532 (0.558) | 0.511 (0.517) | 0.463 (0.433) |
| **all** | **0.536** (0.569) | **0.508** (0.509) | **0.466** (0.432) |

Table 6.4: Mean normalized booster output per song for selected tag types. Values in parentheses are for second-stage boosters.

### 6.6.2 Correlation reweighting

To investigate the effects of reweighting autotag predictions as a function of empirical correlation, we look at the ordering of the artists taken from the ignore list from our training set. Recall that ignore-list artists for a tag $T$ are the ones that were labelled significantly with tag $T$, but not enough to appear among the top 10 artists (i.e the positive list for tag $T$). We assume that having a good ordering of these artists, e.g. "from the most *rock* to the least *rock*", is a good measure of performance.

We generate a ground-truth list by sorting the Last.fm tags by their weight, as defined by equation 6.5. We then compare this ground truth to three lists: a random list, a list ordered by our autotags and a list ordered by correlation-reweighted autotags. Lists are sorted by the median normalized booster output per song over all songs for an artist. Results are shown in 6.5.

| Autotags | TopN 10 | Kendall 50 | TopBucket 20 |
|---|---|---|---|
| random artist order | −0.663 | 0.003 | 0.007% |
| without correlation | −0.577 | 0.024 | 5.64% |
| with correlation | −0.569 | 0.027 | 6.33% |

Table 6.5: Ordering of artists (in the ignore list) per tag. Ordering is based on median normalized booster output.

The correlation reweighting yields improved performance on all three measures we tested. However the improvement is relatively minor (e.g. less than 2% for TopBucket 20). Many factors can explain why the improvement is not greater. First, our method for generating a ground-truth list yields only a general idea of good ordering. Second, Audioscrobbler [51] only gives us access to a limited number of tag counts. Having more data would increase the precision of our correlation factors. Third, the tagging is very sparse: most artists are tagged reliably with only a few of the 360 tags we investigated. This can lead to spurious correlation measures for otherwise unrelated tags. Finally, we trained on only 10 *positive* artists. For a general tag like *rock*, it is obvious that 10 artists cannot represent the whole genre. If all positive artists for *rock* are rockers from the 60s, there is not much chance that Radiohead (heavily tagged *rock*) will be correctly placed among others artists in the ignore list, being too different from the positive artists.

## 6.7 Comparison with GMM approach

We now compare our model to one developed by the Computer Audition Lab (CAL) group that uses a hierarchical Gaussian mixture model (GMM) to predict tags [85, 86]. We make use of the same dataset (CAL500) used in their experiments. One goal of this comparison is to investigate the relative merits of the *small and clean* versus *large and noisy* approaches discussed in 6.4.2. The experiments follow closely [86], and GMM results also come from this paper.

### 6.7.1 The CAL500 data set

The Computer Audition Lab 500-Song (CAL500) data set (Turnbull et al. [85, 86]) was created by the CAL group by having 66 paid students listen to and annotate a set of songs. The collection is made of 500 recent Western songs by 500 different artists in a wide range of genres. The tags applied to the corpus can be grouped into six categories: instrumentation, vocal characteristics, genre, emotions, acoustic qualities and usage terms. The data was collected by presenting the students with an HTML form comprised of a fixed set of 135 tags. This is quite different from the Last.fm tags used in our previous experiments because the respondents were constrained to a predetermined set of words, resulting in cleaner tags. Some of the tags received

a rating on a scale of 1 to 5 (ex.: emotion-related tags), others on a 1 to 3 scale (ex.: presence of a instrument could be marked "yes", "no" or "uncertain") and some other received binary ratings (ex.: genre-related tags). There were a total of 1708 song evaluations collected with a minimum of three per song. All bipolar tags were then broken down into two different tags (thus generating more than the original 135 tags). For example, "The song is catchy" was broken down to "catchy" and "not catchy", with ratings of 1 and 2 counting towards the "not catchy" tag, ratings of 4 and 5 counting towards the "catchy" and the ratings of 3 being simply ignored.

The ground truth was created by assigning a tag to a song if a minimum of two people assigned that tag to the song and if there was an agreement between the different survey respondents. The respondents were considered in agreement if

$$\left[\frac{\#(\text{People who assigned tag to song}) - \#(\text{People who didn't})}{\#(\text{People who evaluated song})}\right] > 0.8. \tag{6.6}$$

As a final step, all the tags that were assigned to less than five songs were pruned, which resulted in a collection of 174 tags.

### 6.7.2 Evaluation

We trained and evaluated our model in the same way as did the CAL group, using 10 fold cross-validation on the 500 songs (i.e., 450-song training set, 50-song test set). We trained one booster per tag using different audio feature sets. First, we trained on the MFCC deltas provided with the CAL500 data set, which were the features used by the CAL group [86]. We then trained on our aggregated audio features (afeats) and on our autotags, creating second-stage autotags (bfeats).

| Category | Avg. # pos. ex. | | Avg. # pos. after expansion | | Avg. # neg. examples | |
|---|---|---|---|---|---|---|
| All words | 69.16 | (74.50) | 85.60 | (63.42) | 382.84 | (74.50) |
| Emotion | 128.46 | (59.28) | 129.21 | (58.09) | 323.54 | (59.28) |
| Genre | 25.06 | (25.91) | 52.87 | (13.96) | 435.13 | (11.78) |
| Instrumentation | 70.35 | (79.42) | 84.80 | (70.71) | 381.65 | (79.42) |
| Solo | 12.70 | (9.79) | 48.59 | (1.48) | 439.30 | (9.79) |
| Usage | 27.38 | (27.31) | 52.91 | (16.59) | 424.62 | (27.31) |
| Vocal | 34.44 | (28.63) | 55.13 | (17.19) | 417.56 | (28.63) |

Table 6.6: Average per-fold number of positive, negative and positive examples after expansion in the CAL500 data set. The numbers in parentheses are the standard deviation. The expansion of the "Solo" tags averages to a number inferior to 50 because we did not have enough songs by the artists in the original positives examples in our own music collection. For example, if there is only one positive example and we do not have any additional song by that artist, we will not be able to do any expansion on that fold.

Since the CAL500 data set is relatively small, some tags have very few positive examples (i.e., 5 positives for 445 negatives) in certain folds. To explore the influence of the number of examples, we tried *expanding* the training set by adding random songs from the artists that were already part of the positive examples, so that every fold had at least 50 positive examples. The new songs were taken from our internal research collection. The training on the expanded training set was done using the afeats (afeats exp.). The average number of positive, negative and positive examples after expansion are listed in Table 6.6. The test set was left unchanged.

### 6.7.3 Results

We discuss the results for annotation and for retrieval separately in the following two sections.

#### 6.7.3.1 Results for Annotation

This section treats the task of annotating a given track with an appropriate set of tags. The annotation evaluation and comparison of the model was done using the two evaluation measures used in Turnbull et al. [86], mean per-tag precision and recall, as well as a third one, the F-score. All three are standard information retrieval metrics. We start by annotating each song in our test set with an arbitrary number of tags that we refer to as the annotation length. Since the CAL group used ten tags, we used that number as well for comparison purposes. Per-tag precision can be defined as the probability that the model annotates relevant songs for a given tag. Per-tag recall can be defined as the probability that the model annotates a song with a tag that should have been annotated with that tag. More formally, for each tag $t$, $|t_{GT}|$ is the number of songs that have been annotated with the tag in the human-generated "ground truth" annotation. $|t_A|$ is the number of songs that our model automatically annotates with the tag $t$. $|t_{TP}|$ is the number of "correct" (true positive) tags that have been used both in the ground truth and in the automatic tag prediction. Per-tag recall is $|t_{TP}|/|t_{GT}|$ and per-tag precision is $|t_{TP}|/|t_A|$. The F-score takes into account both recall and precision and is defined as $2 \cdot (\text{precision} \cdot \text{recall})/(\text{precision} + \text{recall})$. No variance is provided for our F-score measure because it was computed from the averaged precision and recall for all words; per-tag precision and recall values were not available for the GMM. All three of these metrics range between 0 and 1 but are upper bounded by a value of less than one in our results because we forced the model to output less tags than the number that are actually present in the ground truth. The upper bound is listed as "UpperBnd" in the results tables.

The results for the precision and recall scores are given in Table 6.7. In general the models were comparable, with Autotagger performing slightly better on precision for all feature sets while the GMM model performed slightly better for recall. Precision and recall results for different categories are found in the appendix I. Though a good balance of precision and recall is

always desirable, it has been argued that precision is more important for recommender systems. See, for example, Herlocker et al. [40]. Also, training with the second-stage autotags (bfeats) as input to the boosters produced better precision and recall results than the afeats. This suggests that the extra level of abstraction given by the bfeats can help the learning process.

| Category | $A/\|V\|$ | Model | Precision | | Recall | | F-Score |
|---|---|---|---|---|---|---|---|
| All words | 10/174 | Random | 0.144 | (0.004) | 0.064 | (0.002) | 0.089 |
| | | UpperBnd | 0.712 | (0.007) | 0.375 | (0.006) | 0.491 |
| | | GMM | 0.265 | (0.007) | **0.158** | (0.006) | 0.198 |
| | | MFCC delta | 0.281 | (0.066) | 0.131 | (0.019) | 0.179 |
| | | afeats | 0.266 | (0.078) | 0.094 | (0.018) | 0.139 |
| | | afeats exp. | **0.312** | (0.060) | 0.153 | (0.015) | **0.205** |
| | | bfeats | 0.291 | (0.105) | 0.089 | (0.034) | 0.136 |

Table 6.7: Music annotation results. $A$ = Annotation length, $|V|$ = Vocabulary size. Numbers in parentheses are the variance over all tags in the category. GMM is the results of the Gaussian mixture model of the CAL group. MFCC delta, afeats, afeats extended and bfeat are the results of our boosting model with each of these features. Random, UpperBnd and GMM results taken from [86]. Continued in II.1.

The annotation length of 10 tags used to compute the precision and recall results could be restrictive depending on the context in which the tagging is used. If the goal is to present a user with tags describing a song or to generate tags that will be used in a natural language template as the CAL group did, 10 tags seems a very reasonable choice. However, if the goal is to do music recommendation or compute similarity between songs, a much higher annotation length may give rise to better similarity measures via word vector distance. As shown in Figure 6.3, our recall score goes up very quickly while our precision remains relatively stable when increasing the annotation length. This supports the idea that we can scale to larger annotation lengths and still provide acceptable results for music similarity and recommendation.

To provide an overall view of how Autotagger performed on the 35 tags we plot precision in 6.4 for the different feature sets. This figure, and particularly the failure of the model to predict categories such as "World (Best)" and "Bebop" for certain feature sets, is discussed later in 6.7.4.

#### 6.7.3.2 Results for Retrieval

In this section we evaluate our ability to retrieve relevant tracks for a given tag query. The evaluation measures used are the same as in Turnbull et al. [86]. To measure our retrieval performance, we used the same two metrics as the CAL group. They are the mean average precision and the area under the receiver operating characteristic curve (AROC). The metrics

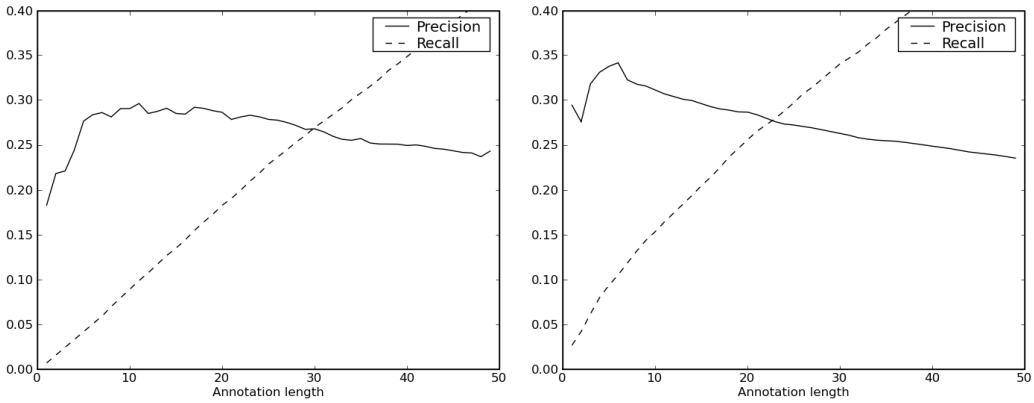


Figure 6.3: Precision and recall results for different annotation lengths when training on (a) the bfeats and (b) the expanded afeats.

| Category | $\|V\|$ | Model | MeanAP | | MeanAROC | |
|---|---|---|---|---|---|---|
| All words | 174 | Random | 0.231 | (0.004) | 0.503 | (0.004) |
| | | GMM | **0.390** | (0.004) | **0.710** | (0.004) |
| | | MFCC delta | 0.305 | (0.057) | 0.678 | (0.015) |
| | | afeats | 0.323 | (0.092) | 0.622 | (0.013) |
| | | afeats exp. | 0.385 | (0.06) | 0.674 | (0.010) |
| | | bfeats | 0.340 | (0.124) | 0.662 | (0.020) |

Table 6.8: Retrieval results. $|V|$ = Vocabulary size. Numbers in parentheses are the variance over all tags in the category. GMM is the results of the Gaussian mixture model of the CAL group. MFCC delta, afeats, afeats extended and bfeat are the results of our boosting model with each of these features. Random and GMM taken from [86]. Continued in II.2.

were computed on a rank-ordered test set of songs for every one-tag query $t_q$ in our vocabulary $V$. Average precision puts emphasis on tagging the most relevant songs for a given tag early on. We can compute the average precision by going down the rank-ordered list and averaging the precisions for every song that has been correctly tagged in regard to the ground truth (true positive). The ROC curve is a plot of the fraction of true positives as a function of the fraction of false positives that we get when moving down our ranked list. The AROC is the area under the ROC curve and is found by integrating it. The mean of both these metrics can be found by averaging the results over all the tags in our vocabulary.

Table 6.8 shows the retrieval results for these measures. Here it can be seen that the GMM model slightly outperforms the Autotagging model but that results for "All words" are comparable. We also see that again the two-stage learner (bfeats) outperforms the single-stage learner.

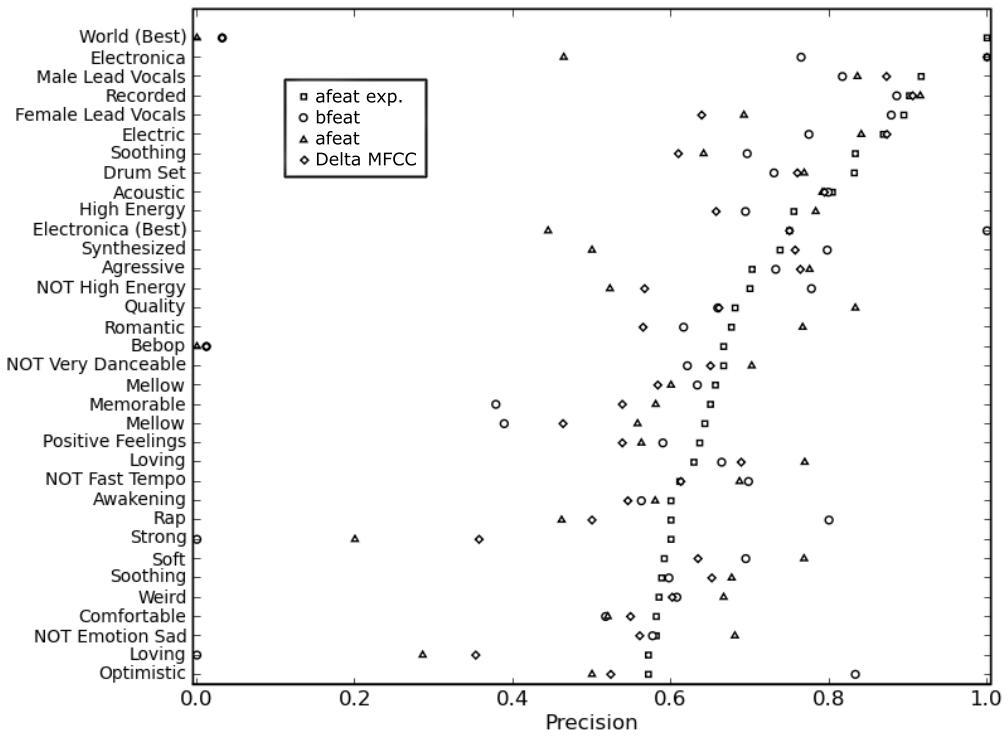


Figure 6.4: "Autotagger's" precision scores using different feature sets on 35 CAL500 tags ordered by performance when using the expanded afeats. Plot inspired by M. Mandel's one [63].

### 6.7.4 Discussion

We now compare the tags generated by Autotagger (using two-stage learning on expanded afeats) to several other lists on the Red Hot Chili Peppers song "Give It Away". Annotations from the GMM model are taken from [86]. The results are presented in 6.9. Here we observe that the Autotagger list includes words which would be difficult or impossible to learn such as "good music" and "seen live." This suggests that filtering out tags with low classification rates would improve performance for annotation.

An important observation is that our model achieves its best performance with the expanded afeats. This provides (modest) evidence in support of large and noisy datasets over small and clean ones. Despite the fact that these audio examples were not labelled by the trained listeners, by adding them to the training set, we improve our performance on the unmodified test set and end up doing better than the GMM model on almost every evaluation metric. We even improve our precision results for tags like "Acoustic Guitar Solo" by considering all the songs added to the training set as having an acoustic guitar solo, which is an assumption that can potentially be wrong most of the time.

| GMM | CAL500 words | autotags | Last.fm Tags |
|---|---|---|---|
| dance pop | not mellow | good music | Rock |
| aggressive | not loving | pop rock | 90s |
| not calming | exercising | Funk Rock | Alternative Punk |
| angry | rapping | crossover | Funk |
| exercising | monotone | rock | Alternative |
| rapping | tambourine | USA | Alternative Rock |
| heavy beat | at work | Favorite Artists | Funk Rock |
| pop | gravelly | american | Hard Rock |
| not tender | hard rock | seen live | Punk |
| male vocal | angry | classic rock | Funk Metal |

Table 6.9: Top 10 words generated for the Red Hot Chili Peppers song *Give it Away* first by the hierarchical Gaussian mixture (GMM) from CAL group, then by our model trained with the extended afeats with CAL500 tags, by our model (using second-stage boosters) with Last.fm tags and finally the top tags on Last.fm. Ordering for GMM is approximated.

One challenge in working with the CAL500 data is that only 3.4 listeners on average rated each song. For example, the song "Little L" by Jamiroquoi was tagged by three different persons who disagreed strongly on some tags like "Drum machine", which was annotated as "None" by one student and as "Prominent" by another. This may introduce problems when using these annotations as ground truth. This issue is addressed by requiring an agreement of 80% among respondents in order to apply a tag to a song. However with only an average of 3.4 survey respondents per song, most of the time all respondents need to tag a song as positive for the tag to be applied in the ground truth. Since the survey participants are not professional music reviewers, it is reasonable to assume that there will be significant disagreement. This issue is easily addressed by obtaining many more annotations per song.

One repercussion of this problem is illustrated in 6.4, which shows our model's precision results on 35 tags using the different feature sets. Since the tags are ordered by their precision score when using the expanded afeats, some of them stand out by having a drastic performance difference when using the expanded afeats or another feature set. In most cases, these outliers stem from having very few positive examples in the training set for these tags. For example, the tags *World (Best)* and *Bebop* respectively have an average of 1.6 and 0.6 per-fold positive examples in the training set. Following the training set expansion and using the afeats in both cases, their precision went from 0.03 to 1.0 for *World (Best)* and from 0.01 to 0.67 for *Bebop*.

Overall we can conclude by looking at 6.7 and 6.8 that the performance of the Autotagger model and the GMM model are comparable, with the GMM performing slightly better at recall while the Autotagger model performs slightly better on precision. However we stress that the best Autotagger results are obtained *when the expanded feature set is used* and that without more

data, the GMM approach performs better. In terms of comparing the algorithms themselves, this is not a fair comparison because it is likely that the GMM approach would also perform better when trained on the expanded feature set. In this sense neither algorithm can be said to be strictly better. A comparison of these two approaches using a realistically-large dataset for music recommendation ($>$ 500K songs and thus millions of audio frames) is called for.

## 6.8 Application to Similarity

As mentioned in section 6.4.3, one key area of interest lies in using our autotags as a proxy for measuring perceived music similarity. By replacing social tag-based similarity with autotag-based similarity we can then address the cold start problem seen in large-scale music recommenders[2]. In the following experiments, we measure our model's capacity to generate accurate artist similarities.

### 6.8.1 Ground Truth

As has long been acknowledged (Ellis et al. [31]), one of the biggest challenges in addressing this task is to find reasonable "ground truth" against which to compare our results. We seek a similarity matrix among artists which is not overly biased by current popularity, and which is not built directly from the social tags we are using for learning targets. Furthermore we want to derive our measure using data that is freely available on the web, thus ruling out commercial services such as the AllMusic Guide [3]. Our solution is to construct our ground truth similarity matrix using correlations from the listening habits of Last.fm users. If a significant number of users listen to artists *A* and *B* (regardless of the tags they may assign to that artist) we consider those two artists similar. Note that, although these data come from the same web source as our artist-level training data, they are different: we train our system using tags applied to artists, regardless of which user applied the tag.

One challenge, of course, is that some users listen to more music than others and that some artists are more popular than others. We use term frequency-inverse document frequency (TF$\times$IDF) weighting scheme to overcome this issue. The complete description of how we build this ground truth can be found in Eck et al. [30].

We also use a second ground truth which has no connection to Last.fm. The All Music Guide (AMG) is a website containing a lot of information about music made by human experts, in particular lists of similar artists. Based on an idea from Ellis et al. [31] we calculate similarity

[2] Of course, real recommenders deal with a more complex situation, caring about novelty of recommendations, serendipity and user confidence among others (see Herlocker et al. [41] for more details). However, similarity is essential. We do it on the artist level because the data available to build a ground truth would be too sparse on the album or song level.

[3] `www.allmusic.com`

using Erdös distances. If an artist $A1$ is similar to another artist $A2$, they have a distance of 1. If artist $A3$ is similar to $A2$, $A1$ and $A3$ have a distance of 2, and so on. Put another way, it is the number of steps to go from one artist to another in a connected graph. We mined 4672 artists on AMG for these experiments.

### 6.8.2 Experiments

We construct similarity matrices from our autotag results and from the Last.fm social tags used for training and testing. The similarity measure we used was *cosine similarity*, defined as:

$$s_{cos}(\mathbf{a}_1, \mathbf{a}_2) = \mathbf{a}_1 * \mathbf{a}_2 / (||\mathbf{a}_1|| \; ||\mathbf{a}_2||), \tag{6.7}$$

where $\mathbf{a}_1$ and $\mathbf{a}_2$ are two vectors of tag magnitudes for an artist. In keeping with our interest in developing a commercial system, we used all available data for generating the similarity matrices, including data used for training. (The chance of overfitting aside, it would be unwise to remove The Beatles from your recommender simply because you trained on some of their songs). The similarity matrix is then used to generate a ranked list of similar artists for each artist in the matrix. These lists are used to compute the measures describe in 6.4.3. Results are found in 6.10.

| Groundtruth | Model | TopN 10 | Kendall 50 | TopBucket 20 |
|---|---|---|---|---|
| Last.fm | social tags | 0.437 | −0.057 | 42.98% |
| | 2nd-stage autotags | 0.149 | −0.361 | 19.90% |
| | autotags | 0.140 | −0.381 | 18.50% |
| | random | 0.006 | −0.626 | 2.00% |
| AMG | social tags | 0.234 | −0.287 | 26.80% |
| | 2nd-stage autotags | 0.109 | −0.431 | 15.30% |
| | autotags | 0.104 | −0.445 | 14.20% |
| | random | 0.006 | −0.626 | 2.00% |

Table 6.10: Performance against Last.fm (top) and AMG (bottom) ground truth.

### 6.8.3 Second-Stage Learning

The second-stage autotags (6.10) are obtained by training a second set of boosted classifiers on the results of the first classifiers (that is, we train using autotags in place of audio features as input). This second step allows us to learn dependencies among tags. The results from the second-stage boosters for similarity are better than those of the first-stage boosters. This leads to the conclusion that there is much to gain from modeling dependencies among tags. However, this is largely an open question that needs more work: what model is the best for second-stage learning, and how can we best take advantage of correlation among tags?

| Wilco | |
|---|---|
| Ground truth Last.fm | Sufjan Stevens, Elliott Smith, The Flaming Lips, The Shins, Modest Mouse |
| Ground truth AMG | The Bottle Rockets, Blue Rodeo, The Flying Burrito Brothers, Neko Case, Whiskeytown |
| Last.fm | Calixico, Grandaddy, Modest Mouse, Mercury Rev, Death Cab for Cutie |
| Autotags | Tuatara, Animal Collective, Badly Drawn Boy, Gomez, Elliott Smith |
| Autotags 2nd stage | Badly Drawn Boy, Animal Collective, Elliott Smith, Gomez, Tuatara |

Table 6.11: Similar artists to Wilco from a) Last.fm ground truth b) AMG ground truth c) similarity from Last.fm tags d) similarity from autotags e) similarity from autotags second-stage.

| The Beatles | |
|---|---|
| Ground truth Last.fm | Radiohead, The Rolling Stones, Led Zeppelin, Pink Floyd, David Bowie |
| Ground truth AMG | George Martin, The Zombies, Duane Eddy, The Yardbirds, The Rolling Stones |
| Last.fm | George Harrison, The Who, The Rolling Stones, Fleetwood Mac, The Doors |
| Autotags | The Rolling Stones, Creedence Clearwater Revival, Elvis Costello, Elvis Costello & The Attractions, Traffic |
| Autotags 2nd stage | The Rolling Stones, Creedence Clearwater Revival, Donovan, The Lovin' Spoonful, Elvis Costello |

Table 6.12: Similar artists to The Beatles from a) Last.fm ground truth b) AMG ground truth c) similarity from Last.fm tags d) similarity from autotags e) similarity from autotags second-stage.

### 6.8.4 Discussion

It seems clear from these results that the autotags are of value. Though they do not outperform the social tags on which they were trained, it was shown in previous work (Eck et al. [30]) that they do yield improved performance when combined with social tags. At the same time, we showed a way to improve them by a second-stage of learning, and they are driven entirely by audio and so can be applied to new, untagged music.

Finally, we present some similar artists to Wilco and The Beatles in Tables 6.11 and 6.12, based on our two ground truths, Last.fm tags, and our two kind of autotags. We can draw two conclusions from these tables: Last.fm ground truth suffers from popularity bias, and our two set of autotag results are very comparable.

## 6.9 Conclusions

We have extended our previous autotagging method to scale more efficiently using FilterBoost. This introduces the concept of infinite training data, where an oracle can go and get the examples it needs. This is particularly appealing for web-based music recommenders that have access to millions of songs. The learning can simply be done using social tagging data, and the data we used [51] is freely available for research.

We tried to shed light on differences between *small and clean* versus *large and noisy* data sets. Though we provide no conclusive evidence to support the superiority of large, unreliably-labelled datasets such as our Last.fm data, we did demonstrate improved performance on the CAL500 task by adding audio which was never listened to by the CAL500 subjects and thus was not well-controlled. This is in keeping with the folk wisdom "There's no data like more data" and points towards methods which take advantage of all data available such as, e.g., semi-supervised learning and multi-task learning.

We have compared our method with the hierarchical mixture of Gaussians from the CAL group. This is to our knowledge the first comparison of algorithms especially designed for automatic tagging of music. In summary, Autotagger performed slightly better on precision for all feature sets while the GMM model performed slightly better for recall. We can in no way conclude from these results that one model is superior to the other. Test on larger datasets would be necessary to draw such conclusions. These results do support the conclusion that Autotagger has great potential as the core of a recommender that can generate transparent and steerable recommendation.

## 6.10 Future Work

One weakness of our current setup is that we blindly include all popular tags from Last.fm regardless of our ability to predict them. This adds significant noise to our similarity measurements. A solution proposed by Torres et al. [83] may prove more effective than our proposal to simply remove tags which we cannot classify above some threshold. A comparison of these and other methods is an important direction for future research.

Another direction for future research is that of second-stage learning. Treating tags as being independent is a useful assumption when you train on large scale data and you want to be able to expand your vocabulary easily. However, we show it is still possible to take advantage of the dependencies among tags, which improves our similarity results. We showed modest increases in classification error and also higher booster confidence values with our second-stage learning approach. However, more work is necessary in this area.

Finally, it should be possible to use the similarity space created by our model to create playlists that move smoothly from one artist to another. In addition, we can draw on our au-

totag predictions to explain the song-to-song transitions. As a first step, we used ISOMAP (see Bishop's book [19] for details) to generate a 2D projection of the artist similarity graph generated from the 360 Last.fm autotags I.1. We then calculated the shortest path from one artist to another. The autotag values from 6.3 are shown in 6.5 for the artist nodes in the shortest path. This is only an illustrative example and leaves many issues uninvestigated such as whether ISOMAP is the right dimensionality reduction algorithm to use. See `www.iro.umontreal.ca/~eckdoug/sitm` to listen to this example and others.

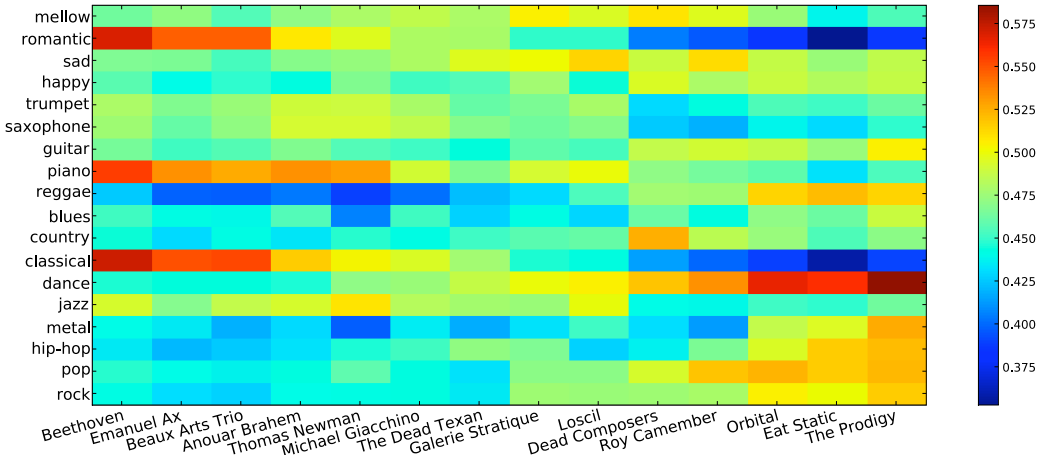


Figure 6.5: Shortest path between Ludwig van Beethoven and UK electronic music group The Prodigy after dimensionality reduction with ISOMAP. The similarity graph was built using all 360 Last.fm tags I.1, but the tags displayed are from 6.3.

## 6.11 Acknowledgement

Many thanks to the members of the CAL group, in particular Luke Barrington, Gert Lanckriet and Douglas Turnbull, for publishing the CAL500 data set and answering our numerous questions. Thanks to the many individuals that provided input, support and comments including James Bergstra, Andrew Hankinson, Stephen Green, the members of LISA lab, BRAMS lab and CIRMMT. Thanks to Joseph Turian for pointing us to the phrase "There's no data like more data." (originally from speech recognition, we believe).

# CHAPITRE 7

# APERÇU DU TROISIÈME ARTICLE

**Steerable Playlist Generation by Learning Song Similarity from Radio Station Playlists.**
François Maillet, Douglas Eck, Guillaume Desjardins et Paul Lamere
*Proceedings of the 10th International Conference on Music Information Retrieval*, Kobe, Japan, 2009

## 7.1 Contexte

Présentement, la recommandation musicale est souvent approchée comme la recommandation de n'importe quelle autre sorte d'item, soit en proposant une liste d'artistes ou de chansons similaires. C'est de cette manière qu'*Amazon* propose des livres et que *Netflix* recommande des films. La musique est pourtant une sorte de média totalement différente et effectuer des recommandations de cette manière ne tient pas compte de sa particularité. Tandis que plusieurs nouvelles interfaces de visualisations ont été proposées pour aider les utilisateurs à explorer l'espace musical[28], nous abordons le problème d'un autre angle.

Il existe différents profils d'auditeurs[25]. Bien que les mélomanes aiment passer du temps à fouiller des sites comme *Hype Machine*[1] pour découvrir de nouvelles chansons, il y a aussi plusieurs personnes qui sont très heureuses de faire une découverte passive. En effet, beaucoup de gens découvrent de nouvelles chansons simplement en écoutant ce qui joue comme musique de fond à leur café du coin ou encore sur le site Pandora. Dans ces deux cas, la musique est présentée sous forme de **liste d'écoute** (*playlist*), c'est-à-dire en faisant jouer plusieurs chansons une après les autres de manière cohérente et agréable.

Nous considérons en effet la génération de liste d'écoute automatique comme une extension naturelle de la recommandation musicale comme elle existe aujourd'hui. En utilisant le concept de personnalisation présentée au chapitre 3.2.2, un générateur de liste d'écoute devrait pouvoir choisir une série de chansons adaptées au contexte de l'utilisateur, en mélangeant des chansons connues et inconnues. Découvrir de la musique de cette façon nous semble plus naturel et elle devrait plaire aux mélomanes puisqu'on leur donne le contrôle sur le générateur.

Cet article tente donc d'aborder le problème de génération de séquences à court terme et d'y combiner toutes les idées de recommandation musicale présentées préalablement pour faire passer l'expérience des utilisateurs au prochain niveau.

Les algorithmes d'apprentissage utilisés sont un régresseur logisitque, un réseau de neurones ainsi que des SdA (voir sections 2.2.1 à 2.2.3).

[1] The Hype Machine : `http://hypem.com`

## 7.2 Contributions

Cet article présente une méthode basée sur deux nouvelles idées pour générer automatiquement des listes d'écoute personnalisables :

1. Utiliser un modèle d'apprentissage pour apprendre un espace de similarité musical où les chansons qui peuvent se suivre de manière harmonieuse dans une liste d'écoute seront similaires. Cet apprentissage se fait à partir de listes d'écoute de stations de radio commerciale ainsi que d'attributs extraits à partir des fichiers audio des chansons.

2. Permettre à l'utilisateur d'utiliser un nuage d'étiquettes personnalisable (voir section 3.2.2) pour que le générateur de liste d'écoute choisisse des chansons adaptées au contexte actuel de l'utilisateur.

Il n'existe pas de bonne réponse quand on se demande quelle chanson suivante jouer ; cela dépend grandement du contexte. La force du système repose sur le fait qu'il tient pleinement compte de la particularité du problème en utilisant deux composantes qui jouent un rôle très différent, mais complémentaire. Le modèle de similarité effectue d'abord le travail de bras en filtrant tous les candidats qui seraient de très mauvais choix comme prochaine chanson à jouer ; il a effectivement été entraîné spécifiquement pour réaliser cette tâche. Cela facilite alors grandement la sélection de la prochaine chanson basée sur le nuage d'étiquettes personnalisable et nous permet de sélectionner la chanson qui représente le mieux le contexte musical de l'utilisateur.

L'algorithme de génération est formalisé à la section 8.6.1.

## 7.3 Commentaires

L'auteur a réalisé la majorité des travaux reliés à cet article, incluant la conceptualisation du projet, la collecte des données, ainsi que la majorité de l'implémentation et de l'écriture.

Ce qui rend le problème de la génération de listes d'écoute si difficile est probablement qu'il n'existe pas de véritable métrique pour déterminer ce qui constitue une *bonne* liste d'écoute. Il existe en effet plusieurs types de listes qui ont chacune leur particularité. Veut-on créer une liste pour une personne proche, pour exprimer notre étant d'âme, comme musique de fond pour un souper, etc.[26] L'évaluation des générateurs automatiques est donc difficile et subjective.

Un article publié au même moment que le nôtre présente plusieurs éléments intéressants. En effet, dans [7], les auteurs ont effectué des expériences pour comparer *Genius*, le générateur de listes d'écoute présent dans *iTunes* à, entre autres, un générateur basé sur la similarité dans l'espace des étiquettes. Ils ont utilisé leur système d'application automatique d'étiquettes[2][86] pour

[2]Il s'agit du même modèle *GMM* dont il a été question à la section 5.2.2.

leurs expériences qui furent réalisées avec des sujets humains. Ils ont pu tirer plusieurs conclusions très intéressantes, comme le fait que la sonorité était l'élément que les sujets ont trouvé le plus important lorsqu'ils ont apprécié une liste d'écoute. Ils ont aussi montré que *Genius*, qui est basé sur le FC, semble indirectement capturer les mêmes particularités des chansons que les approches basées sur le contenu.

Ils concluent aussi qu'une combinaison de la similarité basée sur le contenu audio (pour éviter les mauvais choix) à une similarité basée sur des métadonnées (pour pouvoir offrir de la transparence) aiderait à construire des systèmes de recommandations plus intelligents. Cette conclusion est intéressante puisque notre générateur de listes d'écoute est basé sur ces mêmes concepts, en ajoutant en plus la personnalisation du nuage d'étiquettes.

# CHAPITRE 8

# STEERABLE PLAYLIST GENERATION BY LEARNING SONG SIMILARITY FROM RADIO STATION PLAYLISTS

## 8.1 Abstract

This paper presents an approach to generating steerable playlists. We first demonstrate a method for learning song transition probabilities from audio features extracted from songs played in professional radio station playlists. We then show that by using this learnt similarity function as a prior, we are able to generate steerable playlists by choosing the next song to play not simply based on that prior, but on a tag cloud that the user is able to manipulate to express the high-level characteristics of the music he wishes to listen to.

## 8.2 Introduction

The celestial jukebox is becoming a reality. Not only are personal music collections growing rapidly, but online music streaming services like Spotify[1] or Last.fm[2] are getting closer everyday to making all the music that has ever been recorded instantly available. Furthermore, new playback devices are revolutionizing the way people listen to music. For example, with its Internet connectivity, Apple's iPhone gives listeners access to a virtually unlimited number of tracks as long as they are in range of a cellular tower. In this context, a combination of personalized recommendation technology and automatic playlist generation will very likely form a key component of the end user's listening experience.

This work's focus is on providing a way to generate steerable playlists, that is, to give the user high-level control over the music that is played while automatically choosing the tracks and presenting them in a coherent way. To address this challenge, we use playlists from professional radio stations to learn a new similarity space based on song-level audio features. This yields a similarity function that takes audio files as input and outputs the probability of those audio files being played successively in a playlist. By using radio station playlists, we have the advantage of having a virtually unlimited amount of training data. At the same time, we are able to generalize the application of the model to any song for which we have the audio files. We believe this will be the case in any real-life application we can foresee for the model.

Furthermore, we use the concept of a steerable tag cloud[38] to let the user guide the playlist generation process. Tags[48], a type of meta-data, are descriptive words and phrases applied

[1] `http://www.spotify.com`
[2] `http://www.last.fm`

to any type of item; in our case, music tracks. Tags are words like like *chill*, *violin* or *dream pop*. They have been popularized by Web 2.0 websites like Last.fm, where users can apply them to artists, albums and tracks. The strength of tags, especially when used in a social context, lies in their ability to express abstract concepts. Tags communicate high-level ideas that listeners naturally use when describing music. We tag all tracks in our playlists using an automatic tagging system[18] in order to ensure that they are all adequately tagged. Then, given a seed song, the learnt similarity model is used to preselect the most probable songs to play next, after which the similarity between the user's steerable tag cloud and each of the candidate songs' cloud is used to make the final choice. This allows users to steer the playlist generator to the type of music they want to hear.

The remainder of this paper is organized as follows. Section 8.3 gives a brief overview of related work in music similarity and playlist generation. Section 8.4 explains how the radio station playlists data set was collected and assembled. Section 8.5 presents the creation and evaluation of our new similarity space. Section 8.6 explains how we propose implementing a steerable playlist generator. Finally, section 8.7 explores future research avenues.

### 8.3 Previous work

An increasing amount of work is being conducted on automatic playlist generation, with considerable focus being placed on the creation of playlists by means of acoustic or meta-data similarity [3, 34, 60, 61, 72].

More recently, connectivity graphs derived from music social networks are being used to measure similarity. For example, [33] uses network flow analysis to generate playlists from a friendship graph for MySpace[3] artists. In [37], the authors use Last.fm collaborative filtering data to create a similarity graph considering songs to be similar if they have been listened to by similar users. They then embed the graph into a Euclidean space using landmark multidimensional scaling (LMDS), where similar artists would appear near one another.

Another approach[6] uses a case-based reasoning system. From its pool of real human-compiled playlists, the system selects the most relevant ones in regards to the user's one song query and mixes them together, creating a new playlist.

We are aware of only one other attempt to use radio station playlists as a source of data. In [74] radio station playlists are used to construct a weighted graph where each node represents a song and each arc's weight is the number of times the two songs are observed one after the other. From the graph, the authors are able to infer transition probabilities between songs by creating a Markov random field. Our approach is similar, with the advantage that we can generalize to songs not observed in the training data.

[3] `http://www.myspace.com`

## 8.4 Constructing the data set

Our model is trained on professional radio station playlists. For this experiment, we consider a playlist to be a sequence of 2 or more consecutive plays uninterrupted by a commercial break. Suppose a radio station plays the tracks $t_a$, $t_b$ and $t_c$ one after the other, we will consider $\{t_a, t_b\}$ and $\{t_b, t_c\}$ as two 2-song sequences $\in \mathbb{S}^2$, and $\{t_a, t_b, t_c\}$ as one 3-song sequence $\in \mathbb{S}^3$. We consider the sequences $\{t_a, t_b\}$ and $\{t_b, t_a\}$ as two distinct sequences. The model's output will thus be non-symmetric in regards to the order in which the songs are presented.

The playlist data we used came from two sources which we will cover in section 8.4.1.

### 8.4.1 Playlist sources

#### 8.4.1.1 Radio Paradise

Radio Paradise[4] (RP) is a free Internet-streamed radio station that defines its format as "eclectic online radio." RP provided us with playlist data including every play from January 1st 2007 to July 28th 2008 (575 days). The data consists of 195,692 plays, 6,328 unique songs and 1,972 unique artists.

#### 8.4.1.2 Yes.com

Yes.com is a music community web site that provides, among other things, the playlists for thousands of radio stations in the United States. Developers are able to access the playlist data via a free web based API[97] that returns the data in JSON format. One API call allows the developer to get a list of radio stations, either by searching by genre, by name or even by proximity to a given ZIP code. Then, for each retrieved station, the API provides access to that station's play history for the last 7 days. The self-assigned and non-exclusive genres of the available radio stations cover all major musical styles. The stations we used to build our own dataset were not chosen for any particular reason. Rather, we made a few searches with the API by genre until we obtained enough data for our work, that is 449 stations. The proportion of stations' non-exclusive association with the different genres is detailed in Table 8.1.

We used data mined from the Yes API from November 13th 2008 to January 9th 2009 (57 days), totaling 449 stations, 6,706,830 plays, 42,027 unique songs and 9,990 unique artists.

Unlike RP, Yes did not provide any indication of where the commercial breaks were located in the list of plays. We inferred where they were by looking at the interval between the start time of every pair of consecutive songs. As we will explain in section 8.5.1, we only used sequences made of tracks for which we had the audio files. This allowed us to calculate song length and to infer when commercials were inserted. Specifically, if the second of the two songs did not start

[4] `http://www.radioparadise.com`

Table 8.1: Proportion of the 449 Yes radio stations associated with each genre. Because the genres are non-exclusive the sum of the percentages is $> 100$.

| | | | |
|---|---|---|---|
| Latin | 11.2% | Christian | 11.4% |
| Country | 20.6% | Hip-Hop | 17.2% |
| Jazz | 4.3% | Metal | 14.1% |
| Pop | 23.3% | Punk | 1.6% |
| Rock | 39.4% | R&B/Soul | 13.6% |

within $\pm 20$ seconds of the end of the first one, we assumed that a commercial break had been inserted and thus treated the two songs as non-sequential. This approach is more precise than the method used in [74], where breaks were inserted if the elapsed time between two songs was greater than 5 minutes.

### 8.4.2 Putting the data together

Combining all the data yielded 6,902,522 plays, with an average of 15,338 plays per station. As we will explain in 8.5.1, the features we used as input to our model required us to have access to each of the songs' audio file. Of the 47,044 total songs played in the playlists we used, we were able to obtain the audio files for 7,127 tracks. This reduced the number of distinct usable song sequences to 180,232 and 84,668 for the 2 and 3-song sequence case respectively. The sequences for which we had all the audio files were combinations from 5,562 tracks.

Finally, we did not possess a set of explicit negative examples (i.e. two-song sequences that a radio station would never play). In order to perform classification we needed examples from both the positive and negative class. To address this, we considered any song sequence that was never observed in the playlist as being a negative example. During training, at each new epoch, we randomly sampled a new set of negative examples matched in size to our positive example set. With this strategy it is possible that we generated false-negative training examples (i.e. two-song sequences that we didn't see as positive examples in our data set but that in fact a radio station would play). However, since we resample new negative examples after every training epoch, we do not repeatedly show the model the same false-negative pair, thus minimizing potential impact on model performance.

## 8.5 Song similarity model

### 8.5.1 Features

We use audio-based features as input to our model. First, we compute 176 frame-level autocorrelation coefficients for lags spanning from 250ms to 2000ms at 10ms intervals. These are aggregated by simply taking their mean. We then down sample the values by a factor of two, yielding 88 values. We then take the first 12 Mel-frequency cepstral coefficients (MFCC), calculated over short windows of audio (100ms with 25ms overlaps), and model them with a single Gaussian (G1) with full covariance[64]. We unwrap the values into a vector, which yields 78 values.

We then compute two song-level features, danceability[79] and long-term loudness level (LLML)[91]. Danceability is a variation of detrended fluctuation analysis, which indicates if a strong and steady beat is present in the track, while the LLML gives an indication of the perceived loudness of the track. Both of these features yield a single numeric value per song.

These 4 audio features are concatenated to form an 168 dimensional vector for each track.

### 8.5.2 Learning models

We formulate our learning task as training a binary classifier to determine, given the features for a sequence of tracks, if they form a song sequence that has been observed in our radio station playlists. If a sequence has been observed at least once, it is considered a positive example. As mentioned above, the negative examples are randomly sampled from the pool of all unseen sequences.

We use three types of learning models in our experiments: logistic regression classifiers, multi-layer perceptrons (MLP) and stacked denoising auto-encoders (SdA). Logistic regression, the simplest model, predicts the probability of a song-sequence occurrence as a function of the distance to a linear classification boundary. The second model, a multi-layer perceptron, can also be interpreted probabilistically. It adds an extra “hidden" layer of non-linearity, allowing the classifier to learn a compact, nonlinear set of basis functions.

We also use a type of deep neural network called a stacked denoising auto-encoder (SdA)[93]. The SdA learns a hierarchical representation of the input data by successively initializing each of its layers according to an unsupervised criterion to form more complex and abstract features. The goal of this per-layer unsupervised learning is to extract an intermediate representation which preserves information content whilst being invariant to certain transformations in the input. SdAs are exactly like neural networks with the exception that they have multiple hidden layers that are initialized with unsupervised training.

In our experiments, the models operated directly on pairs (or 3-tuples in the case of predicting

sequences of length 3) of audio features. The input $x$ of our model is thus a vector of length $180 \cdot n$, with $n \in \{2,3\}$, formed by concatenating the features of each track into a single vector.

We used 75% of our unique pairs/triplets for training, keeping 12.5% for validating the hyper-parameters and 12.5% for testing. We did not perform any cross-validation.

### 8.5.3 Similarity evaluation

Measuring the quality of the similarity space induced by the model is not easy and highly subjective. We will first look at its performance on the learning task (8.5.3.1), and then try to evaluate it in a more qualitative way (8.5.3.2).

#### 8.5.3.1 Learning performance

Classification errors for the different models we trained are presented in Table 8.2. The errors represent the proportion of real sequences that were classified as false sequences by each model, or vice versa, on the test set, for the best combination of hyper-parameters.

While the logistic regression clearly lacks learning capacity to adequately model the data, the MLPs and SdAs have similar performance. SdAs have been shown to outperform MLPs in complex image classification tasks ([93]) but were unable to learn a significantly better representation of the features we are using for this task. This could mean that the feature set was not sufficiently rich or that the task was simply too difficult for the hierarchical model to find any kind of compositional solution to the problem.

Table 8.2: Classification errors on the test set for the different models we trained as well as a random baseline. SdA-*n* represents an SdA with *n* hidden layers.

| Model | 2-song seq. | 3-song seq. |
|---|---|---|
| random | 50.00% | 50.00% |
| logistic regression | 31.73% | 21.08% |
| MLP | 8.53% | 5.79% |
| SdA-2 | 8.38% | 5.58% |
| SdA-3 | 8.62% | 5.78% |

#### 8.5.3.2 Retrieval evaluation

By using the original radio station playlists as the ground truth, we can evaluate the retrieval performance of our model. The evaluation is done using *TopBucket* (TB) [18], which is the proportion of common elements in the two top-*N* lists.

Table 8.3: Retrieval performance based on the *TopBucket* (TB) measure of our models compared to random, popularity-biased random, acoustic similarity and autotags similarity. Each score represents the average percentage (and standard error) of songs in the ground truth that were returned by each model.

| | **2-song sequences** | | **3-song sequences** | |
|---|---|---|---|---|
| Model | TB10 | TB20 | TB5 | TB10 |
| random | 0.25%±0.01% | 0.58%±0.08% | 0.11%±0.04% | 0.25%±0.12% |
| popularity-biased random | 1.01%±0.11% | 2.19%±0.14% | 0.51%±0.09% | 0.96%±0.25% |
| acoustic (G1C) | 1.37%±0.13% | 2.37%±0.16% | 0.63%±0.10% | 1.61%±0.32% |
| autotags (Cosine distance) | 1.43%±0.14% | 2.34%±0.17% | 0.58%±0.10% | 2.26%±0.39% |
| logistic regression | 2.47%±0.18% | 6.41%±0.28% | 0.20%±0.06% | 1.16%±0.27% |
| MLP | 16.61%±0.50% | 23.48%±0.57% | 7.72%±0.41% | 20.26%±1.43% |
| SdA-2 | 13.11%±0.42% | 19.13%±0.48% | 7.25%±0.40% | 21.74%±1.59% |
| SdA-3 | 13.17%±0.39% | 18.22%±0.51% | 9.74%±0.52% | 26.39%±1.83% |

Constructing the ground truth from the playlists is done as follows. Each 2-song sequence $S_n^2 \in \mathbb{S}^2$ is made up of tracks $\{t_n^1, t_n^2\}$ and has been observed $|S_n^2|$ times. We construct one top list $\mathscr{L}_{t_i} \forall t_i \in \mathbb{T}$, as the set of all sequences $S_n^2$ for which $t_n^1 = t_i$, ordered by $|S_n^2|$. $\mathscr{L}_{t_i}$ essentially gives a list of all the tracks that have followed $t_i$ ordered by their occurrence count. In the 3-song sequence case, we construct a top list $\mathscr{L}_{\{t_i,t_j\}}$ for pairs of tracks since in practice, a playlist generation algorithm would know the last two songs that have played.

For our experiments, we used the top 10 and 20 elements and 5 and 10 elements for the 2 and 3-song sequence case respectively. The results, shown in Table 8.3, represent the average number of common elements in the ground truth's top list and each of the similarity models' for every song.

Because most sequences were only observed once ($|S_n^2| = 1$), we were often in the situation where all the sequences in $\mathscr{L}_{t_i}$ had an occurrence of 1 ($\forall S_n^2 \in \mathscr{L}_{t_i} : |S_n^2| = 1$) and the number of sequences in $\mathscr{L}_{t_i}$ was greater than $N$ for a top-$N$ list. Because in such a situation there was no way to determine which sequences should go in the top-list, we decided to extend the top-$N$ list to all the sequences that had the same occurrence count as the $N^{th}$ sequence. In the 2-song sequence case, we also ignored all sequences that had $|S_n^2| = 1$ to keep the top-$N$ lists from growing a lot larger than $N$. Ignoring as well all the songs that did not have at least $N$ tracks in their top-list, we were left, in the 2-song sequence case, with 834 songs that had an average of 14.7 songs in their top-10 list and 541 songs with an average of 28.8 songs in their top-20 list. In the 3-song sequence case, the top-5 list was made up of 1,181 songs with a 6.6 top songs average and the top-10 list was composed of 155 songs with an average of 13.8 top songs.

We compared our model's retrieval performance to two other similarity models. First, we computed the similarity in the space of autotags[18] from the cosine distance over song's tags vector[38]. The second comparison was performed by retrieving the most acoustically similar songs. Acoustic similarity was determined by using G1C[71] which is a weighted combination of spectral similarity and information about spectral patterns. We also compared our model to a popularity-biased random model that probabilistically chooses the top songs based on their popularity. Each song's popularity was determined by looking at the number of sequences it is part of.

In the 3-song sequence case, for the autotag and acoustic similarity, we represent the similarity $sim(\{t_1,t_2\},t_3)$ as the mean of $sim(t_1,t_3)$ and $sim(t_2,t_3)$.

The results of Table 8.3 clearly show that there is more involved than simple audio similarity when it comes to reconstructing sequences from radio station playlists. The performance of the audio and autotag similarity are indeed significantly lower than models that were trained on actual playlists.

Furthermore, the TB scores of Table 8.3 are from the models that have the best classification error (see Table 8.2). It is interesting to note that some models with a worst classification error

have better TB scores. While classification is done by thresholding a model's certainty at 50%, TB gives an indication of the songs for which a model has the highest certainty. Since these are the songs that will be used when generating a playlist, this metric seems more appropriate to judge the models. The relation between classification error and TB scores is a topic for further investigation.

## 8.6 Steerable playlist generation

While the model presented above is able to build a similarity space in which nearby songs fit well together in a playlist, it does not provide a mechanism for allowing the user to personalize the sequence for a given context. To address this, final song selection was done using the *Aura*[5][38] recommendation engine from Sun Microsystems Labs. Aura is able to generate transparent and steerable recommendations by working with a textual representation — a tag cloud — of the items it is recommending. Specifically it finds the most similar items to any other in its pool by computing the cosine distance on their respective tag clouds. It can also explain to the user why an item is recommended by showing the overlap between tag clouds.

We use Aura as a means to allow users to personalize ("steer") the playlist generation by allowing them to create a personal tag cloud that represents the music they wish to listen to. In order to generate tag clouds for our tracks, we used *Autotagger*[18], a content-based machine learning model. This model is designed to generate a tag cloud (specifically a weighted vector of 360 music-relevant words) from an audio track, thus allowing us to use Aura's cosine distance measure to compute the similarity between each track and the user's personal cloud.

### 8.6.1 Steps for generating a steerable playlist

Our playlist generation algorithm works as follows :

1. A seed track $t_s \in \mathbb{T}$ is selected amongst all possible tracks.

2. The similarity model is used to compute transitional probabilities between the seed song and all other ones (with more similar songs having higher transition probabilities), keeping only the top $\varphi$, or thresholding at a certain transition probability $\rho$. Let $\mathscr{T}$ be the group of these top songs:

$$\mathscr{T} = \underset{t_i \in \mathbb{T} \setminus t_s}{\arg\max^{\varphi}} \mathscr{M}(t_s, t_i) \tag{8.1}$$

3. The user is then invited to create a tag cloud $\mathscr{C}_U$ by assigning weights to any of the 360 tags in the system. In this way the cloud is personalized to represent the mood or type of songs

[5] `http://www.tastekeeper.com`

the user would like to hear. The higher the weight of a particular tag, the more impact it will have on the selection of the next song.

4. Autotagger is used to generate a tag cloud $\mathscr{C}_{t_j}$ for all tracks $t_j \in \mathscr{T}$. The cosine distance ($cd(\cdot)$) between these tag clouds and $\mathscr{C}_U$ is used to find the song that best matches the abstract musical context the user described with his or her cloud:

$$t_{min} = \arg\min_{t_j \in \mathscr{T}} cd(\mathscr{C}_U, \mathscr{C}_{t_j}) \tag{8.2}$$

5. The track $t_{min}$ is selected to play next. Since the system is transparent, we can tell the user we chose the song $t_{min}$ because it has a certain transition probability from the seed song but also because its tag cloud overlapped with $\mathscr{C}_U$ in a particular way. The user can then go back and modify the tag cloud $\mathscr{C}_U$ to influence how subsequent songs will be selected.

Naturally, a lot of extra factors can be used when determining which song to play in step 4. For instance, we could consider the user's taste profile to take into account what types of songs he normally likes, mixing his current steerable cloud to the one representing his musical tastes. We could also include a discovery heuristic to balance the number of novel songs selected as opposed to ones the user already knows. Furthermore, we also implicitly keep a list of the tracks we have already played and ignore them when constructing $\mathscr{T}$ at step 2.

### 8.6.2 Example playlists

To illustrate the effect of the steerable tag cloud, we generate two playlists seeded with the same song but with very different steerable clouds. The first 9 iterations of both playlists are shown in Table 8.4. The effect of the cloud is clearly visible by the different direction each playlist takes. In our view, this transition is done smoothly because it is constrained by the underlying similarity model.

To visualize the similarity space and the playlist generating algorithm, we compute a full track-to-track similarity matrix and reduce its dimensionally using the t-SNE[90] algorithm (see Figure 8.1). We chose t-SNE because it tends to retain local distances while sacrificing global distances, yielding an appropriate two-dimensional visualization for this task (i.e. the distance between very similar songs is more important to us than the relative global placement of, e.g., jazz with respect to classical). We have overlaid the trajectory of the two playlists in Table 8.4 to illustrate their divergence.

Table 8.4: Both the following playlists are seeded with the song *Clumsy* by *Our Lady Peace*. To give a clear point of reference, we use the tag clouds of actual songs as the steerable cloud. The *mellow* tag cloud is made up of the tags for *Imagine* by *John Lennon* and the *hard* tag cloud with the tags for *Hypnotize* by *System of a Down*.

| **Mellow tag cloud** |
|---|
| Viva la Vida by Coldplay |
| Wish You Were Here by Pink Floyd |
| Peaceful, Easy Feeling by Eagles |
| With or Without You by U2 |
| One by U2 |
| Fields Of Gold by Sting |
| Every Breath You Take by The Police |
| Gold Dust Woman by Fleetwood Mac |
| Enjoy The Silence by Depeche Mode |
| **Hard tag cloud** |
| All I Want by Staind |
| Re-Education (Through Labor) by Rise Against |
| Hammerhead by The Offspring |
| The Kill by 30 Seconds To Mars |
| When You Were Young by The Killers |
| Hypnotize by System of a Down |
| Breath by Breaking Benjamin |
| My Hero by Foo Fighters |
| Turn The Page by Metallica |

## 8.7 Conclusions

We have demonstrated a method for learning song similarity based on radio station playlists. The learnt model induces a new space in which similar songs fit well when played successively in a playlist. Several classifiers were evaluated on a retrieval task, with SdAs and MLPs performing better than other similarity models in reconstructing song sequences from professional playlists. Though we were unable to show that SdAs outperform MLPs, we did show much better performance than logistic regression and measures such as G1C over standard audio features. Furthermore we argue that our model learns a direct similarity measure in the space of short song sequences rather than audio or meta-data based similarity. Finally, we showed a way of doing steerable playlist generation by using our similarity model in conjunction with a tag-based distance measure.

Though this model is only a first step, its power and simplicity lie in the fact that its two components play very different but complementary roles. First, the similarity model does the

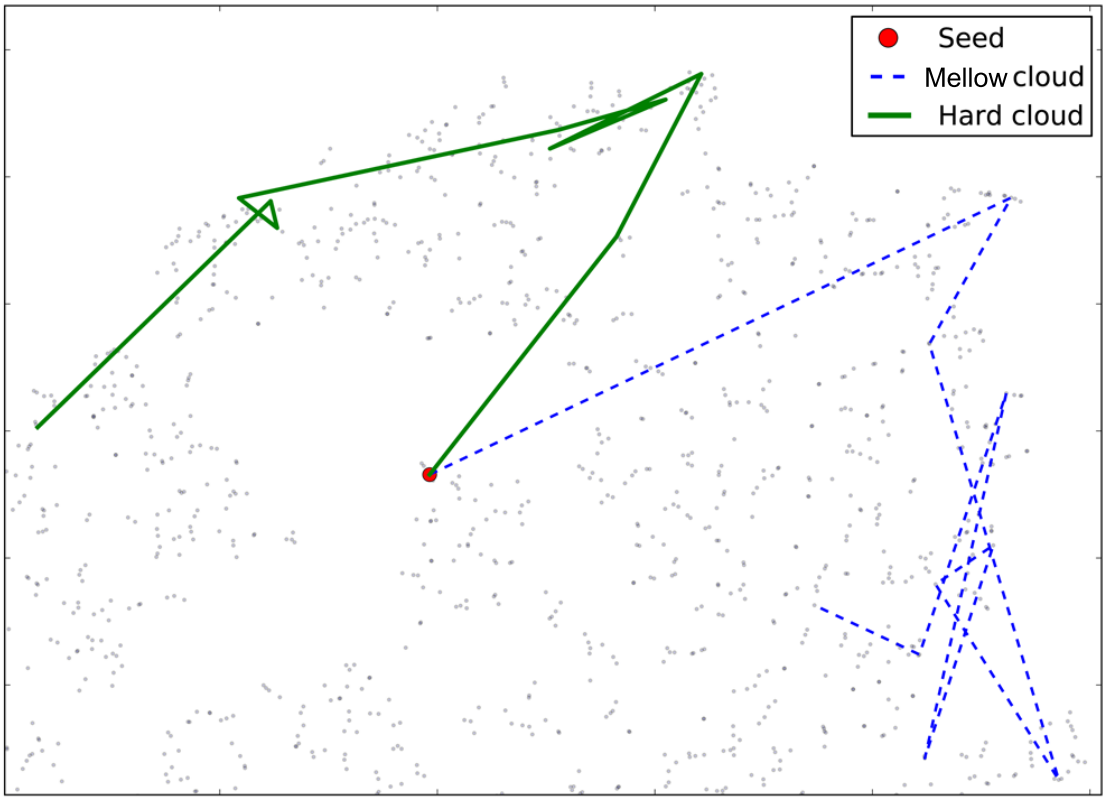


Figure 8.1: Part of the 2-d representation of the track-to-track similarity matrix generated by a 2-song sequence SdA model. This 2-d representation is obtained by running the t-SNE algorithm on the similarity matrix, giving us positions for each track, which are represented by the grey dots. The arcs represent the trajectories of the two playlists described in Table 8.4 and they are overlaid over the tracks. Both playlists are seeded with the same song, which is represented by the bigger dot. Each playlist diverges because of the steerable tag cloud that is guiding its generation.

*grunt work* by getting rid of all unlikely candidates, as it was trained specifically for that task. This then greatly facilitates the steerable and fine-tuned selection of the subsequent track based on the textual aura, as most of the obvious bad picks have already been removed.

Future work should attempt to use the number of occurrences of each sequence to give more importance to more reliable sequences. Also, the model might learn a better similarity space by being trained with richer features as input. For example, adding meta-data such as tags, a measure of popularity, the year the song was recorded, etc., might prove helpful. Such a richer input space is likely necessary to show a performance gain for SdAs over competing probabilistic classifiers. Our experiments also led us to believe that an increase in the quality of the learnt similarity could probably be attained by simply adding more training data, something that can be easily accomplished as thousands of songs are played on the radio everyday.

# CHAPITRE 9

# CONCLUSION

Les travaux présentés dans ce mémoire étaient motivés par le problème de la recommandation musicale, qui est à ce jour loin d'être résolu et suscite beaucoup d'intérêt autant dans le monde académique qu'industriel. Nous concluons en résumant tout d'abord chacun des trois articles présentés pour ensuite explorer des avenues de recherche futurs.

## 9.1 Résumé des articles

### 9.1.1 Generating Transparent, Steerable Recommendations from Textual Descriptions of Items

Nous avons présenté au chapitre 3 un système de recommandation musicale hybride utilisant des étiquettes comme représentation des items. Cette approche permettait d'éviter les problèmes typiques des systèmes de FC, tels le problème du démarrage à froid ainsi que le bias de popularité.

De plus, les recommandations du système sont transparentes, de sorte qu'il est possible d'expliquer à l'utilisateur pourquoi un item a été recommandé. Nous donnons aussi plein contrôle à l'utilisateur via l'interface du nuage d'étiquettes personnalisable pour qu'il puisse exprimer exactement le type de musique qu'il souhaite entendre.

Une évaluation de la qualité des recommandations fut réalisée à l'aide d'un sondage sur le web où 200 participants évaluèrent la qualité de notre système, d'un système de FC ainsi que des recommandations d'experts en musique. Bien qu'il ne faille pas accorder trop d'importance à ce sondage, notre système termina premier, ce qui démontre à tout de moins que notre approche, qui ne souffre pas du problème du démarrage à froid, est comparable aux systèmes de FC.

De plus, nous avons effectué une petite étude par rapport à la convivialité du *Music Explaura*, qui est l'implémentation de l'interface du nuage d'étiquettes personnalisable. Bien que l'interface eut semé un peu de confusion, les testeurs ont grandement apprécié pouvoir avoir un impact les recommandations.

### 9.1.2 Autotagger : a model for predicting social tags from acoustic features on large music databases

Principalement pour adresser le problème du démarrage à froid dans des systèmes de recommandation comme celui présenté à la section précédente, nous avons développé un modèle d'application automatique d'étiquettes à des pièces musicales, présenté au chapitre 5. Le système, appelé *Autotagger*, utilise Filterboost, ce qui lui permet de fonctionner sur des ensembles

de données extrêmement grands. Les caractéristiques utilisées sont extraites des fichiers sonores, ce qui permet au modèle de fonctionner avec des pièces musicales complètement inconnues. Nous avons entrainé un total de 360 modèles à chacun déterminer si une étiquette spécifique s'applique à une chanson ou non.

Nous avons aussi testé deux méthodes pour permettre à notre modèle de prendre avantage de la corrélation qui existe entre les étiquettes dans les données sociales. La première est d'entraîner une deuxième série de classifieurs en utilisant la sortie de la première série en entrée. La deuxième est de combiner la prédiction des différents modèles de manière proportionnelle à la corrélation empirique des étiquettes dans les données sociales. Ces deux opérations améliorent légèrement la performance de notre modèle.

Nous avons aussi comparé *Autotagger* à une approche basée sur les GMM en utilisant un ensemble de données commun, soit l'ensemble *CAL500*. Par contre, puisque cet ensemble de données est très petit, nous avons conclu qu'il n'était pas possible de déterminer de manière adéquate quel modèle est supérieur.

### 9.1.3 Steerable Playlist Generation by Learning Song Similarity from Radio Station Playlists

Bien que l'on fasse l'expérience de la musique de manière fondamentalement différente d'autres types de média comme les films ou les livres, elle est généralement recommandée de la même manière. Nous voyons la génération de listes d'écoute comme une extension naturelle à la manière dont la musique est présentement recommandée.

Nous présentons au chapitre 7 un système de génération de listes d'écoute qui fonctionne en deux étapes. Premièrement, un modèle est entraîné en utilisant des listes d'écoute de stations de radio commerciale à déterminer quelles sont des séquences de chansons qui peuvent se suivre de manière harmonieuse dans une liste d'écoute. Les différents modèles que nous avons testés sont la régression logistique, un réseau de neurones ainsi que des *SdA*.

La deuxième étape consiste à utiliser un nuage d'étiquettes personnalisable pour permettre à l'utilisateur d'exprimer d'une manière abstraite les caractéristiques de la musique qu'il désire écouter. On peut donc ensuite comparer ce nuage personnalisable à celui de n'importe quelle chanson pour déterminer quelles sont celles qui sont le plus similaires.

En générant des listes d'écoute de cette manière, on peut d'une part s'assurer d'avoir de bonnes transitions entre les chansons tout en respectant le contexte musical actuel de l'utilisateur.

## 9.2 Discussion et travaux futurs

### 9.2.1 Taille des ensembles de données communs

Un des problèmes qui revient souvent est l'absence d'ensemble de données d'entraînement ou d'évaluation assez grands ou complets[54]. Dans le cas de l'application automatique d'étiquettes, il existe par exemple les ensembles *Majorminor* et *CAL500*, mais comme il est expliqué à la section 5.3, leur taille est trop petite pour évaluer efficacement notre algorithme. Un nouvel ensemble de données, MagnaTagatune[56], qui a été généré à l'aide d'un jeu comme *Majorminor*, a été rendu public cette année et est beaucoup plus gros. Les divers algorithmes dans la littérature devraient donc être testés sur cet ensemble de données.

Il serait de plus souhaitable qu'une entreprise comme Pandora rende publique une partie de leur ensemble de données. On pourrait de cette manière tester les algorithmes sur une base de données très propre et très grosse. En ce moment, le compromis entre *petit et propre* et *gros mais bruité* rend une évaluation réaliste difficile.

Notons aussi qu'une expérience pour déterminer la validité des métriques présentement utilisées a été réalisée dans [57]. Les auteurs montrent que la métrique qu'ils proposent, la métrique *TagATune*, est plus robuste à certains problèmes qu'ont les métriques comme la précision et le rappel. Plus d'expériences sont donc nécessaires pour déterminer ce qu'est une performance que l'ont peut juger comme acceptable pour un système commercial.

### 9.2.2 Nettoyer les ensembles de données bruités

Nos travaux sont basés sur l'utilisation d'un gros ensemble de données bruité, soit les données sociales du site Last.fm. L'expérience des utilisateurs, dans le cas de la recommandation, et la performance d'*Autotagger*, dans le cas de l'application automatique d'étiquettes, bénéficierait fort probablement d'un nettoyage et d'un groupage (*clustering*) automatique des données. En effet, traiter des étiquettes telles que *canada* et *canadien* comme étant différentes sème la confusion pour un utilisateur et est inefficace pour un algorithme. En combinant de telles étiquettes, on obtiendrait des données plus denses et moins bruitées. Des travaux ont déjà été effectués sur ce sujet, voir par exemple [78].

De plus, pouvoir expliquer les recommandations aux utilisateurs en utilisant des phrases au lieu d'un nuage d'étiquettes améliorerait probablement leur expérience. Cela a été réalisé dans [86] mais en utilisant l'ensemble de données *CAL500*, qui utilise un vocabulaire fixé à l'avance, ce qui permet aux auteurs de savoir exactement quelle sorte de mot chaque étiquette représente. Nos données sociales sont issues d'un vocabulaire libre, ce qui complique cette tâche. Nous devrions tenter d'utiliser des ressources linguistiques comme WordNet[1] pour extraire le sens des

[1] WordNet : `http://wordnet.princeton.edu`

diverses étiquettes.

### 9.2.3 Faire le pont entre le FC et le contenu

Ultimement, nous aimerions savoir comment combiner l'information que nous obtenons des données d'interaction utilisées par le FC ainsi que des données extraites du contenu. Il a déjà été montré dans [30] que combiner l'information provenant des étiquettes générées automatiquement aux étiquettes sociales améliore ces dernières dans des tests de similarité musicale. De plus, les tests réalisés par [7] ont montré que les étiquettes automatiques fonctionnent aussi bien que le FC dans certains cas pour la génération de liste d'écoute.

Il semble évident que des travaux devraient se pencher à trouver une manière de combiner toutes ces informations en un seul système qui pourrait utiliser les données issues du contenu pour régler les problèmes connus du FC et aussi probablement améliorer sa performance quand les données d'interactions sont disponibles.

### 9.2.4 Améliorations à *Autotagger*

Plusieurs améliorations à *Autotagger* sont proposées dans le mémoire du premier auteur de l'article[16]. Il propose par exemple de s'intéresser aux attributs parcimonieux (*sparse features*), par exemple dans [68]. Notons qu'il est possible d'obtenir de tels attributs en utilisant un auto-encodeur débruiteur. Il propose aussi d'utiliser un oracle plus intelligent avec Filterboost. Dans [45], l'algorithme est modifié de manière à effectuer un échantillonnage intelligent de type Metropolis-Hastings, ce qui semble aider l'apprentissage. Finalement, on peut aussi tenter de déterminer quelles étiquettes peuvent bien s'apprendre. Cela revient à se demander quelles étiquettes ont un lien direct avec l'information que l'on peut extraire des fichiers audio. Une étiquette comme *favorites* n'est par exemple pas un descripteur de l'audio, mais plutôt une forme de donnée d'interaction active que l'on devrait tenter d'identifier et de traiter différemment.

### 9.2.5 Listes d'écoute collaboratives

Une amélioration évidente au générateur de liste d'écoute que nous avons présenté est de permettre la personnalisation de la liste par plusieurs utilisateurs simultanément. De nouveaux travaux ont été récemment publiés à ce sujet.

Dans [5], l'auteur présente une méthode très complète appelée le *Poolcasting*, qui est basée sur le raisonnement par cas (*case-based reasoning* ou CBR). Il détermine que sont de bonnes séquences de chansons dans une liste d'écoute en analysant des listes issues de sites sociaux et tient aussi compte des goûts musicaux de plusieurs auditeurs simultanément.

Une autre approche[32], qui est basée sur [33] où l'analyse des graphes sociaux sur le site MySpace est utilisée pour construire des listes d'écoute aussi par CBR, propose de simplement

choisir la dernière chanson de la liste d'écoute par un vote des utilisateurs inclus dans le système. Contrairement au *Poolcasting* où la sélection des chansons est faite itérativement, cette méthode génère une liste de plusieurs chansons avant de refaire une sélection.

#### 9.2.5.1 PartyLister

L'auteur a développé une extension au modèle que nous avons présenté dans ce mémoire (chapitre 7) lors du *Boston Music Hackday*[2] suite à la publication de l'article. Appelée *PartyLister*, cette extension, qui est encore au stade de prototype, vise à ajouter la possibilité de construire des listes d'écoutes collaboratives, c'est-à-dire en considérant les goûts musicaux de plusieurs utilisateurs en même temps.

L'idée est de modifier l'équation 8.2, qui constitue la deuxième étape de notre algorithme, pour y considérer le nuage de plusieurs utilisateurs. Nous assignons un poids $\alpha_i$ à chaque utilisateur, qui correspond à sa satisfaction par rapport à la musique qui a joué récemment. Plus nous avons joué de chansons qui correspondent à ce qu'un utilisateur veut entendre, moins il aura d'importance dans le choix de la prochaine chanson. Une version simplifiée de la nouvelle équation qui ne considère que la dernière chanson jouée dans le calcul du poids $\alpha_i$ est :

$$t^{(n+1)} = \arg\min_{t_j \in \mathscr{T}} \sum_{i=1}^{I} \left( \alpha_i \cdot cd(\mathscr{C}_{U_i}, \mathscr{C}_{t_j}) \right), \tag{9.1}$$

où $I$ est le nombre d'utilisateurs, $n$ est la position de la chanson dans la liste et où

$$\alpha_i = cd(\mathscr{C}_{U_i}, \mathscr{C}_{t^{(n)}}). \tag{9.2}$$

Nos brèves expériences ont montré qu'en considérant plusieurs chansons dans le calcul de $\alpha_i$, on se retrouve avec un générateur qui fera jouer en alternance des chansons pour toutes les personnes étant dans le système.

### 9.2.6 Source des listes d'écoute

Notre modèle utilise les listes de lecture de stations de radio commerciale étant donné qu'une grande quantité de données était disponible par l'entremise de Yes.com[97]. Puisque ce qui est joué à la radio n'est pas nécessairement une bonne référence dans tous les contextes, nous devrions aussi essayer d'utiliser des listes d'écoute provenant d'autres sources, par exemple celles créées par les utilisateurs de Last.fm, comme il est fait dans [20].

D'autres sources potentielles de listes d'écoute seraient des communautés Internet comme Imeem (récemment acheté par MySpace) ou Playlist.com.

[2] `http://boston.musichackday.org`

### 9.2.7 Apprentissage du modèle de similarité

Lors de l'entraînement du SdA tel que présenté au chapitre 8, le premier auto-encodeur débruiteur recevait en entrée les attributs de deux ou trois chansons concaténés ensemble. Un test à effectuer est de tenter d'apprendre une représentation compacte des chansons prises individuellement au lieu de toutes en même temps comme nous l'avons fait. Il s'agirait donc d'effectuer une phase de prétraitement supplémentaire pour entraîner un audo-encodeur denoising sur le vecteur de chaque chanson. Une fois une représentation compacte trouvée, on concatène ces vecteurs compacts et on utilise cette représentation pour l'entraînement.

Cela revient un peu sur l'exploration des représentations parcimonieuses des vecteurs d'attributs comme il a été mentionné à la section 9.2.4.

### 9.2.8 Recommandations personnalisables

L'auteur croit fermement que les nuages d'étiquettes personnalisables ont la puissance nécessaire pour permettre aux utilisateurs de communiquer efficacement avec les systèmes de recommandation. En supposant un espace d'étiquettes suffisamment riche, des concepts complexes peuvent être exprimés très simplement et d'une manière naturelle. Des expériences supplémentaires devraient être effectuées pour déterminer avec quel type d'interface les utilisateurs sont confortables et de quelles manières on peut faciliter leur interaction avec le système. Par exemple, on pourrait effectuer des recommandations d'étiquettes supplémentaires à ajouter à leur nuage personnalisable.

## BIBLIOGRAPHIE


[1] AHRENDT, P., AND MENG, A. Music genre classification using the multivariate AR feature integration model. Extended Abstract, 2005. MIREX genre classification contest.

[2] ANDERSON, C. *The Long Tail : Why the Future of Business Is Selling Less of More*. Hyperion, 2006.

[3] AUCOUTURIER, J., AND PACHET, F. Scaling up music playlist generation. In *Proceedings of IEEE International Conference on Multimedia and Expo (ICME)* (Lausanne, Switzerland, 2002), pp. 105–108.

[4] AUCOUTURIER, J., AND PACHET, F. Representing musical genre : A state of the art. *Journal of New Music Research 32*, 1 (2003), 83–93.

[5] BACCIGALUPO, C. *Poolcasting : an intelligent technique to customise music programmes for their audience*. PhD thesis, Universitat Autonoma de Barcelona, Barcelona, Spain, 2009.

[6] BACCIGALUPO, C., AND PLAZA, E. Case-based sequential ordering of songs for playlist recommendation. *Lecture Notes in Computer Science 4106* (2006), 286 – 300.

[7] BARRINGTON, L., ODA, R., AND LANCKRIET, G. Smarter than Genius ? Human evaluation of music recommender systems. In *Proceedings of the 10th International Conference on Music Information Retrieval (ISMIR 2009)* (Kobe, Japan, 2009).

[8] BARRINGTON, L., TURNBULL, D., TORRES, D., AND LANCKRIET, G. Semantic similarity for music retrieval, 2007. Submission to the *Audio Music Similarity and Retrieval Task* of *MIREX07*.

[9] BENGIO, Y., AND LECUN, Y. Scaling learning algorithms towards AI. In *Large-Scale Kernel Machines* (2007), MIT Press, pp. 321–360.

[10] BENNETT, J., AND LANNING, S. The netflix prize. In *Proceedings of KDD Cup and Workshop* (2007), vol. 2007.

[11] BERENZWEIG, A., LOGAN, B., ELLIS, D., AND WHITMAN, B. A large-scale evaluation of acoustic and subjective music similarity measures. In *Proceedings of the 4th International Conference on Music Information Retrieval (ISMIR 2003)* (2003).

[12] BERGSTRA, J. Algorithms for classifying recorded music by genre. Master's thesis, Université de Montréal, Canada, 2006.

[13] BERGSTRA, J., CASAGRANDE, N., AND ECK, D. Genre classification : Timbre- and rhythm-based multiresolution audio classification, 2005. Submission to the *Genre Classification Task* of *MIREX05*.

[14] BERGSTRA, J., CASAGRANDE, N., ERHAN, D., ECK, D., AND KÉGL, B. Aggregate features and AdaBoost for music classification. *Machine Learning 65*, 2-3 (2006), 473–484.

[15] BERGSTRA, J., LACOSTE, A., AND ECK, D. Predicting genre labels for artists using freedb. In *Proceedings of the 7th International Conference on Music Information Retrieval (ISMIR 2006)* (2006).

[16] BERTIN-MAHIEUX, T. Apprentissage statistique pour l'étiquetage de musique et la recommandation. Master's thesis, Université de Montréal, Canada, août 2009.

[17] BERTIN-MAHIEUX, T., ECK, D., BENGIO, Y., AND LAMERE, P. Mirex tagging contest : A boosting approach, 2008. Submission to the *Audio Tag Classification Task* of *MIREX08*.

[18] BERTIN-MAHIEUX, T., ECK, D., MAILLET, F., AND LAMERE, P. Autotagger : A model for predicting social tags from acoustic features on large music databases. *Journal of New Music Research 37*, 2 (2008), 115–135.

[19] BISHOP, C. M. *Pattern Recognition and Machine Learning (Information Science and Statistics)*, 1 ed. Springer, October 2007.

[20] BOSTEELS, K., PAMPALK, E., AND KERRE, E. Evaluating and analysing dynamic playlist generation heuristics using radio logs and fuzzy set theory. In *In Proceedings of the 10th International Conference on Music Information Retrieval (ISMIR 2009)* (2009), pp. 351–256.

[21] BRADLEY, J. K. FilterBoost : Regression and Classification on Large Datasets, 2009. Carnegie Mellon University project paper : `http://www.ml.cmu.edu/current_students/DAP_Bradley.pdf`.

[22] BRADLEY, J. K., AND SCHAPIRE, R. Filterboost : Regression and classification on large datasets. In *Advances in Neural Information Processing Systems 20*, J. Platt, D. Koller, Y. Singer, and S. Roweis, Eds. MIT Press, Cambridge, MA, 2008.

[23] BURRED, J. J., AND PEETERS, G. An adaptive system for music classification and tagging, 2009. Submission to the *Audio Tag Classification Task* of *MIREX09*.

[24] CELMA, O. *Music Recommendation and Discovery in the Long Tail*. PhD thesis, Universitat Pompeu Fabra, Barcelona, Spain, 2008.

[25] CELMA, O., AND LAMERE, P. Music Recommend Tutorial. Tutorial at *ISMIR 2007*, `http://www.iua.upf.edu/~ocelma/MusicRecommendationTutorial-ISMIR2007`, 2007.

[26] CUNNINGHAM, S. J., BAINBRIDGE, D., AND FALCONER, A. 'more of an art than a science' : Supporting the creation of playlists and mixes. In *ISMIR* (2006), pp. 240–245.

[27] DESJARDINS, G. Training Deep Convolutional Architectures for Vision. Master's thesis, Université de Montréal, Canada, août 2009.

[28] DONALDSON, J., AND LAMERE, P. Using Visualizations for Music Discovery. Tutorial at *ISMIR 2009*, `http://sites.google.com/site/musicviz2`, 2009.

[29] ECK, D., BERTIN-MAHIEUX, T., AND LAMERE, P. Autotagging music using supervised machine learning. In *Proceedings of the 8th International Conference on Music Information Retrieval (ISMIR 2007)* (2007).

[30] ECK, D., LAMERE, P., BERTIN-MAHIEUX, T., AND GREEN, S. Automatic generation of social tags for music recommendation. In *Advances in Neural Information Processing Systems 20*, J. Platt, D. Koller, Y. Singer, and S. Roweis, Eds. MIT Press, Cambridge, MA, 2008.

[31] ELLIS, D., WHITMAN, B., BERENZWEIG, A., AND LAWRENCE, S. The quest for ground truth in musical artist similarity. In *Proceedings of the 3th International Conference on Music Information Retrieval (ISMIR 2002)* (2002).

[32] FIELDS, B., AND RHODES, C. An Audience Steerable Automatic Music Director for Online Radio Broadcast, 2009. Late-breaking demo session at ISMIR09.

[33] FIELDS, B., RHODES, C., CASEY, M., AND JACOBSON, K. Social playlists and bottleneck measurements : Exploiting musician social graphs using content-based dissimilarity and pairwise maximum flow values. In *Proceedings of the 9th International Conference on Music Information Retrieval* (Philadelphia, USA, 2008), pp. 559–564.

[34] FLEXER, A., SCHNITZER, D., GASSER, M., AND WIDMER, G. Playlist generation using start and end songs. In *Proceedings of the 9th International Conference on Music Information Retrieval* (Philadelphia, USA, 2008).

[35] FREUND, Y., AND SCHAPIRE, R. E. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences 55*, 1 (1997), 119–139.

[36] GOLD, B., AND MORGAN, N. *Speech and Audio Signal Processing : Processing and Perception of Speech and Music*. John Wiley & Sons, Inc., New York, NY, USA, 1999.

[37] GOUSSEVSKAIA, O., KUHN, M., LORENZI, M., AND WATTENHOFER, R. From web to map : Exploring the world of music. In *WI-IAT '08 : Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology* (Washington, DC, USA, 2008), IEEE Computer Society, pp. 242–248.

[38] GREEN, S. J., LAMERE, P., ALEXANDER, J., MAILLET, F., KIRK, S., HOLT, J., BOURQUE, J., AND MAK, X.-W. Generating transparent, steerable recommendations from textual descriptions of items. In *RecSys '09 : Proceedings of the third ACM conference on Recommender systems* (New York, NY, USA, 2009), ACM, pp. 281–284.

[39] GUY, M., AND TONKIN, E. Tidying up tags. *D-Lib Magazine* (2006). online article : www.dlib.org/dlib/january06/guy/01guy.html.

[40] HERLOCKER, J. L., KONSTAN, J. A., AND RIEDL, J. T. Explaining collaborative filtering recommendations. In *Computer Supported Cooperative Work* (2000), pp. 241–250.

[41] HERLOCKER, J. L., KONSTAN, J. A., TERVEEN, L. G., AND RIEDL, J. T. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst. 22*, 1 (2004), 5–53.

[42] HINTON, G., OSINDERO, S., AND TEH, Y. A fast learning algorithm for deep belief nets. *Neural Computation 18* (2006), 1527–1554.

[43] HJORTH-ANDERSEN, C. Chris anderson, the long tail : How endless choice is creating unlimited demand. the new economics of culture and commerce. *Journal of Cultural Economics 31*, 3 (September 2007), 235–237.

[44] KOREN, Y. Collaborative filtering with temporal dynamics. In *KDD '09 : Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining* (2009), pp. 447–456.

[45] KÉGL, B., BERTIN-MAHIEUX, T., AND ECK, D. Metropolis-hastings sampling in a filter-boost music classifier. In *International Workshop on Machine Learning and Music (MML 2008)* (2008).

[46] LAMBROU, T., KUDUMAKIS, P., SPELLER, R., SANDLER, M., , AND LINNEY, A. Classification of audio signals using statistical features on time and wavelet tranform domains. In *Proc. Int. Conf. Acoustic, Speech, and Signal Processing (ICASSP-98)* (1998), vol. 6, pp. 3621–3624.

[47] LAMERE, P. Last.fm artist tags 2007 data set. `http://tinyurl.com/6ry8ph`.

[48] LAMERE, P. Social tagging and music information retrieval. *Journal of New Music Research 37*, 2 (2008), 101–114.

[49] LAMERE, P. Help ! My iPod thinks I'm emo. Panel at *South by Southwest (SXSW) 2009*, `http://musicmachinery.com/2009/03/26/help-my-ipod-thinks-im-emo-part-1`, 2009.

[50] LAROCHELLE, H. *Étude de techniques d'apprentissage non-supervisé pour l'amélioration de l'entraînement supervisé de modèles connexionnistes*. PhD thesis, University of Montréal, Mar. 2009.

[51] LAST.FM. API. Web Services described at `http://www.last.fm/api`.

[52] LAST.FM. Multi tag search. `http://playground.last.fm/multitag`.

[53] LAST.FM LTD. Last.fm. `http://www.last.fm`.

[54] LAW, E. The problem of accuracy as an evaluation criterion, 2008. ICML Workshop on Evaluation Methods in Machine Learning.

[55] LAW, E., V. AHN, A., DANNENBERG, R., AND CRAWFORD, M. Tagatune : a game for music and sound annotation. In *Proceedings of the 8th International Conference on Music Information Retrieval (ISMIR 2007)* (2007).

[56] LAW, E., AND VON AHN, L. Input-agreement : a new mechanism for collecting data using human computation games. In *CHI '09 : Proceedings of the 27th international conference on Human factors in computing systems* (New York, NY, USA, 2009), ACM, pp. 1197–1206.

[57] LAW, E., WEST, K., MANDEL, M., BAY, M., AND DOWNIE, J. S. Evaluation of algorithms using games : The case of music tagging. In *Proceedings of the 10th International Conference on Music Information Retrieval (ISMIR 2009)* (Kobe, Japan, 2009).

[58] LI, T., OGIHARA, M., AND LI, Q. A comparative study on content-based music genre classification. In *SIGIR '03 : Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval* (New York, NY, USA, 2003), ACM Press, pp. 282–289.

[59] LO, H.-Y., WANG, J.-C., AND WANG, H.-M. An ensemble method for mirex audio tag classification, 2009. Submission to the *Audio Tag Classification Task* of *MIREX09*.

[60] LOGAN, B. Content-based playlist generation : Exploratory experiments. In *Proceedings of the 3rd International Conference on Music Information Retrieval* (Paris, France, October 2002), M. Fingerhut, Ed., Ircam - Centre Pompidou.

[61] LOGAN, B. Music recommendation from song sets. In *Proceedings of the 5th International Conference on Music Information Retrieval* (Barcelona, Spain, 2004).

[62] LOGAN, B., AND SALOMON, A. A music similarity function based on signal analysis. In *2001 IEEE International Conference on Multimedia and Expo (ICME'01)* (2001), p. 190.

[63] MANDEL, M. blog at `http://blog.mr-pc.org/2008/03/04/autotagging`.

[64] MANDEL, M., AND ELLIS, D. Song-level features and support vector machines for music classification. In *Proceedings of the 6th International Conference on Music Information Retrieval (ISMIR 2005)* (2005), T. Crawford and M. Sandler, Eds.

[65] MANDEL, M., AND ELLIS, D. A web-based game for collecting music metadata. In *Proceedings of the 8th International Conference on Music Information Retrieval (ISMIR 2007)* (2007).

[66] MANDEL, M., AND ELLIS, D. A Web-Based Game for Collecting Music Metadata. *Journal of New Music Research 37*, 2 (2008), 151–165.

[67] MANDEL, M. I., AND ELLIS, D. P. W. Labrosa's audio classification submissions, 2008. Submission to the *Audio Tag Classification Task* of *MIREX08*.

[68] MANZAGOL, P.-A., BERTIN-MAHIEUX, T., AND ECK, D. On the use of sparse time relative auditory codes for music. In *Proceedings of the 9th International Conference on Music Information Retrieval (ISMIR 2008)* (Philadelphia, USA, 2008).

[69] MCKAY, C., AND FUJINAGA, I. Musical genre classification : is it worth pursuing and how can it be. In *Proceedings of the 7th International Conference on Music Information Retrieval (ISMIR 2006)* (2006).

[70] NESS, S. R., THEOCHARIS, A., TZANETAKIS, G., AND MARTINS, L. G. Improving automatic music tag annotation using stacked generalization of probabilistic svm outputs. In *MM '09 : Proceedings of the seventeen ACM international conference on Multimedia* (New York, NY, USA, 2009), ACM, pp. 705–708.

[71] PAMPALK, E. *Computational Models of Music Similarity and their Application in Music Information Retrieval*. PhD thesis, Vienna University of Technology, Vienna, Austria, March 2006.

[72] PAMPALK, E., POHLE, T., AND WIDMER, G. Dynamic playlist generation based on skipping behavior. In *Proceedings of 6th International Conference on Music Information Retrieval* (London, UK, 2005).

[73] PITCHFORK MEDIA INC. Pitchfork Magazine. `http://pitchfork.com`.

[74] RAGNO, R., BURGES, C. J. C., AND HERLEY, C. Inferring similarity between music objects with application to playlist generation. In *MIR '05 : Proceedings of the 7th ACM SIGMM international workshop on Multimedia information retrieval* (New York, NY, USA, 2005), ACM, pp. 73–80.

[75] ROBERT, D. L., Ed. *Le petit Robert 1*. Les Dictionnaires ROBERT-CANADA S.C.C., Montréal, Canada, 1991.

[76] SCHAFER, J., FRANKOWSKI, D., HERLOCKER, J., AND SEN, S. Collaborative filtering recommender systems. *Lecture Notes In Computer Science 4321* (2007), 291–324.

[77] SCHAPIRE, R. E., AND SINGER, Y. Improved boosting algorithms using confidence-rated predictions. *Machine Learning 37*, 3 (December 1999), 297–336.

[78] SHEPITSEN, A., GEMMELL, J., MOBASHER, B., AND BURKE, R. Personalized recommendation in social tagging systems using hierarchical clustering. In *RecSys '08 : Proceedings of the 2008 ACM conference on Recommender systems* (New York, NY, USA, 2008), ACM, pp. 259–266.

[79] STREICH, S. *Music Complexity a multi-faceted description of audio content*. PhD thesis, Ph.D. Dissertation. UPF, 2007.

[80] SUN MICROSYSTEMS LABORATORIES. The Music Explaura. `http://music.tastekeeper.com`.

[81] THE INTERNATIONAL MUSIC INFORMATION RETRIEVAL SYSTEMS EVALUATION LABORATORY. MIREX : Music Information Retrieval Evaluation eXchange, information at `http://www.music-ir.org/mirexwiki/index.php`.

[82] TINTAREV, N., AND MASTHOFF, J. A survey of explanations in recommender systems. In *ICDEW '07 : Proceedings of the 2007 IEEE 23rd International Conference on Data Engineering Workshop* (Washington, DC, USA, 2007), IEEE Computer Society, pp. 801–810.

[83] TORRES, D., TURNBULL, D., BARRINGTON, L., AND LANCKRIET, G. Identifying words that are musically meaningful. In *Proceedings of the 8th International Conference on Music Information Retrieval (ISMIR 2007)* (2007).

[84] TURNBULL, D., BARRINGTON, L., AND LANCKRIET, G. Five approaches to collecting tags for music. In *In Proceedings of the 9th International Conference on Music Information Retrieval (ISMIR 2008)* (2008), pp. 225–230.

[85] TURNBULL, D., BARRINGTON, L., TORRES, D., AND LANCKRIET, G. Towards musical query-by-semantic-description using the cal500 data set. In *SIGIR '07 : Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval* (New York, NY, USA, 2007), ACM, pp. 439–446.

[86] TURNBULL, D., BARRINGTON, L., TORRES, D., AND LANCKRIET, G. Semantic annotation and retrieval of music and sound effects. *IEEE Transactions on Audio, Speech & Language Processing 16*, 2 (2008).

[87] TURNBULL, D., LANCKRIET, G., PAMPALK, E., AND GOTO, M. A supervised approach for detecting boundaries in music using difference features and boosting. In *Proceedings of the 8th International Conference on Music Information Retrieval (ISMIR 2007)* (2007).

[88] TURNBULL, D., LIU, R., BARRINGTON, L., AND LANCKRIET, G. A game-based approach for collecting semantic annotations of music. In *Proceedings of the 8th International Conference on Music Information Retrieval (ISMIR 2007)* (2007).

[89] TZANETAKIS, G., AND COOK, P. Musical genre classification of audio signals. *IEEE Transactions on Speech and Audio Processing 10*, 5 (Jul 2002), 293–302.

[90] VAN DER MAATEN, L., AND HINTON, G. Visualizing data using t-sne. *Journal of Machine Learning Research 9* (November 2008), 2579–2605.

[91] VICKERS, E. Automatic long-term loudness and dynamics matching. In *In Proceedings of the AES 111th Convention* (2001).

[92] VIG, J., SEN, S., AND RIEDL, J. Tagsplanations : explaining recommendations using tags. In *IUI '09 : Proceedings of the 13th international conference on Intelligent user interfaces* (New York, NY, USA, 2009), ACM, pp. 47–56.

[93] VINCENT, P., LAROCHELLE, H., BENGIO, Y., AND MANZAGOL, P.-A. Extracting and composing robust features with denoising autoencoders. In *ICML '08 : Proceedings of the 25th international conference on Machine learning* (New York, NY, USA, 2008), ACM, pp. 1096–1103.

[94] VON AHN, L., AND DABBISH, L. 2004, labeling images with a computer game. In *CHI '04 : Proceedings of the SIGCHI conference on Human factors in computing systems* (New York, NY, USA, 2004), ACM Press, pp. 319–326.

[95] WEST, K., AND COX, S. Features and classifiers for the automatic classification of musical audio signals. In *ISMIR 2004, 5th International Conference on Music Information Retrieval* (Barcelona, Spain, 2004).

[96] WETZKER, R., UMBRATH, W., AND SAID, A. A hybrid approach to item recommendation in folksonomies. In *ESAIR '09 : Proceedings of the WSDM '09 Workshop on Exploiting Semantic Annotations in Information Retrieval* (New York, NY, USA, 2009), ACM, pp. 25–29.

[97] YES.COM API. Web Services described at `http://api.yes.com`.

[98] ZANARDI, V., AND CAPRA, L. Social ranking : uncovering relevant content using tag-based recommender systems. In *RecSys '08 : Proceedings of the 2008 ACM conference on Recommender systems* (New York, NY, USA, 2008), ACM, pp. 51–58.

## Annexe I

### The 360 Last.fm tags used in our experiments

Tableau I.1 – The 360 Last.fm tags used in our experiments

| | | | |
|---|---|---|---|
| 00s | drum n bass | humour | progressive rock |
| 60s | dub | icelandic | progressive trance |
| 70s | dutch | idm | prog rock |
| 80s | duyster | idols | proto-punk |
| 90s | east coast | indie | psychedelic |
| acid jazz | easy listening | indie folk | psychedelic rock |
| acoustic | ebm | indie pop | psychobilly |
| acoustic rock | electro | indie rock | psytrance |
| africa | electroclash | indietronic | punk |
| alt-country | electro industrial | indietronica | punk-pop |
| alternative | electronic | industrial | punk rock |
| alternative country | electronica | industrial metal | quirky |
| alternative hip-hop | electropop | industrial rock | r and b |
| alternative metal | elephant 6 | instrumental | rap |
| alternative rock | emo | instrumental rock | rapcore |
| alt rock | emocore | irish | rasta |
| amazing | emusic | italian | reggae |
| ambient | england | jam | relax |
| american | english | jam band | relaxing |
| americana | epic metal | japan | rhythm and blues |
| anarcho-punk | ethereal | japanese | riot grrrl |
| anime | eurodance | japanese music | rnb |

Continued on next page

Continued from previous page

| | | | |
|---|---|---|---|
| anti-christian | experimental | japanese rock | rock |
| art rock | experimental rock | jazz | rockabilly |
| asian | fantasy | jazz fusion | rock and roll |
| atmospheric | fav | jazz piano | rock n roll |
| aussie | favorite | jazz vocal | romantic |
| australian | favorite artists | j-pop | roots |
| avant-garde | favorite bands | j-rock | roots reggae |
| avant-garde metal | favorites | kill rock stars | russian |
| awesome | favourite | krautrock | sad |
| band | favourite artists | latin | saddle creek |
| baroque | favourite bands | left-wing | saxophone |
| beats | favourites | lesser known yet streamable artists | scandinavian |
| beautiful | female | lo-fi | scottish |
| belgian | female artists | lounge | screamo |
| belgium | female fronted metal | love | seattle |
| big beat | female vocalist | love metal | seen live |
| bittersweet | female vocalists | male | sexy |
| black metal | female vocals | male vocalists | sh*t |
| black music | female voices | math rock | shoegaze |
| bluegrass | finland | medieval | singer-songwriter |
| blues | finnish | meditation | ska |
| blues rock | finnish metal | melancholic | ska punk |
| bossa nova | folk | melancholy | slowcore |
| brand new | folk metal | mellow | sludge |
| brasil | folk-punk | melodic black metal | soft rock |
| brazil | folk rock | melodic death metal | soul |
| brazilian | francais | melodic hardcore | soundtrack |

Continued on next page

Continued from previous page

| | | | |
|---|---|---|---|
| breakbeat | france | melodic metal | soundtracks |
| breakcore | freak folk | metal | southern rock |
| british | free jazz | metalcore | space rock |
| britpop | french | mike patton | spanish |
| britrock | french rock | minimal | speed metal |
| brutal death metal | fun | minimal techno | stoner |
| california | funk | motown | stoner rock |
| canada | funk rock | mpb | straight edge |
| canadian | funky | music | street punk |
| celtic | funny | my music | surf |
| chanson | fusion | neoclassical | sweden |
| chanson francaise | futurepop | neofolk | swedish |
| check out | gangsta rap | neo-soul | swedish metal |
| chill | garage | new age | swing |
| chillout | garage rock | new romantic | symphonic black metal |
| christian | gay | new wave | symphonic metal |
| christian rock | genius | new weird america | synth |
| classic | gentle | new york | synthpop |
| classical | german | ninja tune | tango |
| classic rock | germany | noise | technical death metal |
| club | girl group | noise rock | techno |
| cold wave | glam | norway | the beatles |
| comedy | glam rock | norwegian | the good stuff |
| composer | glasgow | nu-jazz | the worst thing ever to happen to music |
| composers | glitch | nu metal | thrash |
| contemporary classical | goa | nwobhm | thrash metal |
| cool | good | officially sh*t | traditional |

Continued on next page

Continued from previous page

| | | | |
|---|---|---|---|
| country | good music | oi | trance |
| crap | goth | oldies | trip-hop |
| crossover | gothic | old school | trumpet |
| dance | gothic metal | ost | turntablism |
| dancehall | gothic rock | pagan metal | twee |
| danish | goth rock | piano | uk |
| dark | great lyricists | piano rock | underground hip hop |
| dark ambient | grind | polish | underground rap |
| dark electro | grindcore | political | uplifting trance |
| dark metal | grunge | polskie | urban |
| darkwave | guilty pleasures | pop | us |
| death | guitar | pop punk | usa |
| death metal | guitar virtuoso | pop rock | video game music |
| deathrock | hair metal | post-grunge | viking metal |
| deutsch | happy | post-hardcore | visual kei |
| disco | hardcore | post-punk | vocal |
| dnb | hardcore punk | post-rock | vocal jazz |
| doom | hard rock | power metal | vocal trance |
| doom metal | heavy | power pop | warp |
| downtempo | heavy metal | prog | world |
| dream pop | hip-hop | progressive | world music |
| drone | horror punk | progressive death metal | wristslitters |
| drum and bass | house | progressive metal | |

# Annexe II

## Category-level results for CAL500 experiments

| Category | $A/\|V\|$ | Model | Precision | | Recall | | F-Score |
|---|---|---|---|---|---|---|---|
| Emotion | 4/36 | Random | 0.276 | (0.012) | 0.113 | (0.004) | 0.160 |
| | | UpperBnd | 0.957 | (0.005) | 0.396 | (0.010) | 0.560 |
| | | GMM | 0.424 | (0.008) | **0.195** | (0.004) | 0.267 |
| | | MFCC delta | 0.444 | (0.025) | 0.192 | (0.016) | **0.268** |
| | | afeats | 0.433 | (0.03) | 0.171 | (0.011) | 0.245 |
| | | afeats exp. | **0.449** | (0.026) | 0.176 | (0.011) | 0.253 |
| | | bfeats | 0.418 | (0.053) | 0.156 | (0.037) | 0.227 |
| Genre | 2/31 | Random | 0.055 | (0.005) | 0.079 | (0.008) | 0.065 |
| | | UpperBnd | 0.562 | (0.026) | 0.777 | (0.018) | 0.652 |
| | | GMM | 0.171 | (0.009) | **0.242** | (0.019) | 0.200 |
| | | MFCC delta | 0.154 | (0.024) | 0.168 | (0.021) | 0.161 |
| | | afeats | 0.173 | (0.048) | 0.134 | (0.033) | 0.151 |
| | | afeats exp. | **0.236** | (0.047) | 0.234 | (0.016) | **0.235** |
| | | bfeats | 0.147 | (0.027) | 0.160 | (0.045) | 0.153 |
| Instrumentation | 4/24 | Random | 0.141 | (0.009) | 0.195 | (0.014) | 0.164 |
| | | UpperBnd | 0.601 | (0.015) | 0.868 | (0.018) | 0.710 |
| | | GMM | 0.267 | (0.008) | 0.320 | (0.022) | 0.291 |
| | | MFCC delta | 0.267 | (0.047) | **0.363** | (0.021) | 0.308 |
| | | afeats | 0.294 | (0.073) | 0.275 | (0.074) | 0.284 |
| | | afeats exp. | 0.276 | (0.044) | 0.350 | (0.033) | **0.309** |
| | | bfeats | **0.329** | (0.065) | 0.289 | (0.084) | 0.308 |
| Solo | 1/9 | Random | 0.031 | (0.007) | 0.155 | (0.035) | 0.052 |
| | | UpperBnd | 0.197 | (0.019) | 0.760 | (0.052) | 0.313 |
| | | GMM | **0.060** | (0.012) | 0.261 | (0.050) | **0.098** |
| | | MFCC delta | 0.054 | (0.002) | 0.374 | (0.035) | 0.094 |
| | | afeats | 0.045 | (0.002) | 0.278 | (0.078) | 0.078 |
| | | afeats exp. | 0.056 | (0.001) | **0.396** | (0.017) | **0.098** |
| | | bfeats | 0.042 | (0.002) | 0.312 | (0.094) | 0.074 |
| Usage | 2/15 | Random | 0.073 | (0.008) | 0.154 | (0.016) | 0.099 |
| | | UpperBnd | 0.363 | (0.014) | 0.814 | (0.031) | 0.502 |
| | | GMM | **0.122** | (0.012) | **0.264** | (0.027) | **0.167** |
| | | MFCC delta | **0.122** | (0.011) | 0.239 | (0.028) | 0.162 |
| | | afeats | 0.103 | (0.010) | 0.188 | (0.054) | 0.133 |
| | | afeats exp. | 0.118 | (0.007) | 0.237 | (0.015) | 0.157 |
| | | bfeats | 0.106 | (0.010) | 0.209 | (0.066) | 0.140 |
| Vocal | 2/16 | Random | 0.062 | (0.007) | 0.153 | (0.018) | 0.088 |
| | | UpperBnd | 0.321 | (0.017) | 0.788 | (0.019) | 0.456 |
| | | GMM | **0.134** | (0.005) | **0.335** | (0.021) | **0.191** |
| | | MFCC delta | 0.116 | (0.011) | 0.252 | (0.029) | 0.159 |
| | | afeats | 0.130 | (0.030) | 0.198 | (0.050) | 0.157 |
| | | afeats exp. | 0.108 | (0.009) | 0.228 | (0.019) | 0.147 |
| | | bfeats | 0.133 | (0.017) | 0.212 | (0.046) | 0.164 |

Tableau II.1 – Music annotation results for specific categories. $A$ = Annotation length, $|V|$ = Vocabulary size. Numbers in parentheses are the variance over all tags in the category. GMM is the results of the Gaussian mixture model of the CAL group. MFCC delta, afeats, afeats extended and bfeat are the results of our boosting model with each of these features. Random, UpperBnd and GMM results taken from [86]. This is a continuation of 6.7.

xxx

| Category | $\|V\|$ | Model | MeanAP | | MeanAROC | |
|---|---|---|---|---|---|---|
| Emotion | 36 | Random | 0.327 | (0.006) | 0.504 | (0.003) |
| | | GMM | 0.506 | (0.008) | **0.710** | (0.004) |
| | | MFCC delta | 0.503 | (0.031) | 0.702 | (0.005) |
| | | afeats | 0.469 | (0.026) | 0.652 | (0.005) |
| | | afeats exp. | 0.478 | (0.023) | 0.655 | (0.006) |
| | | bfeats | **0.565** | (0.048) | 0.686 | (0.006) |
| Genre | 31 | Random | 0.132 | (0.005) | 0.500 | (0.005) |
| | | GMM | **0.329** | (0.012) | 0.719 | (0.005) |
| | | MFCC delta | 0.094 | (0.013) | 0.705 | (0.013) |
| | | afeats | 0.088 | (0.011) | 0.626 | (0.010) |
| | | afeats exp. | 0.117 | (0.036) | **0.720** | (0.011) |
| | | bfeats | 0.118 | (0.032) | 0.693 | (0.015) |
| Instrumentation | 24 | Random | 0.221 | (0.007) | 0.502 | (0.004) |
| | | GMM | **0.399** | (0.018) | 0.719 | (0.006) |
| | | MFCC delta | 0.137 | (0.022) | 0.707 | (0.005) |
| | | afeats | 0.182 | (0.033) | 0.658 | (0.015) |
| | | afeats exp. | 0.173 | (0.03) | 0.705 | (0.006) |
| | | bfeats | 0.140 | (0.032) | **0.720** | (0.015) |
| Solo | 9 | Random | 0.106 | (0.014) | 0.502 | (0.004) |
| | | GMM | **0.180** | (0.028) | **0.712** | (0.006) |
| | | MFCC delta | 0.052 | (0.002) | 0.565 | (0.025) |
| | | afeats | 0.050 | (0.002) | 0.582 | (0.009) |
| | | afeats exp. | 0.051 | (0.002) | 0.650 | (0.010) |
| | | bfeats | 0.042 | (0.003) | 0.519 | (0.026) |
| Usage | 15 | Random | 0.169 | (0.012) | 0.501 | (0.005) |
| | | GMM | **0.240** | (0.016) | **0.707** | (0.004) |
| | | MFCC delta | 0.12 | (0.009) | 0.621 | (0.022) |
| | | afeats | 0.133 | (0.022) | 0.538 | (0.016) |
| | | afeats exp. | 0.127 | (0.007) | 0.637 | (0.008) |
| | | bfeats | 0.101 | (0.014) | 0.563 | (0.034) |
| Vocal | 16 | Random | 0.137 | (0.006) | 0.502 | (0.004) |
| | | GMM | **0.260** | (0.018) | **0.705** | (0.005) |
| | | MFCC delta | 0.111 | (0.012) | 0.652 | (0.018) |
| | | afeats | 0.090 | (0.007) | 0.586 | (0.014) |
| | | afeats exp. | 0.105 | (0.012) | 0.628 | (0.009) |
| | | bfeats | 0.128 | (0.018) | 0.630 | (0.020) |

Tableau II.2 – Retrieval results. $|V|$ = Vocabulary size. Numbers in parentheses are the variance over all tags in the category. GMM is the results of the Gaussian mixture model of the CAL group. MFCC delta, afeats, afeats extended and bfeat are the results of our boosting model with each of these features. Random and GMM taken from [86]. This is a continuation of 6.8.