SY19 - Facial Recognition

Jo COLINA & Paul GOUJON

December 2016

Chapitre 1

Introduction

Introduction Bonjour

Objectifs du TP ça ça et ça

Code Notre code source s'organise comme suit : blabla

Chapitre 2

Analyses préliminaire des données

Introduction Nous sommes en présence d'une matrice de taille 216 x 4200 d'individus et d'un vecteur de taille 216 d'étiquettes. Chaque ligne de la matrice est en réalité une photo de taille 60 x 70 pixels en niveaux de gris d'un visage exprimmant une émotion parmis les six suivantes :

- joie
- surprise
- tristesse
- dégout
- colère
- peur

2.1 Analyse initiale

Dimensions La première chose qui nous frappe lorsque l'on est confronté à ce dataset est le ratio $p >> n$: nous sommes en présence de beaucoup plus de prédicteurs que d'individus, ce qui nous laisse penser que nous serons amenés à utiliser des méthodes de réduction de dimensions afin de contrer la fameuse "curse of dimensionality".

Pré-traitement En affichant les différentes images, on se rend également compte qu'une partie de chacune des images est totalement inutile dans le cadre de nos analyses, voire risquerait de la bruiter : il s'agit de tous les pixels noir en bas de chaque image. Nous décidons de les exclure de toute future analyse.

2.2 Répartition moyenne par expression

Répartition moyenne & variance Ensuite, afin de voir si les différentes expressions produisent des "vecteur-image moyens" significativement différents, nous décidons de ploter la réparition moyenne des prédicteurs pour chaque expression. Nous obtenons les courbes suivantes.

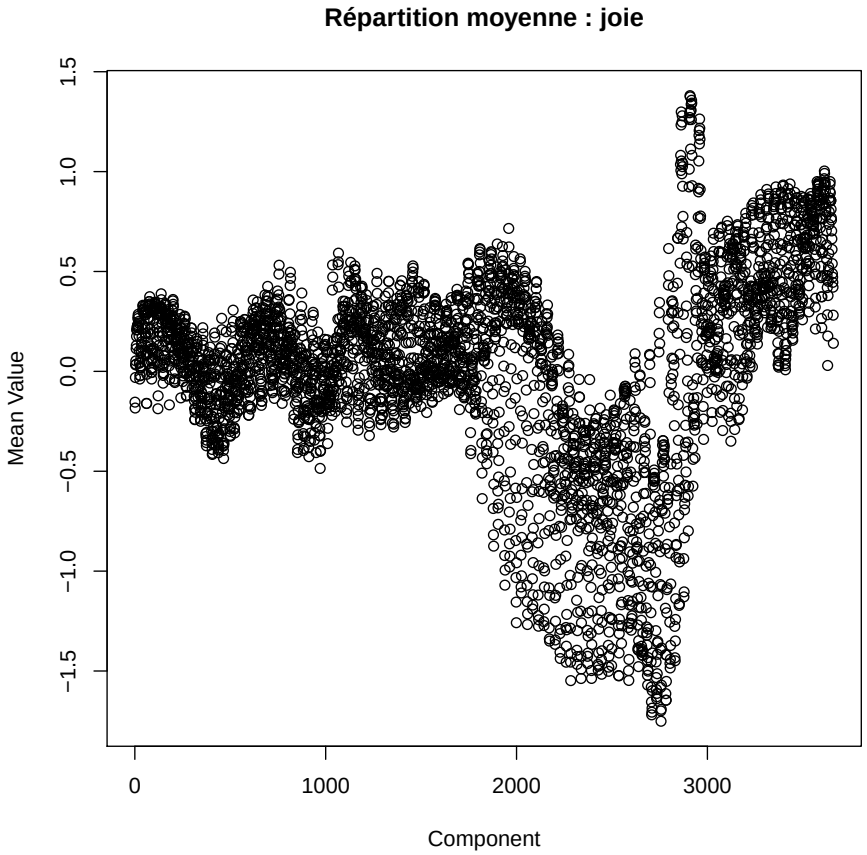

FIGURE 2.1 – Répartition moyenne des prédicteurs : joie

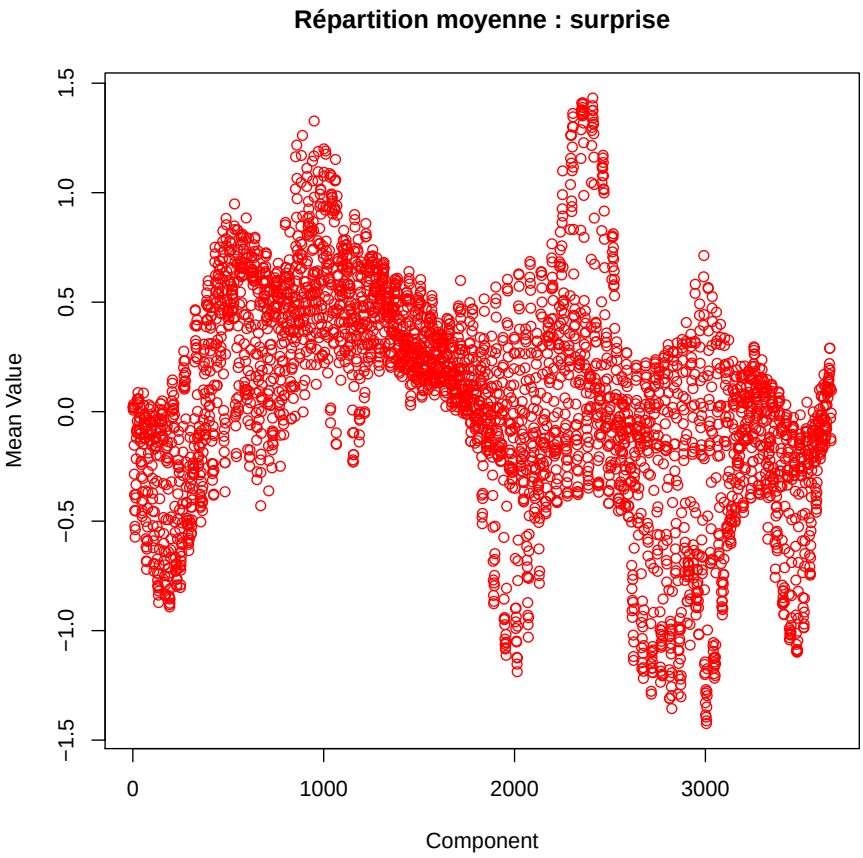

FIGURE 2.2 – Répartition moyenne des prédicteurs : surprise

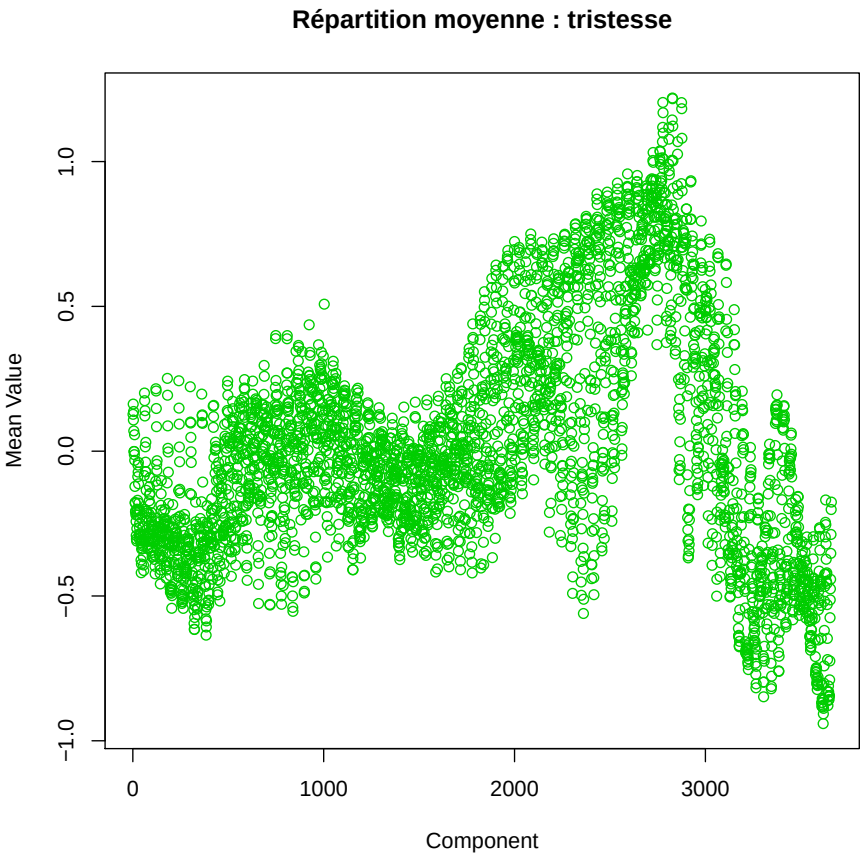

FIGURE 2.3 – Répartition moyenne des prédicteurs : tristesse

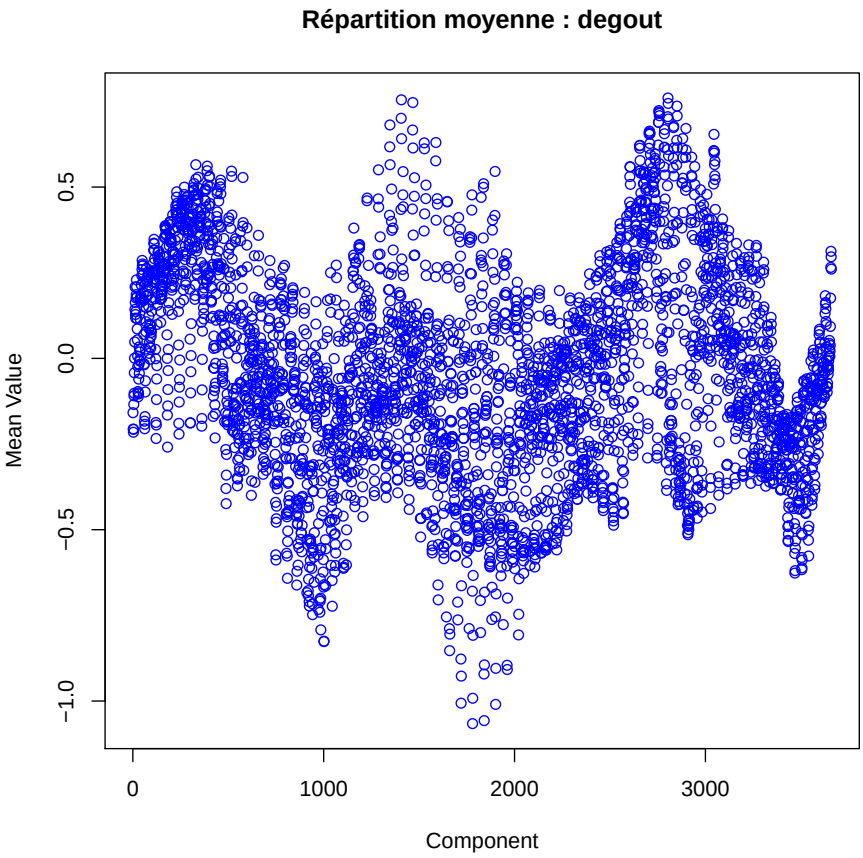

FIGURE 2.4 – Répartition moyenne des prédicteurs : degoût

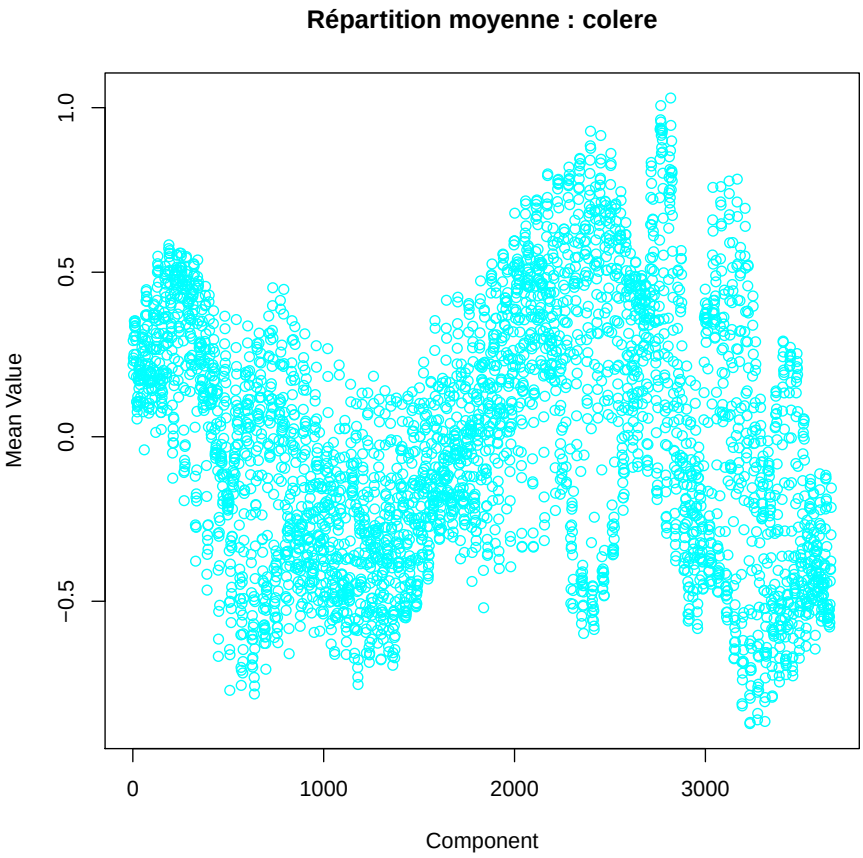

FIGURE 2.5 – Répartition moyenne des prédicteurs : colère

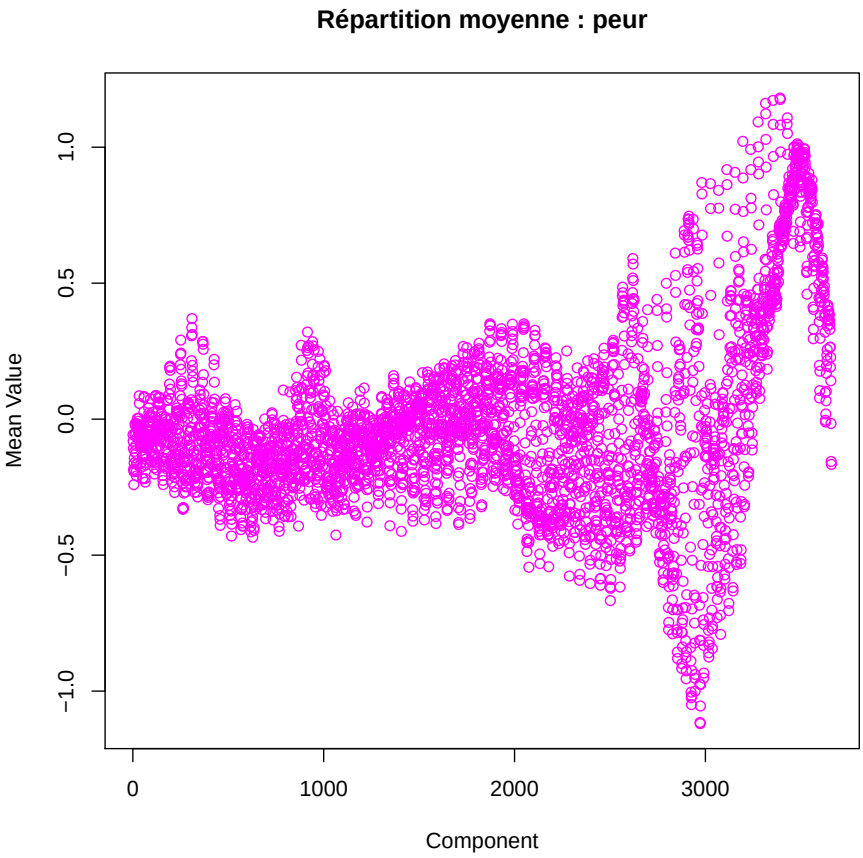

FIGURE 2.6 – Répartition moyenne des prédicteurs : peur

NB : Standardized La matrice des individus est centrée et réduite une fois effectué le pré-traitement décrit précédemment. C'est le cas pour les représentations ci-dessus.

Interprétation Les répartitions obtenue semblent différer significativement selon l'expression. Nous pouvons donc envisager que l'usage de modèles de prédiction nous permettrait de distinguer les différentes expressions.

Variance CHECKER LA VARIANCE TOUT DE MEME CAR CA POURRAIT ETRE TOUT SIMPLEMENT DU A CA

2.3 Expression moyenne

Curieux Nous avons tenté d'effectuer la moyenne pour chaque expression des images : c'est à dire produire six images issues de la moyenne mathématique des images de l'expression considérée. Ci-dessous sont les "visages moyens" issus de cette moyenne. Nous avons trouvé extrêmement curieux l'aspect humain que prend une "moyenne de visages".

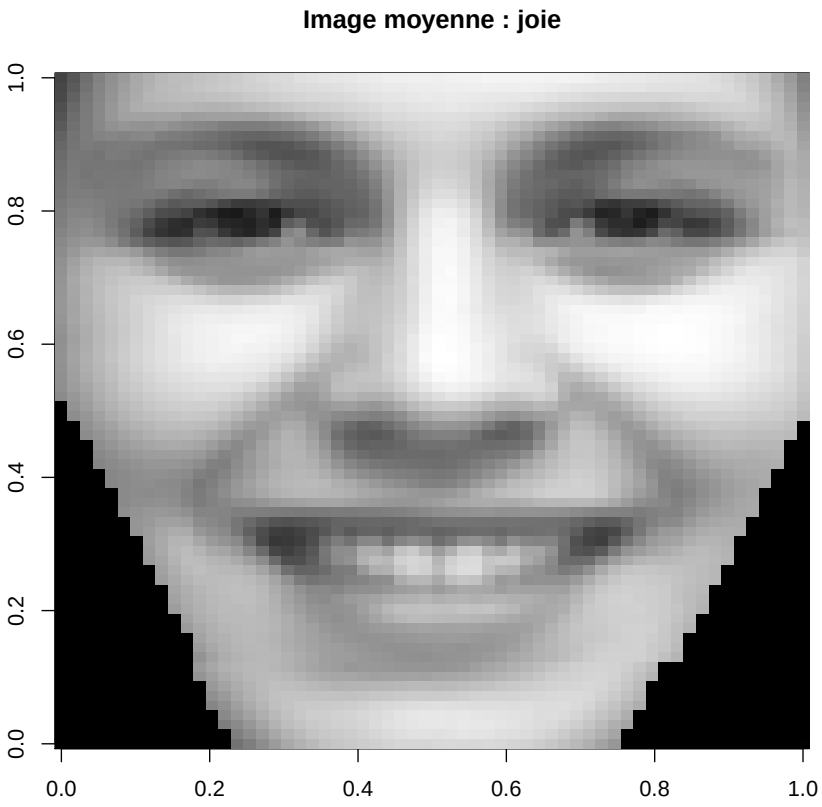

FIGURE 2.7 – Visage moyen : joie

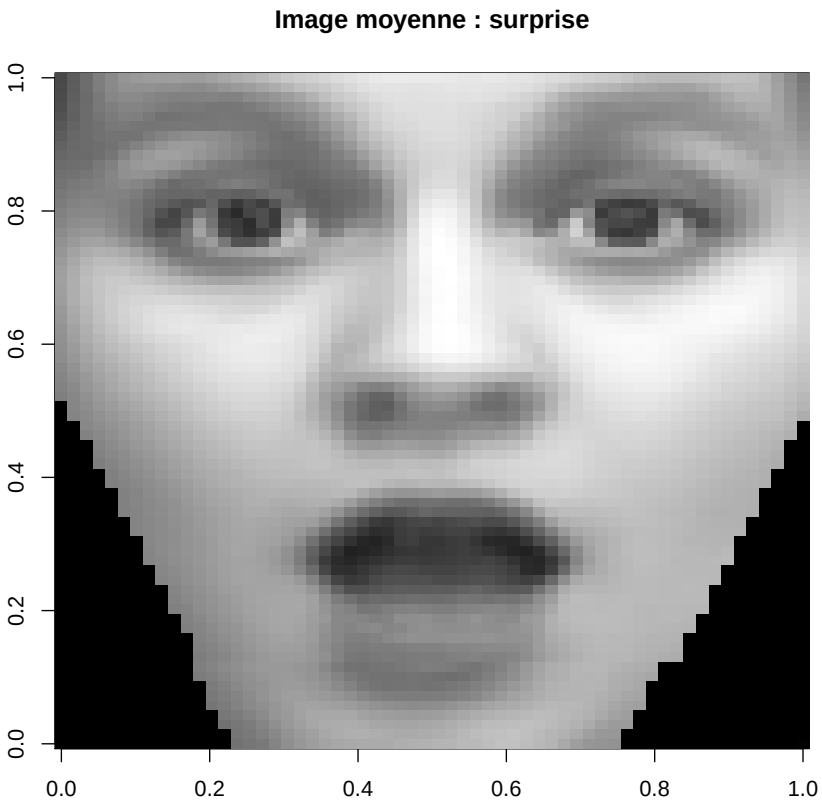

FIGURE 2.8 – Visage moyen : surprise

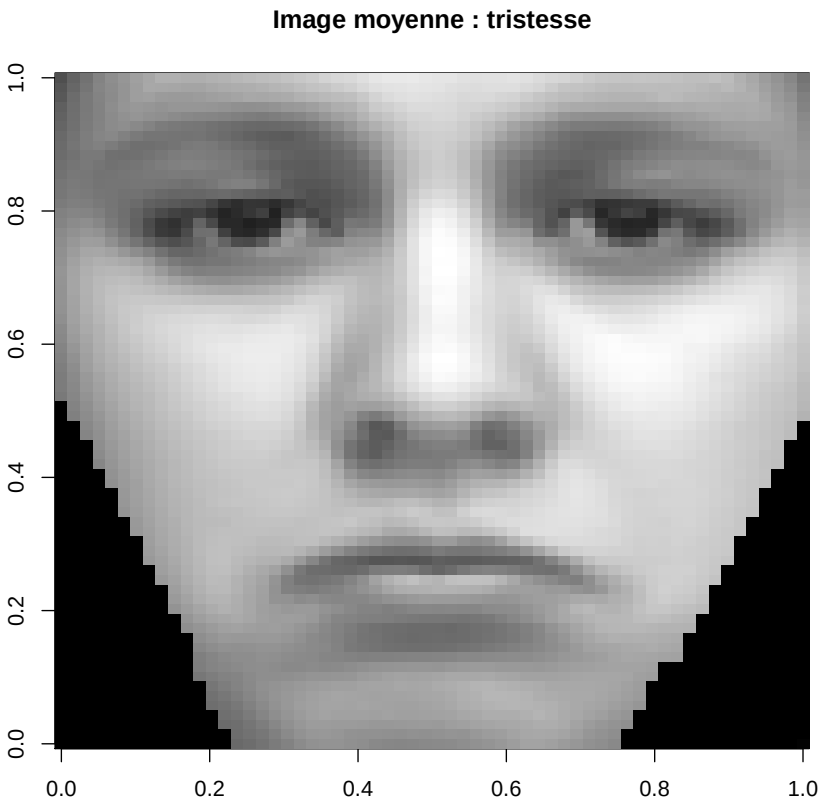

FIGURE 2.9 – Visage moyen : tristesse

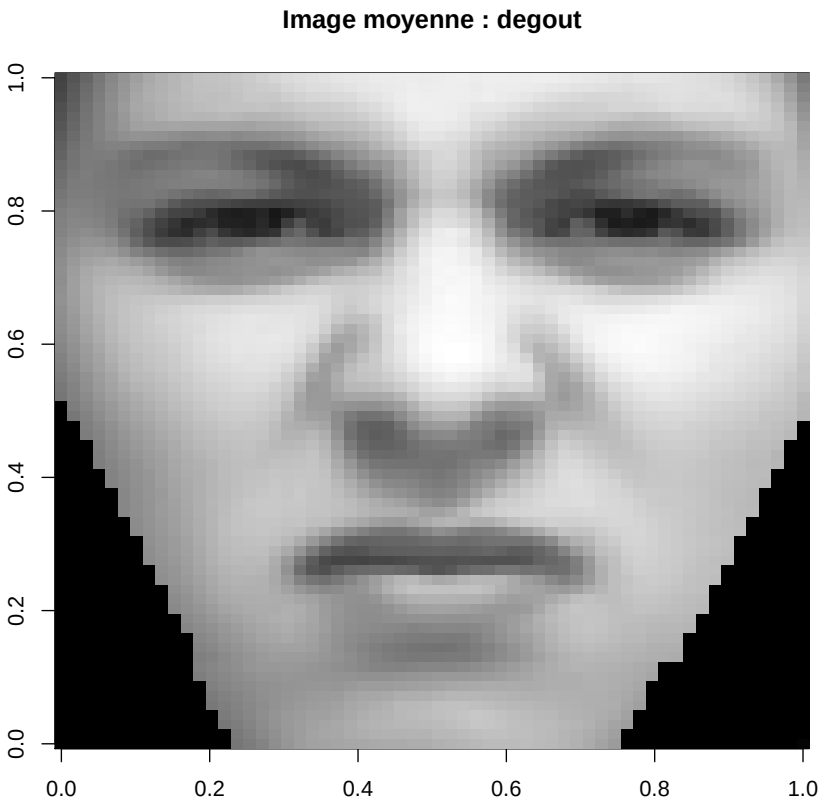

FIGURE 2.10 – Visage moyen : degoût

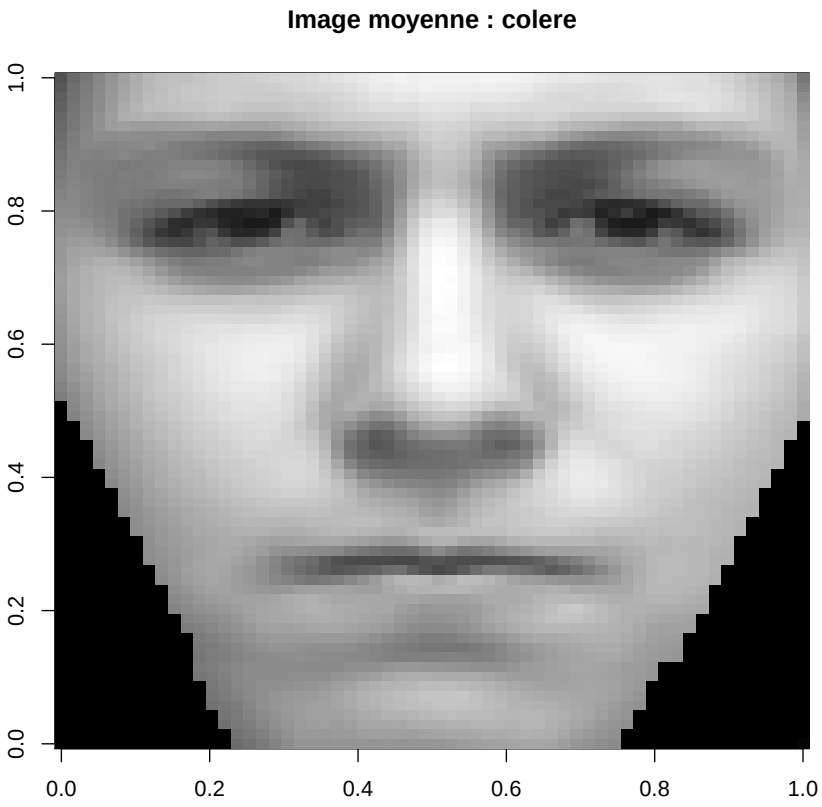

FIGURE 2.11 – Visage moyen : colère

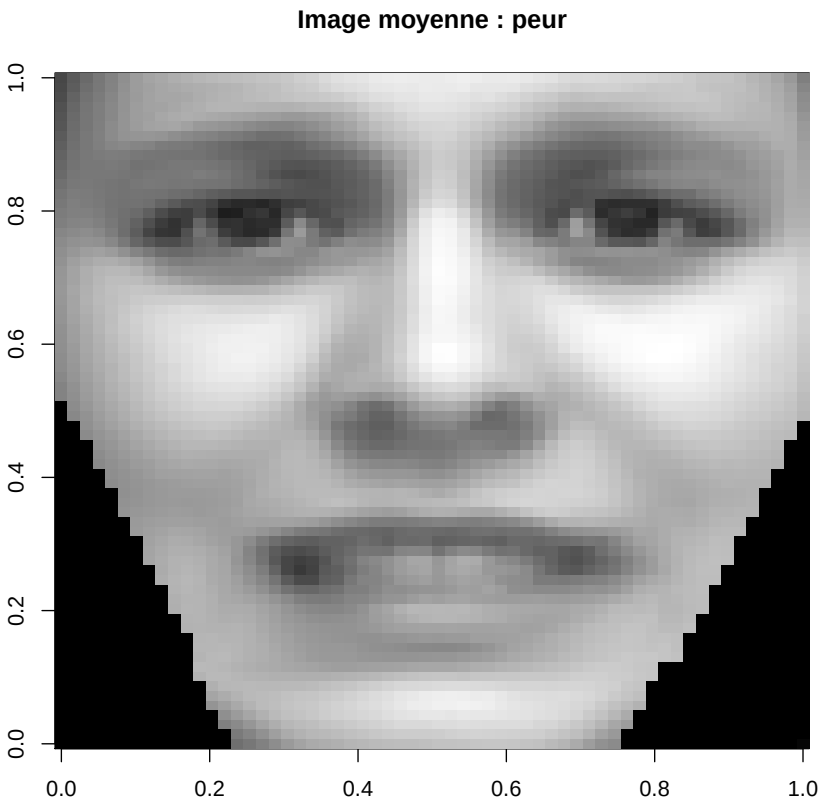

FIGURE 2.12 – Visage moyen : peur

Chapitre 3

Principal Component Analysis

3.1 Principe

Principe RESUME DU PRINCIPE ICI ?

Motivation Comme expliqué précédemment, nous sommes en présence d'un dataset comprenant un nombre restreint d'individus, chacun présentant un nombre élevé de prédicteurs. Il nous parait donc nécessaire d'utiliser un méthode de réduction de dimensions avant toute analyse. Nous décidons d'utiliser la PCA.

3.2 PCA - Réalisation

R Nous réalisons la PCA à l'aide de la fonction `prcomp` de R. Cette fonction effectue l'analyse en composantes principales de notre matrice d'individus de manière automatique et nous fourni ainsi la matrice de composantes principales, leurs variances, ainsi que la matrice de rotation.

Proportion de variance expliquée Afin de ne retenir qu'un certain nombre de composantes principales, nous nous intéressons à la proportion de variance expliquée par chacune d'elles en plottant leur proportion de variance expliquée brute, et cumulée. Les courbes obtenues sont les suivantes.

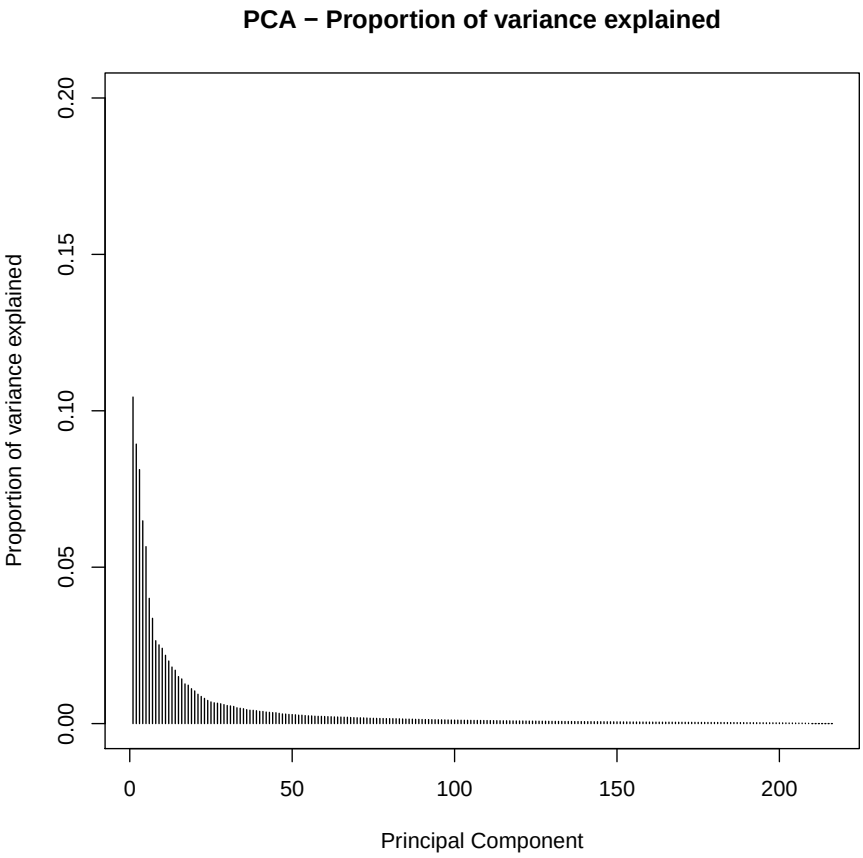

FIGURE 3.1 – PCA : Proportions de variance expliquée

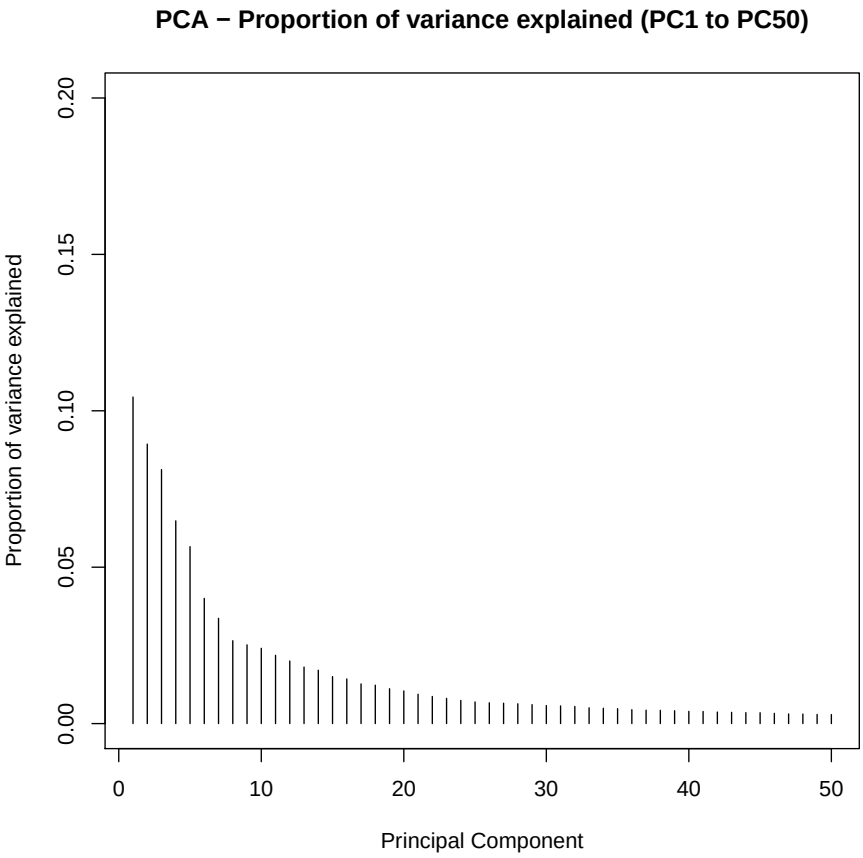

FIGURE 3.2 – PCA : Proportions de variance expliquée (50 premières PC)

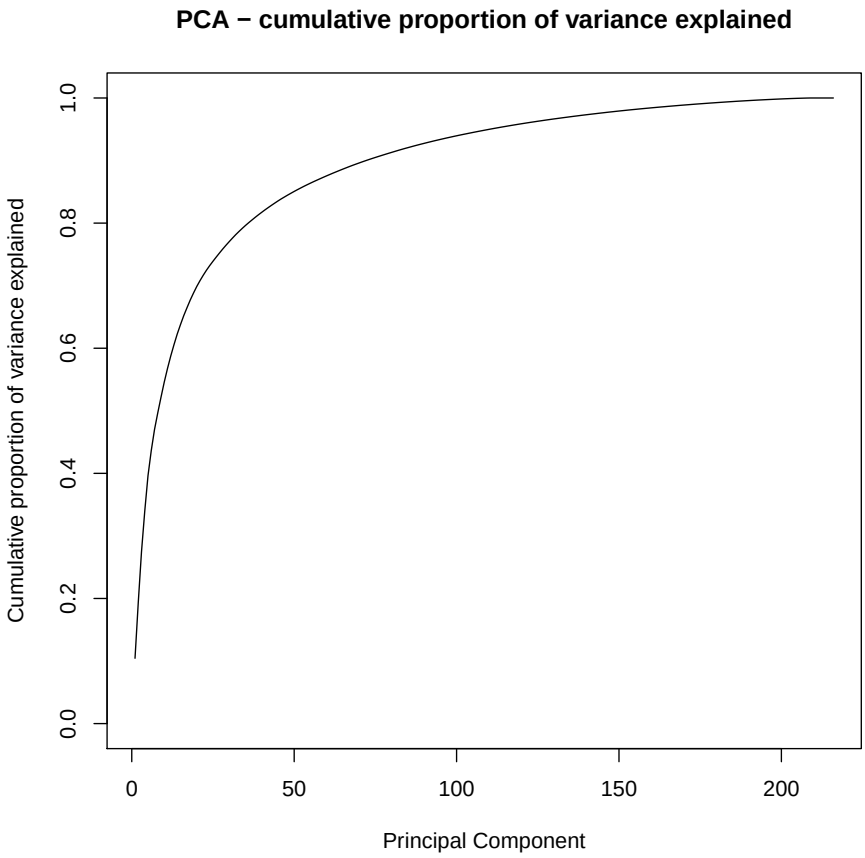

FIGURE 3.3 – PCA : Pourcentage de variance expliquée cumulée

Interprétation Afin de ne garder qu'un nombre très restreint de composantes principales, nous décidons de choisir un seuil de sélection de 75% environ. Nous sélectionnons à cet instant de notre TP les 15 premières composantes principales pour la suite de nos expériences.
NB : Si nous disposons d'assez de temps, nous tenterons peut être une sélection de composantes principales par Forward Stepwise Selection.

Deux premières PC Nous tentons également de plotter nos deux premières principal component, dans l'espoir de constater un quelconque pattern intéressant.

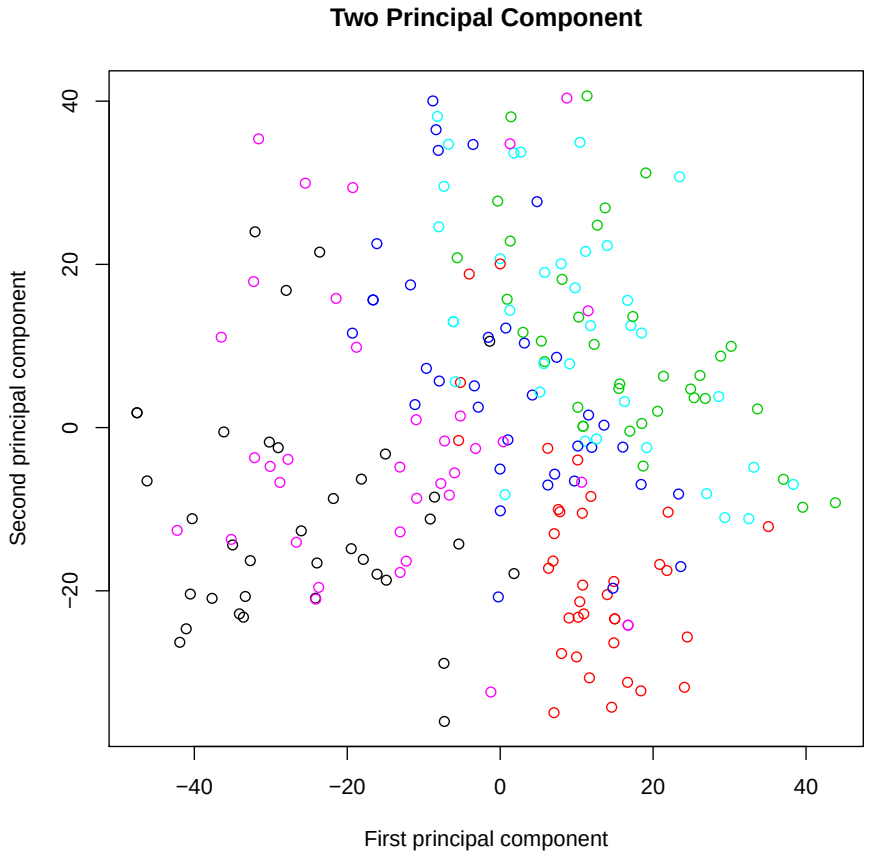

FIGURE 3.4 – PCA : Deux premières composantes principales

Interprétation Nous remarquons une relativement bonne répartition "par classe" des points : les vert en haut à droite, les turquoise également, les bleus de préférence vers le centre, les rouges centrés en bas, les violets et noirs à gauche, avec les noirs de préférence vers le bas. Nous ne nous attendions pas à ce que les individus aient autant tendance à être rassemblés par classe (bien que cela soit loin d'être parfait).
Cela nous laisse de nouveau penser que nous pouvons espérer des performances intéressantes de la part de nos modèles de machine learning.

Continuons Lançons nous maintenant dans notre comparaison des méthodes de machines learning pour cette tâche de facial recognition.

Chapitre 4

Linear Discriminant Analysis, Quadratic Discriminant Analysis

Introduction Les deux premier modèles que nous décidons d'éprouver sont la LDA et la QDA (que nous maitrisons le mieux étant donné qu'il s'agit des premiers étudiés en SY19 de ce TP).

4.1 LDA & QDA : Protocol experimental

6-folds CV Nous souhaitons estimer le taux d'erreur de test de ces deux modèles. Pour cela nous allons utiliser la méthode de resampling dite "6-folds cross validation" (Pourquoi 6 plutôt que 5 ? Parce que 216/6 est un chiffre rond : 36).

Apprentissage / Prédiction Nous effectuons donc pour chaque modèle 6 apprentissages sur les 5 folds de train, suivis de leur prédiction sur le fold de test.

Estimation de l'erreur de test Nous estimons par la suite l'erreur de test en calculant l'erreur de cross validation.

Confusion matrix Nous construisons par ailleurs la matrice de confusion issue de la prédiction.

4.2 LDA & QDA : Résultats

Test error rate La boxplot ci-dessous présente le taux d'erreur des deux modèles.

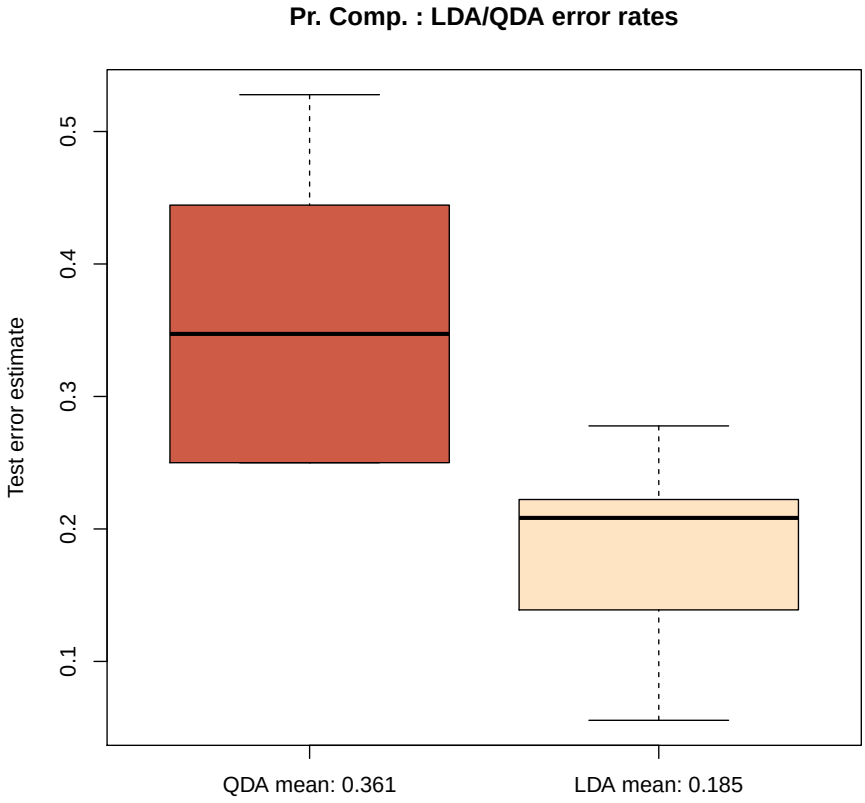

FIGURE 4.1 – LDA & QDA : Taux d'erreurs de test

Matrice de confusion Les matrice de confusion ci-dessous peuvent être obtenues par LDA et QDA.

TABLE 4.1 – LDA : Matrice de confusion

Actual \ Predicted	Joie	Surprise	Tristesse	Dégoût	Colère	Peur	Total
Joie	31	0	0	1	0	4	36
Surprise	0	34	0	0	2	0	36
Tristesse	0	0	29	2	4	1	36
Dégoût	0	0	1	29	4	2	36
Colère	0	0	3	6	27	0	36
Peur	7	0	2	0	1	26	36
Total	38	34	35	38	38	33	216

TABLE 4.2 – QDA : Matrice de confusion

Actual \ Predicted	Joie	Surprise	Tristesse	Dégoût	Colère	Peur	Total
Joie	21	0	0	1	0	14	36
Surprise	0	32	1	0	0	3	36
Tristesse	1	0	22	3	6	4	36
Dégoût	2	0	1	21	5	7	36
Colère	0	0	4	12	19	1	36
Peur	8	0	1	3	1	23	36
Total	32	32	29	40	31	52	216

Test error rates & Standard deviation Finalement, le tableau ci-dessous donne les valeurs des taux d'erreur de test moyens ainsi que leur écart-types.

TABLE 4.3 – Taux d'erreurs de test et écart-types

	LDA	QDA
Test error rate	0.19	0.36
Standard deviation	0.08	0.11

4.3 LDA & QDA : Interprétations

Observations La LDA semble délivrer de meilleures performances que la QDA. La "joie" a tendance a être étiquetée "peur". La "tristesse" a tendance a être confondue avec "dégoût", "colère", ou "peur". Le "dégoût" est quant à lui confondu avec "colère" ou "peur". La colère est très souvent classifiée "dégoût". Pour finir, la "peur" est régulièrement étiquetée "joie".

Interprétation Les classes concernées par ces confusions semblent correspondre aux classes que nous voyons superposées sur la représentation des deux principales composantes analysée précédemment.
Nous pouvons émettre l'hypothèse que certaines expressions sont difficilement différentiables les unes des autres car relativement superposées sur les composantes principales choisies.

LDA vs QDA il semblerait que la LDA nous permette d'obtenir de meilleures performances que la QDA.
Il semblerait donc que ce modèle, moins flexible, propose de meilleures frontières de décision, réduisant la variance de notre modèle (au prix, certainement, d'un certain biais).

Chapitre 5

Support Vector Machine

5.1 SVM - Principe

Du Maximal Margin Classifier à la SVM blabla ?

Kernel blabla

Construction blabla

5.2 Multinomial classification - Strategy

Stratégies Plusieurs stratégies peuvent être utilisées pour utiliser les SVM dans la cas de tâches de classification entre plusieurs classes.

- One vs One : la stratégie One versus One consiste à construire $\binom{K}{2}$ SVMs, comparant les classes par paires, une à une. Les individus de test sont ensuite classés en utilisant chacun des classifieurs, et on leur attribue la classe la plus fréquemment attribuée par ces derniers.
- One vs All : la stratégie One versus All consiste à construire K SVMs, comparant à chaque fois l'une des classes à toutes les autres. On attribue par la suite aux individus de test la classe ayant l'indice de confiance le plus élevé.

Librairie R - Stratégie Nous utilisons dans le cadre de notre étude la librairie `e1071`, nous fournissant une fonction `svm()`. Il nous paraissait donc important de préciser la stratégie utilisée pour la classification multinomiale. La documentation du package nous a permis de trouver la réponse : la stratégie "one versus one" est utilisée dans le cadre de notre étude.

5.3 Protocol experimental, résultats & interprétations

Démarche Nous partons sans hypothèse a priori concernant le modèle le plus approprié pour notre dataset. Notre démarche va donc consister à effectuer un premier "balayage", en testant un large spectre de paramètres, puis sélectionner un ou plusieurs jeux de paramètres nous ayant donné de bons résultats puis essayer d'optimiser ces jeux de paramètres.

Première analyse Nous commençons donc au cours d'une première analyse à tester ce large spectre de paramètres. Grace à la méthode `tune` du package `R`, nous pouvons faire varier les paramètres suivants :

- **Kernel** : Le noyau utilisé par la SVM
- **Cost** : Le "budget", agissant sur le "bias-variance trade-off" du modèle en permettant plus ou moins de violation de "margin"
- **Degree** : Le degré à utiliser (seulement dans le cas d'un noyau polynomial)

Résultats La figure ci-dessous présente le résultat de ce premier "parameters tuning".

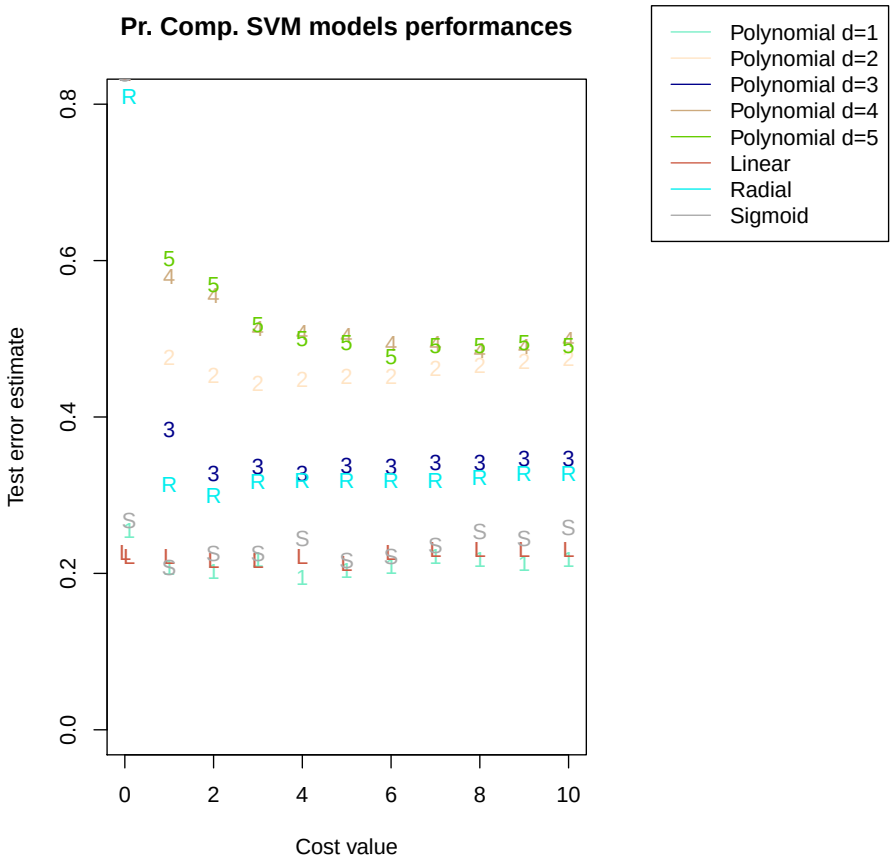

FIGURE 5.1 – SVM : Première analyse

Interprétation - modèles approfondis Au vu de la figure ci-dessus et des performances du tuning que nous avons exportées au format csv (annexes), il semblerait que les modèles les plus performants soient le modèle utilisant un noyau de forme `sigmoid` pour un `cost` égal à 1, et le modèle `polynomial` de degré 1 (linear donc), pour un `cost` avoisinant la valeur 5.
On remarque également les performances relativement médiocres de nos modèles pour un `cost` nul. Cela corrobore ce que nous avons vu en cours : l'ajout du `cost` permet d'optimiser notre "bias-variance tradeoff".
Nous sélections ces deux modèles, afin de leur faire passer une phase d'optimisation supplémentaire.

Modèle Sigmoid - Optimisation Nous commençons par optimiser le modèle utilisant un noyau `sigmoid`. nous lui faisons passer deux phases supplémentaires de "tuning". Au cours de la première, nous faisons varier la valeur de `gamma` et de `cost`, deux paramètres utilisés par ce modèle. Nous obtenons les meilleures performances pour les valeurs suivantes :

TABLE 5.1 – Résultats de première optimisation du modèle à noyau sigmoid

Gamma	Cost	Test error estimate
0.007	1.5	0.24

Optimisation du cost Nous fixons par la suite gamma à la valeur obtenue et réalisons notre dernière optimisation, sur le paramètre `cost`. La courbe ci-dessous représente l'évolution de de l'estimation du taux d'erreur de test en fonction de la valeur du paramètre `cost`.

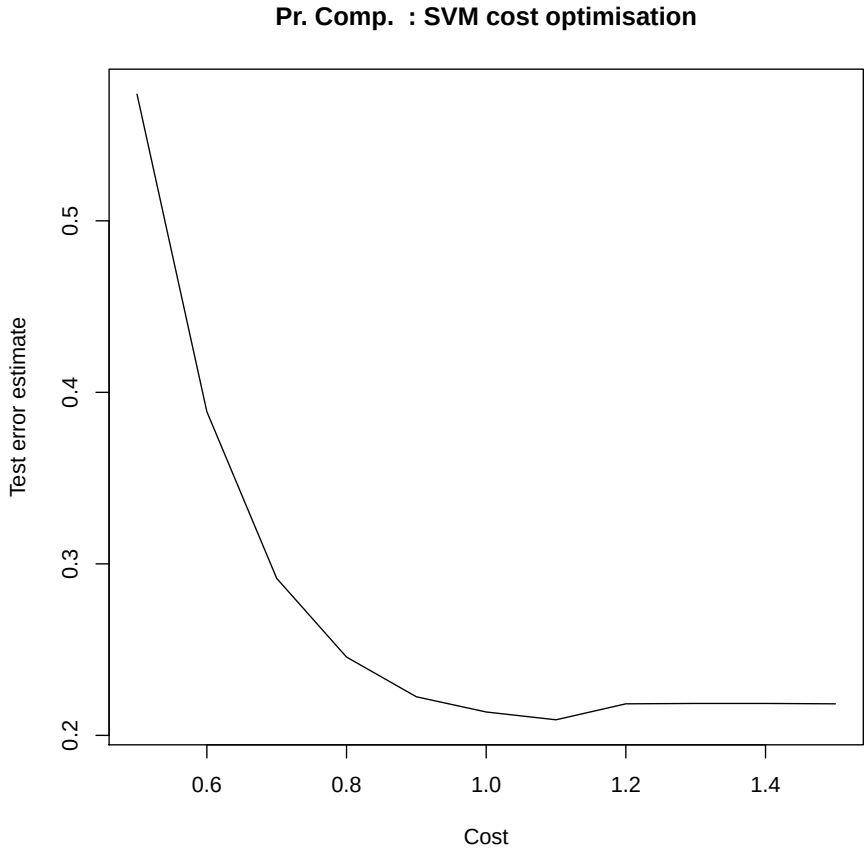

FIGURE 5.2 – SVM : optimisation du cost pour le modèle à noyau sigmoid

Performances finales - noyau sigmoid Finalement, nous obtenons le jeu de paramètre et les performances suivantes :

TABLE 5.2 – Résultats de seconde optimisation du modèle à noyau sigmoid

Gamma	Cost	Test error estimate
0.007	1.1	0.21

Modèle polynomial - optimisation Tout d'abord, il est à noter que le modèle polynomial, au degré 1 est un modèle linéaire, à un facteur `gamma` près. De la même manière que nous l'avons fait pour le modèle à noyau sigmoid, nous "tunons" les deux paramètres de ce modèle. Après la première optimisation, nous obtenons le jeu de paramètres suivant :

TABLE 5.3 – Résultats de première optimisation du modèle à noyau polynomial de degré 1

Gamma	Cost	Test error estimate
0.01	4.3	0.22

Cost optimisation Une nouvelle fois, nous fixons notre `gamma` et effectuons le tuning du `cost`. Nous obtenons la courbe ci-dessous.

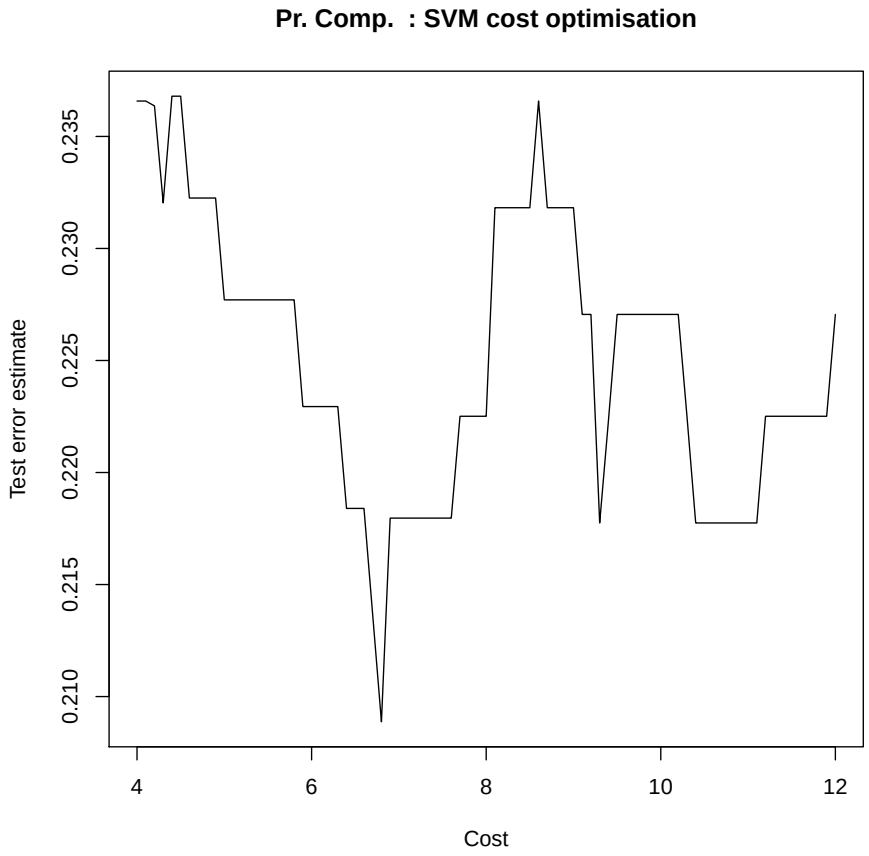

FIGURE 5.3 – SVM : optimisation du cost pour le modèle à noyau polynomial de degré 1

Performances finales - noyau polynomial Après cette nouvelle optimisation, nous obtenons le jeu de paramètres et les performances suivantes :

TABLE 5.4 – Résultats de seconde optimisation du modèle à noyau polynomial de degré 1

Gamma	Cost	Test error estimate
0.01	6.8	0.21

Test errors & confusion matrix Pour finir, nous utilisons notre 6-fold cross-validation "homemade" afin dessiner, pour chacun des deux modèles, boxplots d'estimations d'erreur de test, et matrices de confusion ci-dessous.

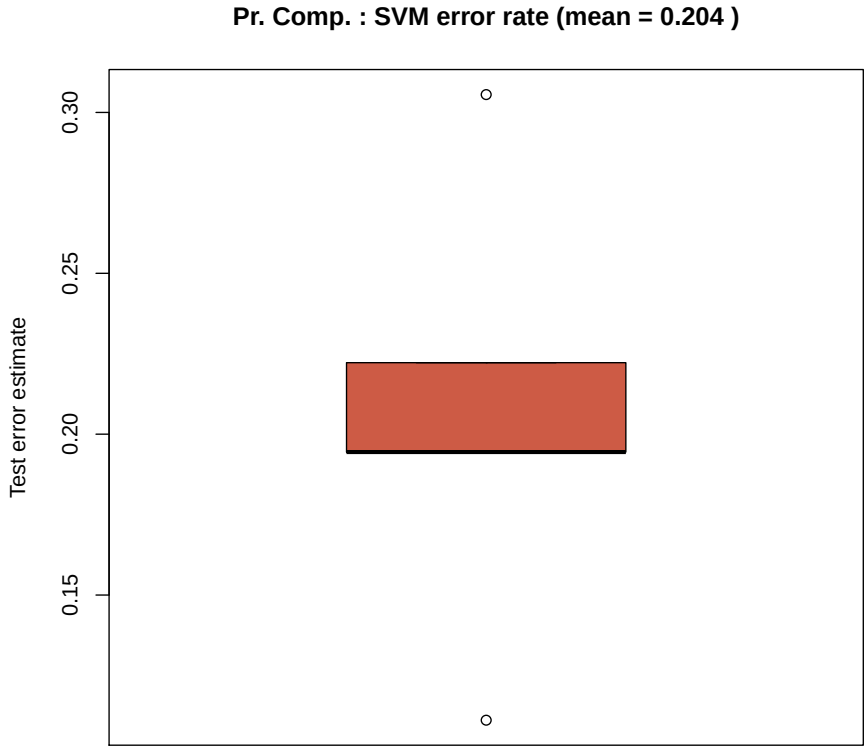

FIGURE 5.4 – SVM : estimation du taux d'erreur pour le modèle à noyau polynomial de degré 1

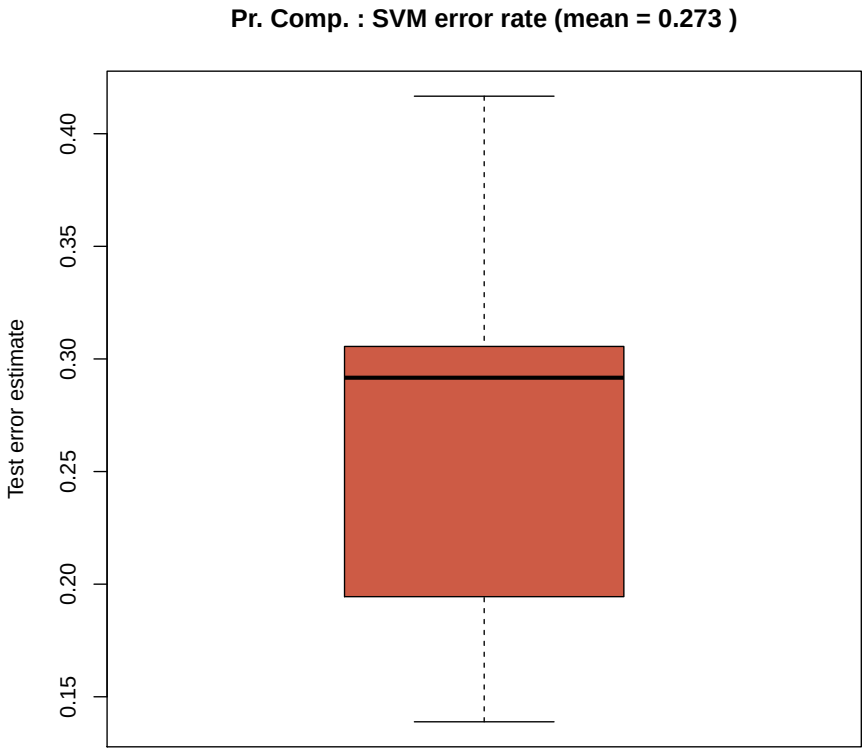

FIGURE 5.5 – SVM : estimation du taux d'erreur pour le modèle à noyau sigmoid

TABLE 5.5 – SVM - polynomial d=1 : Matrice de confusion

Actual \ Predicted	Joie	Surprise	Tristesse	Dégoût	Colère	Peur	Total
Joie	33	0	0	0	0	3	36
Surprise	0	36	0	0	0	0	36
Tristesse	0	0	27	1	6	2	36
Dégoût	1	0	0	28	6	1	36
Colère	0	0	4	6	26	0	36
Peur	8	1	1	2	2	22	36
Total	42	37	32	37	34	28	216

TABLE 5.6 – SVM - sigmoid : Matrice de confusion

Actual \ Predicted	Joie	Surprise	Tristesse	Dégoût	Colère	Peur	Total
Joie	32	0	0	1	0	3	36
Surprise	0	32	0	0	4	0	36
Tristesse	0	0	27	2	7	0	36
Dégoût	0	0	1	23	12	0	36
Colère	0	0	4	11	20	1	36
Peur	6	0	3	3	1	23	36
Total	38	32	35	40	44	27	216

Interprétations finales Le modèle à noyau polynomial de degré 1 (donc linéaire) performe mieux que le modèle à noyau sigmoid. Son taux d'erreur de test est moins important, sa variance également. Cela semble corroborrer les résultats obtenus précédemment, lors de la LDA.
Notons également de fortes similitudes entre la matrice de confusion obtenue via SVM à noyau polynomial de degré 1, et celle de la LDA (sensiblement les mêmes confusions ont lieu).
Tout comme celles de la LDA, les performances, de la SVM linéaire sont relativement convenables pour cette tâche de facial recognition ! Poursuivons notre analyse à l'aide des Random Forest.

Chapitre 6

Random Forest

6.1 Introduction

Principe PRINCIPE ICI ?

6.2 Protocol expérimental, résultat & interprétations

Démarche Une nouvelle fois, nous partons avec aucun a priori concernant le modèle de Random Forest optimal pour notre dataset. De la même manière que nous l'avons fait pour les modèles de type SVM, nous allons effectuer un premier "balayage" des différents paramètres à optimiser, afin de nous faire un première idée du jeu de paramètres optimal. Les paramètres sur lesquels nous allons travailler dans la cadre de la Random Forest sont :

- **Ntree** : Le nombre d'arbres dans notre forêt
- **Mtry** : Le nombre de variables candidates à chaque split

Optimisation Construisant ici la Random Forest sur les composantes principales de notre dataset, les prédicteurs sont au nombre de 15. La construction des Random Forest n'étant pas trop contraignante du point de vue temps de calcul nous allons pouvoir immédiatemment tester toutes les valeurs entre 1 et 15 pour le paramètre `mtry`. Ainsi, notre résultat de tuning sera notre résultat final d'optimisation.
Le "tuning" du jeu de paramètres nous donne ainsi la plot suivante :

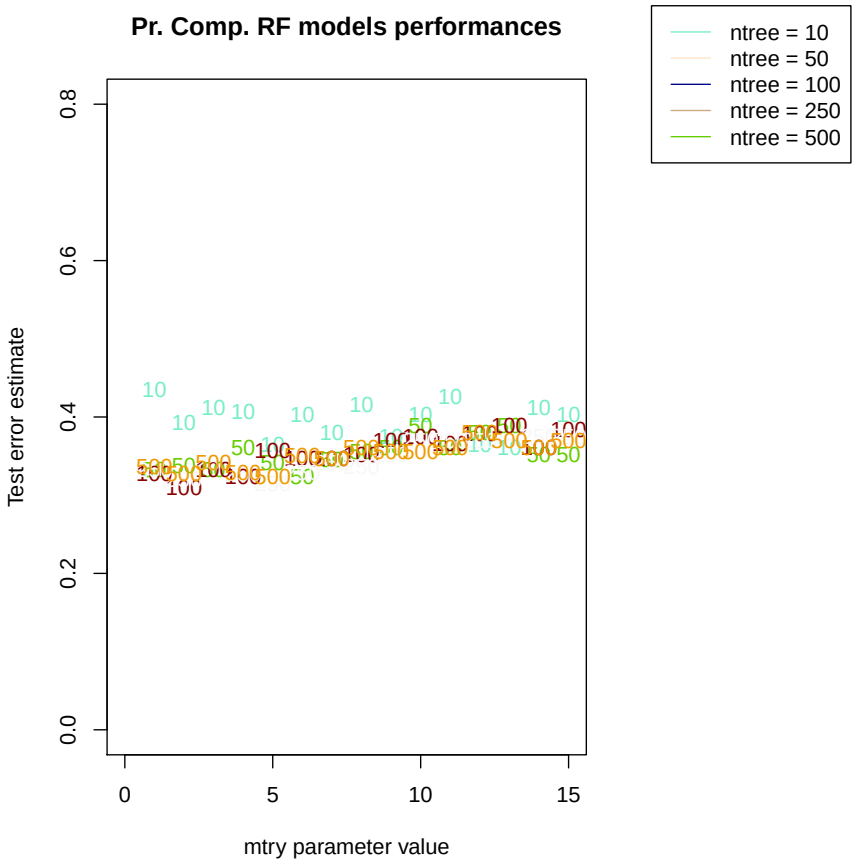

FIGURE 6.1 – RF : Résultat de tuning du jeu de paramètres

Interprétation Cette figure nous permet de voir que les performances de la Random Forest s'améliorent bien au fur et à mesure de l'augmentation du nombre d'arbres dans la forêt, jusqu'à se stabiliser pour un nombre élevé d'entre eux. Notre tuning nous retourne en tant que meilleur jeu de paramètres :

TABLE 6.1 – Résultat de première optimisation de la Random Forest

ntree	mtry	Test error estimate
100	2	0.31

Matrice de confusion et taux d'erreur de test Une fois encore, nous plottons l'estimation du taux d'erreur de test du modèle, ainsi que sa matrice de confusion en utilisant une 6-folds cross validation.

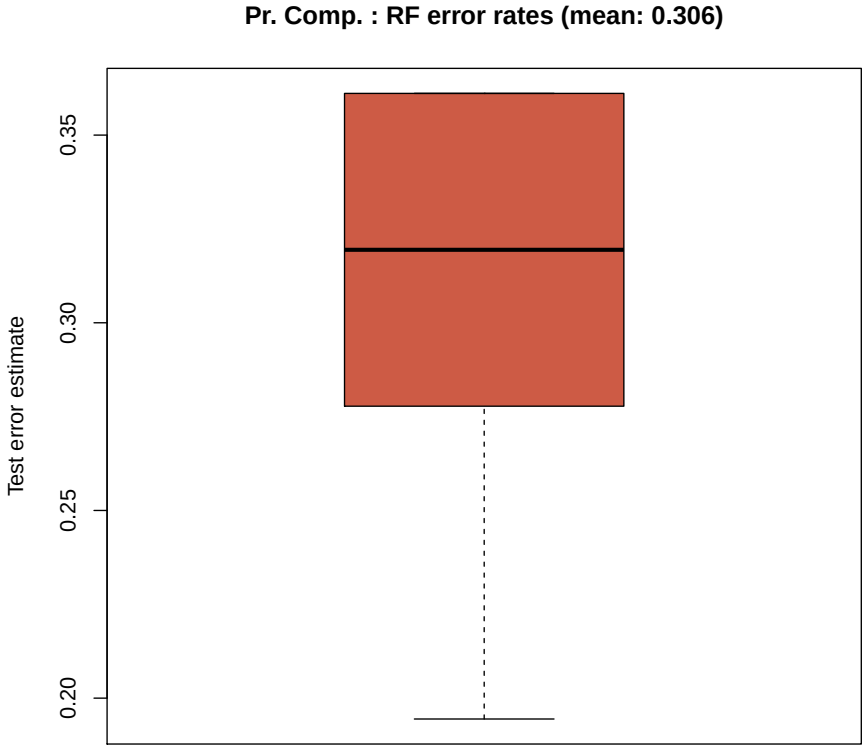

FIGURE 6.2 – RF : Estimation du taux d'erreur de test

Table 6.2 – Random Forest : Matrice de confusion

Actual \ Predicted	Joie	Surprise	Tristesse	Dégoût	Colère	Peur	Total
Joie	27	0	1	0	1	7	36
Surprise	0	32	0	2	1	1	36
Tristesse	0	0	26	5	4	1	36
Dégoût	0	2	4	20	7	3	36
Colère	0	0	5	6	24	1	36
Peur	7	1	2	2	3	21	36
Total	34	35	38	35	39	33	216

Interprétation Avec la Random Forest, nous n'obtenons des performances bien moins intéressantes qu'avec la LDA, ou la SVM polynomiale de degré 1. De plus la variance de ce modèle semble relativement élevée.
Nous pouvons émettre l'hypothèse d'un éventuel overfitting : peut être qu'en prunant les arbres composant la forêt, nous ajouterions un léger biais mais améliorerions le "bias-variance tradeoff" du modèle.

Chapitre 7

Neural Networks

7.1 Introduction

Principe PRINCIPE HERE ?

7.2 Protocole expérimental, résultats & interprétations

Démarche Une nouvelle fois, nous n'avons pas d'a priori au moment de commencer nos essais sur le modèle convenant le mieux pour la résolution de notre problème. Nous allons donc procéder, comme précédemment, à un premier balayage suivi d'une optimisation du jeu de paramètre choisis suite au premier balayage. Les paramètres que nous nous apprêtons à optimiser sont les suivants :

- **Size** : Le nombre d'unité internes au Neural Network
- **Decay** : Valeur utilisée en tant que "Weight decay" (TO DO : PLUS DE PRECISIONS ICI)

NB : Package R Nous utilisons le package `R` le plus "basique" pour notre implémentation de Neural Network. Ce dernier ne nous permet d'implémenter qu'un simple "Single-Hidden-Layer Neural Network", c'est à dire un NN ne présentant qu'une seule couche cachée. Ainsi, nous n'implémentons pas (pour l'instant du moins) de réseau de neurones présentant des features telles qu'une "connectivité locale", ou des "poids partagés" ("Locally connected Neural Network" et "Convolutional Neural Network"). Peut être prolongerons nous cet TP par leur implémentation, si le temps ne nous manque pas.

Première optimisation Suite à notre premier "balayage", nous obtenons la plot suivante :

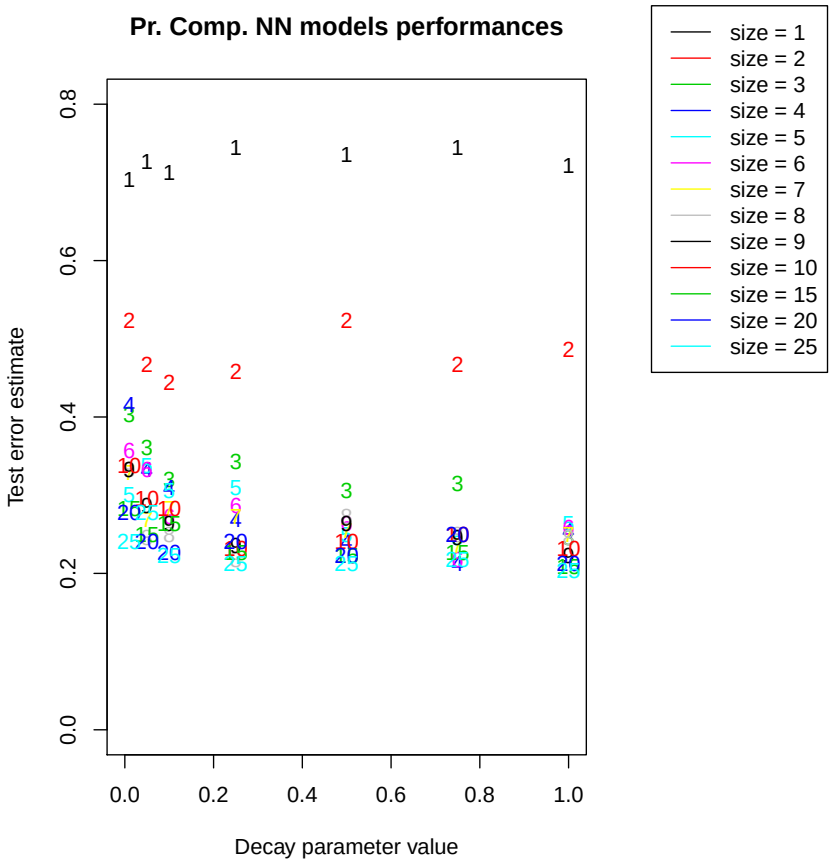

FIGURE 7.1 – NN : Résultat de tuning du jeu de paramètres

Interprétation Les performances de notre réseau de neurone s'améliorent avec l'augmentation du nombre d'unités au sein de la "hidden layer", jusqu'à ce stabiliser pour un nombre élevé.
Cela se confirme, lorsque nous examinons l'output csv (ci-dessous) de notre tuning, selon lequel c'est lorsque nous avons choisi un paramètre de taille égal à 25 (le maximum que nous ayons testé) que nous obtenons les meilleures performances.

TABLE 7.1 – Résultat de première optimisation du Neural Network

size	decay	Test error estimate
25	1	0.20

Optimisation du Weight Decay Fixant la taille de notre NN à 25, nous optimisons le paramètre de `weight decay` et obtenons la courbe ci-dessous.

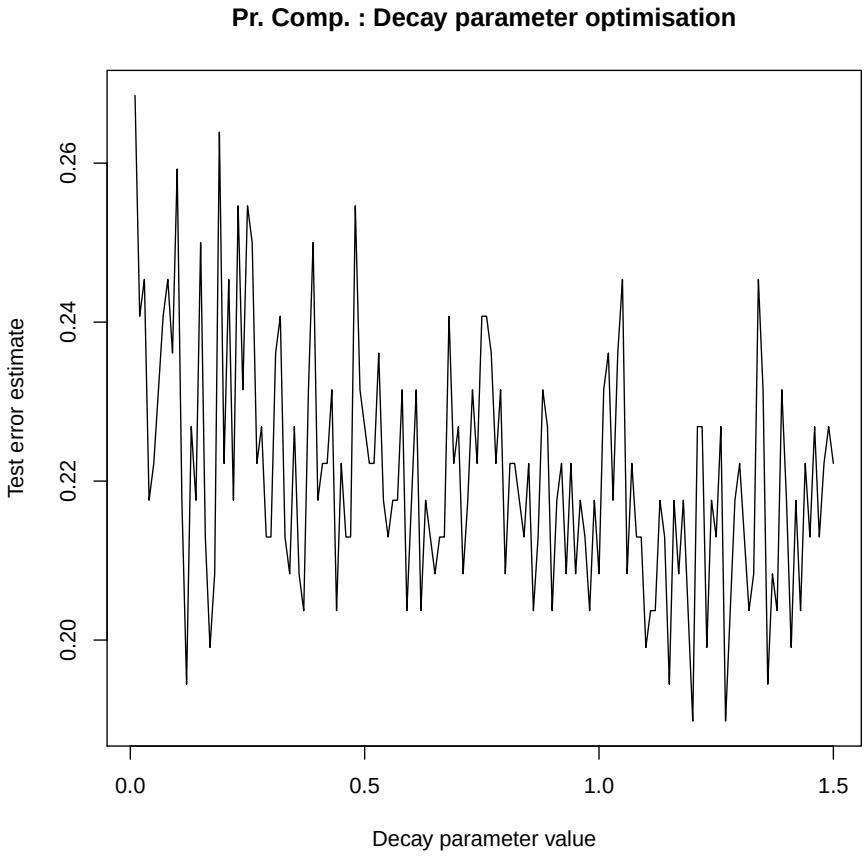

FIGURE 7.2 – NN : optimisation du weight decay

Jeu de paramètres optimal Nous obtenons après optimisation le jeu de paramètres suivant :

TABLE 7.2 – Résultat de seconde optimisation du Neural Network

size	decay	Test error estimate
25	1.27	0.19

Erreur de test et matrice de confusion Pour finir, nous utilisons une dernière fois (peut être pas la dernière) la 6-fold cross validation pour plotter notre estimation de l'erreur de test de notre NN, et sa matrice de confusion.

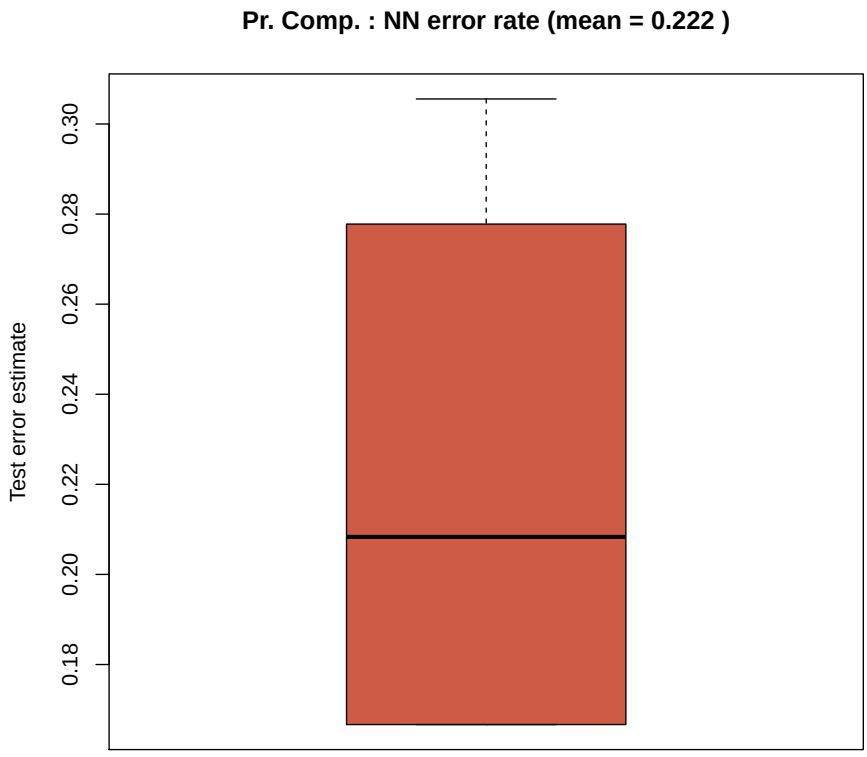

FIGURE 7.3 – NN : Estimation du taux d'erreur de test

TABLE 7.3 – Random Forest : Matrice de confusion

Actual \ Predicted	Joie	Surprise	Tristesse	Dégoût	Colère	Peur	Total
Joie	31	0	0	1	0	4	36
Surprise	0	36	0	2	1	1	36
Tristesse	0	0	27	4	4	1	36
Dégoût	1	1	1	28	4	1	36
Colère	0	0	5	8	21	2	36
Peur	7	0	2	0	2	25	36
Total	34	35	38	35	39	33	216

Interprétation L'estimation du taux d'erreur de test du NN optimisé ainsi obtenu est relativement basse, cependant sa variance est relativement élevée, ce qui n'en fait pas le modèle le plus sûr.

Chapitre 8

Analyses complémentaires

ICI Eventuelles analyses complémentaires parmi :

- Utiliser la Forward Stepwise Selection pour choisir les principal component à envoyer à nos méthodes d'opti
- Prendre plus de princomp
- Tester les différentes méthodes optimisées sur le dataset initial ? (je pense que ça fonctionne sur certaines)
- ...

Chapitre 9

Conclusion

9.1 Comparaison des classifieurs

Taux d'erreur de test ICI : boxplot rassemblant tous les modèles, et paragraphe d'interprétation tirant des conclusions blabla

9.2 Perspectives

Volonté vs time Nous aurions aimé avoir le temps de blablabla...

— Neuralnet pour réseaux de neurons plus complexes

— ...

9.3 Conclusion

Conclusion Ce TP nous aura permis de ... blablagagaga