Rapport TP n°6 SY19

Stephane Louis - Karim Amrane

3 Janvier 2018

Apprentissage à partir de trois jeux de données réelles

1. Introduction

L'objectif du TP est d'étudier trois jeux de données réelles afin de construire des classifieurs aussi performants que possible par l'apprentissage. Chaque jeu de données contient une variable qualitative à expliquer et concerne donc le domaine de l'apprentissage supervisé.

Il sera parfois nécessaire d'appliquer des méthodes d'apprentissage non supervisée au préalable dans le but d'améliorer les performances des classifieurs. L'objectif principal est de comparer différents classifieurs dans le but de prévoir l'appartenance ou non d'un individu à un groupe donné.

Puis en qualifiant les classifieurs sur différents critères, il en viendra de déterminer le meilleur prédicteur et d'en justifier sa raison. Tout au long du semestre nous avons pu découvrir de nombreuses techniques de machine learning que nous allons appliquer sur ces jeux de données.

2. Méthodologie

Lors de ce TP, nous avons adopté la même méthodologie globale pour chacun des jeux de données.
En effet, nous avons commencé par une analyse exploratoire des différents datasets afin d'évaluer leurs caractéristiques, leurs qualités et leurs défauts.

Par la suite, nous avons gardé un dataset de données brutes scalées, puis nous avons effectué si nécessaire un nettoyage des données ainsi que des méthodes de réduction de dimensions afin de créer de nouveaux datasets supplémentaires sur lesquels tester les classifieurs.

Nous avons ensuite créé des fonctions pour chacun des différents classifieurs qui optimisent les hyper-paramètres en utilisant le grid-search et qui appliquent une cross-validation avec la technique des k-folds (avec 6 folds), puis en réitérant un grand nombre de fois ces fonctions pour stabiliser les résultats.

Enfin, nous avons récolté les tables de confusion et par conséquent les taux d'erreurs et variances pour chaque classifieur utilisé afin de choisir le plus adapté à chacun des trois datasets. Dans les tableaux récapitulatifs de résultats, les valeurs "NC" correspondent aux valeurs non calculées ou qui dépassent 50% d'erreur, et les valeurs "NA" correspondent aux valeurs qui n'étaient pas calculables.

Ci-dessous l'exemple d'une des fonctions que nous avons utilisé pour tester nos classifieurs (celle d'un perceptron multi-couches, qui nécessite le package mxnet) :

```
mlp.cv <- function(r, k, data,hn,on,round,bsize,lr, type)

  {
      err_rate=vector(length=r)
      for(i in 1:r){
          n = nrow(data)
```

```
        m=ncol(data)
        folds = sample(1:k,n,replace=TRUE)
        CV = 0
#cross-validation
        for(j in (1:k)){
            e.train=data[folds!=j,]
            e.test=data[folds==j,]
#using FDA during cross-validation
            if(type=="fda"){
                e.train.lda<-lda(y~.,e.train)
                e.train.fda<- as.matrix(e.train[,1:m-1])%*%e.train.lda$scaling
                e.train.fda<-cbind(as.data.frame(e.train.fda), e.train$y)
                colnames(e.train.fda)[dim(e.train.fda)[2]] = "y"
                e.test.fda<-as.matrix(e.test[,1:(m-1)])%*%e.train.lda$scaling
                e.test.fda<-cbind(as.data.frame(e.test.fda), e.test$y)
                colnames(e.test.fda)[dim(e.test.fda)[2]] = "y"
                e.train<-e.train.fda
                e.test<-e.test.fda
            }

#MLP model generation
            model <- mx.mlp(as.matrix(e.train[,1:ncol(e.train)-1]), e.train$y, hidden_node=hn,
                            out_node=on, out_activation="softmax", num.round=round,
                            array.batch.size=bsize, learning.rate=lr, momentum=0.9,
                            eval.metric=mx.metric.accuracy, array.layout = "rowmajor")
            pred.test<-predict(model, as.matrix(e.test[,1:ncol(e.test)-1]))
            ntst<-nrow(e.test)
            pred.test.clean<-vector(length=ntst)
#Finding the best class
            for (j in 1:ntst){
                pred.test.clean[j]<-which(pred.test[,j]==max(pred.test[,j]), arr.ind = T)-1
                if(as.numeric(pred.test.clean[j])==as.numeric(e.test$y[j])){
                    CV=CV+1
                }
            }
        }
        err_rate[i] <- CV/n
    }
    print(var(err_rate)) # variance of error rates
    print(mean(err_rate)) # mean error rate
    return(1-mean(err_rate)) # accuracy
}
```

3. Jeu de données Expressions

3.1. Analyse préparatoire

Ce jeu de données nous a semblé être le plus difficile à classifier des trois jeux.
En effet, nous avons été tout d'abord frappés par le faible nombre d'individus par rapport au nombre de prédicteurs (108 individus seulement pour 4200 prédicteurs). Ce ratio très faible nous force à imaginer que notre classifieur ne sera pas de très bonne qualité.

De plus, le jeu de données étant constitué d'images de visages, les prédicteurs sont fortement corrélés et l'utilisation de techniques de réduction de dimensions semble indispensable pour éviter la "curse of dimensionnality".

Les images sont déjà en niveau de gris, sont centrées sur les visages et sont toutes du même format (avec des pixels noirs sur les côtés en bas des joues), ce qui est plutôt positif.

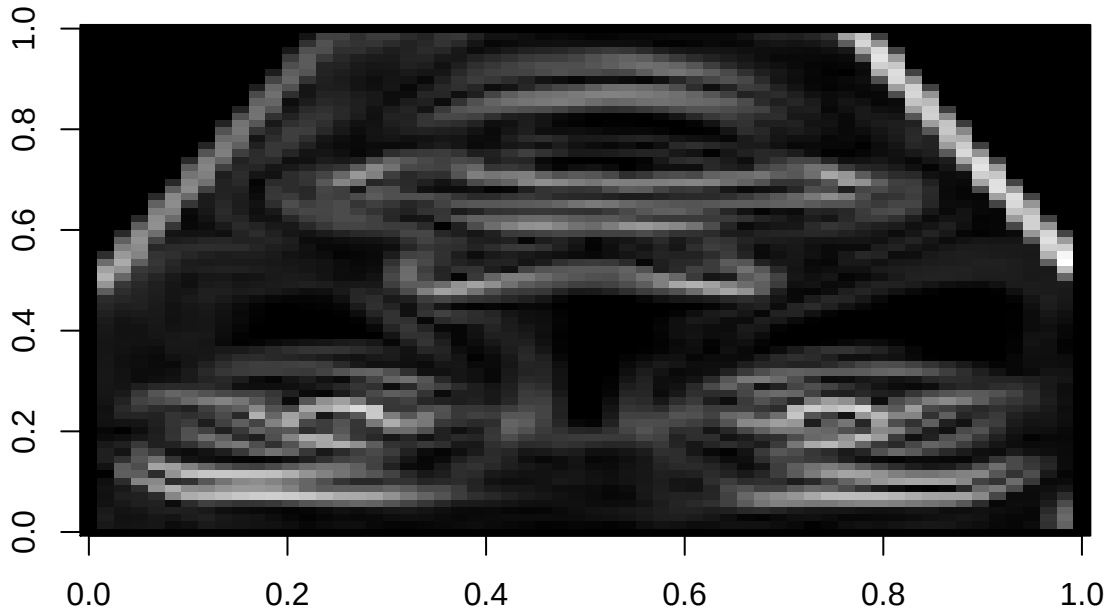
3.2. Nettoyage et préparation des données

Pour nettoyer le jeu de données, plusieurs possibilités s'offrent à nous.
Tout d'abord, les visages étant pour une grande majorité quasiment symétriques, nous avons décidé pour tester de n'en garder qu'une moitié afin de voir si diminuer le nombre de prédicteurs permettait d'améliorer les performances.

Après essai, les résultats étaient moins bons avec seulement la moitié des pixels, chose qui nous a amenés à conserver la totalité du dataset. Nous avons également scalé les données et supprimé les pixels noirs redondants dans toutes les images, étant donné qu'ils n'ont aucune utilité en tant que prédicteurs.

Enfin, nous avons également créé un nouveau dataset d'images après utilisation d'un filtre de Sobel (détection de contours, package adimpro).

Voici une image illustrant l'application du filtre sobel sur une observation du dataset expression :



Malheureusement encore une fois, ce dataset ne nous a pas permis d'obtenir de meilleurs résultats que les autres.

Afin de réduire encore le nombre de prédicteurs, nous avons utilisé en priorité la FDA (qui consiste à multiplier la matrice des individus par la matrice scaling donnée par la méthode LDA).

Il est important de noter qu'afin d'avoir des taux d'erreurs représentatifs, il faut générer la matrice scaling de la LDA à chaque itération de la cross validation avec les folds d'apprentissage, puis l'appliquer sur le fold de test.

Si l'on utilise directement la FDA sur tout le dataset, les résultats sont biaisés car les ensembles de test auront "influencé" la FDA et donc le taux d'erreur final sera bien plus faible que ce qu'il devrait être.

En ayant fait l'erreur au début, nous nous sommes aperçus que le taux d'erreur moyen (faussé) pour les données expressions était de seulement 5% pour certains classifieurs si l'on utilise qu'une seule fois la FDA sur le dataset d'origine, ce qui est bien loin du taux d'erreur réel obtenu à la fin de ce TP !

Par ailleurs, étant donné le faible nombre d'individus, le grand nombre de prédicteurs et la corrélation entre ces derniers, nous avons décidé d'utiliser également la PCA (fonction prcomp) sur ce dataset, pour laquelle nous avons gardé 27 composantes principales, qui correspondent à environ 80% de proportion de variance expliquée.

L'utilisation de la PCA permet de maximiser la variance dans le dataset et donc peut-être de diminuer l'overfitting et d'améliorer nos résultats. Nous avons également combiné les deux méthodes c'est à dire utilisé la FDA sur le dataset PCA créé précédemment afin de profiter des avantages de chacune.

Enfin, pour aller plus loin, nous avons essayé d'utiliser la méthode PenalizedLDA du package du même nom, pour appliquer la pénalité lasso sur les vecteurs discriminants. Malheureusement, cette méthode n'a pas fonctionné et renvoyait un vecteur de prédictions à une seule modalité.

3.3. Utilisation des classifieurs

Les données étant des images, nous avons tout d'abord pensé que l'utilisation d'un SVM ou d'un CNN serait la solution optimale pour notre problème.

- **SVM :**

  Après avoir utilisé la fonction tune out pour déterminer les meilleurs hyper-paramètres pour un SVM, nous avons constaté que les kernels linéaire et radial étaient les meilleurs sur ce jeu de données. L'utilisation du dataset réduit par FDA semble donner des résultats meilleurs que les autres.

- **CNN :**

  Les réseaux de neurones convolutifs sont connus pour être particulièrement efficaces pour les problématiques de traitement d'image, c'est pourquoi nous avons décidé de les tester rapidement sur notre jeu de données.

  Étant donné que le principe même des réseau de neurones convolutifs est de détecter des régions d'intérêt au sein des images (c'est à dire grossièrement de trouver des liens entre les pixels proches les uns des autres), nous avons pris le dataset sans traitement préalable, que nous avons passé sous forme de matrices 60*70.

  En effet, l'utilisation d'un dataset avec réduction de dimension supprime cette proximité entre prédicteurs et donc fait perdre l'intérêt de ce type de réseau de neurones.

  À l'aide du package mxnet, nous avons pu essayer différents types de CNN, avec entre une et trois couches de convolutions, entre une et deux couches entièrement connectée, et en faisant varier tous les hyper-paramètres (nombre de rounds, fonctions d'activation, batch size, learning rate, momentum, taille du noyau de convolution, taille du noyau de pooling et nombre de neurones par couches).

Voici une visualisation d'un CNN utilisé dans notre cas (fonction graph.viz du package mxnet) :

Malheureusement, l'apprentissage d'un CNN étant très lent et nos machines peu performantes, nous avons été limités à quelques dizaines de combinaisons, par manque de temps. En cela, nous n'avons pas réussi à trouver de modèle de CNN performant sur notre jeu de données. La plupart des configurations essayées ne permettaient pas au modèle de converger en un temps raisonnable.

La meilleure performance que nous ayons obtenue avec un CNN était de 35.5% d'erreur avec 300 rounds, un learning rate de 0.005, un batch size de 10, des fonctions d'activation relu (plus rapides que des fonctions tanh ou sigmoid), deux couches de convolution et deux couches entièrement connectées. Il est possible que le faible nombre de données nous ait empêché de profiter pleinement de la puissance des CNN.

De plus, un des intérêts principaux de ce type de réseau de neurones est qu'il permet de retrouver des régions d'intérêt de manière insensible aux transformations (rotations, translations.). Or dans notre jeu de données, les visages sont toujours positionnés de la même façon, et ainsi cette caractéristique des CNN n'était pas utile dans notre cas, ce qui explique peut-être pourquoi son utilisation n'était pas forcément pertinente pour nous alors qu'elle l'est dans la plupart des problèmes de traitement d'image.

Suite à cette conclusion, nous avons testé plusieurs autres classifieurs, à commencer par un perceptron multi-couches, suivi des random forests, d'une LDA, d'une QDA, des KNN et du classifieur Bayesien Naïf.

- **MLP :**

  Le perceptron multi-couches est un autre type de réseau de neurones qui fonctionne par propagation directe (feed-forward) et qui possède également plusieurs couches cachées. Son fonctionnement se fait par l'optimisation des poids appliqués à chaque neurone par rétropropagation de l'erreur selon l'importance des éléments qui ont participé à cette erreur. Comme pour le CNN, nous avons pu faire varier un grand nombre d'hyper-paramètres afin d'optimiser notre réseau de neurones.

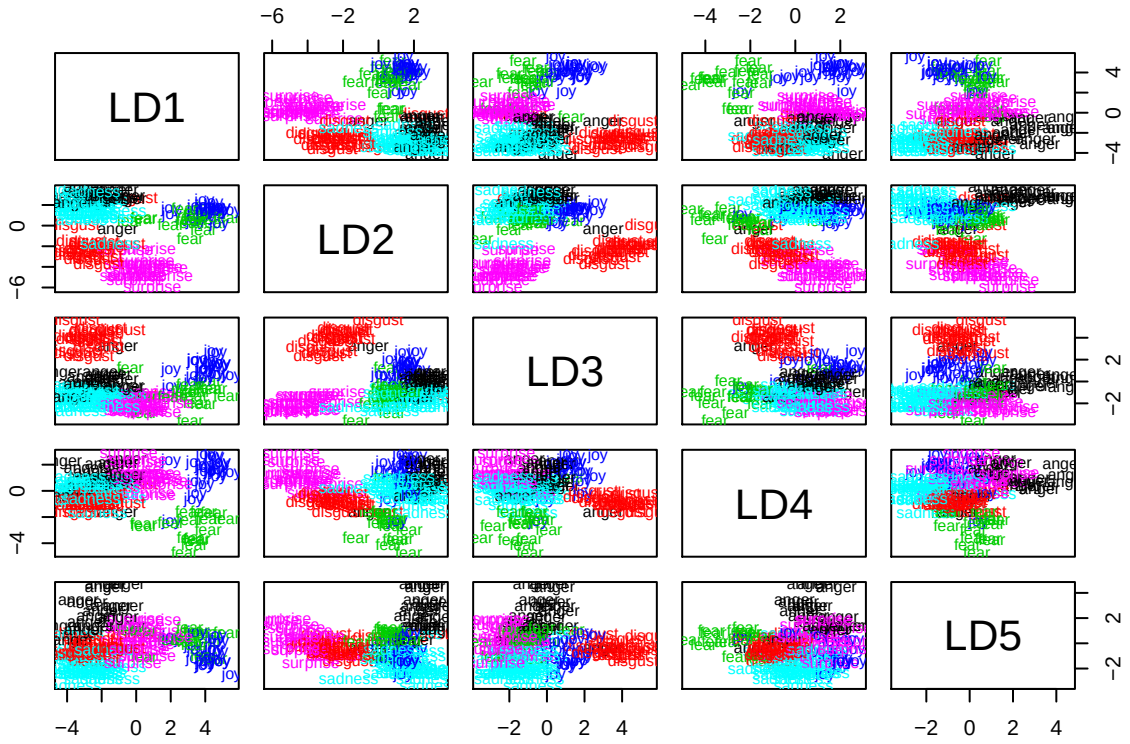
  En mettant le nombre de neurones de la couche de sortie à 6 (c'est à dire le nombre de classes de notre dataset), et en faisant varier les autres paramètres, nous avons réussi à obtenir un taux d'erreur de 24% sur le jeu de données réduit par FDA. Le MLP appliqué au jeu de données brutes ou même sans pixels noirs ne convergeait pas en un temps raisonnable.

- **Random Forests :**

  Les randoms forests sont un autre type de classifieur basé sur un ensemble d'arbres décisionnels et sur le principe de bagging. Pour ce jeu de données, nous avons obtenu au mieux un taux d'erreur de 25.5% sur le jeu de données réduit par FDA, avec l'utilisation de 1000 arbres et le nombre de variables à comparer de 2.

- **LDA :**

  L'analyse discriminante linéaire nous a donné un résultat très correct de 21.1% d'erreur sur le dataset initial, qui est le meilleur obtenu jusqu'à présent. On peut voir sur le plot des composantes principales (ci-dessous) que la réduction de variables a globalement bien fonctionné et que chacune des composantes permet d'améliorer la classification. On peut constater par exemple que le duo de composantes 1 et 4 permettent de séparer clairement les expressions "joie" et "peur".



- **QDA :**

  La QDA nous a donné un donné un résultat de 33.8% d'erreur sur le dataset réduit par FDA (la QDA n'étant pas applicable sur le dataset initial à cause du trop grand nombre de variables). C'est moins bien que la plupart des classifieurs utilisés jusqu'à présent

- **KNN :**

  La méthode des k plus proches voisins souffre d'une faiblesse sur les jeux de données où le nombre de prédicteurs est grand par rapport au nombre de variables. Ainsi son utilisation n'est valorisable dans notre cas que sur le dataset réduit.

  Sur les datasets réduits par FDA (que l'on suppose être les meilleurs pour notre cas car améliorant les frontières entre les classes), le taux d'erreur est supérieur aux autres classifieurs (28.9% pour le dataset PCA + FDA) et la variance est extrêmement élevée, due au caractère aléatoire de l'initialisation de l'algorithme.

  Les KNN ne sont donc pas un bon classifieur pour ce jeu de données.

- **Classifieur Bayésien Naïf :**

  Enfin, le classifieur Bayésien naïf nous a donné un taux d'erreur moyen de 22.2% sur le dataset réduit par FDA, ce qui est raisonnable.

3.4. Résultats

Voici un tableau illustrant les différents résultats obtenus en appliquant les classifieurs sur le jeu de données initial ou transformé :

Dataset	SVM				LDA	QDA	RF	Naive Bayes	CNN	MLP	KNN
	linéaire	poly	sigmoid	radial							
Initial	NC	NC	NC	NC	NC	NC	0.367	NC	0.35	NA	NC
Nettoyé	0.28	NC	NC	NC	0.211	NA	0.354	0.276	NA	NA	NC
FDA	0.232	0.321	0.278	0.316	NA	0.338	0.293	0.222	NA	0.230	0.331
PCA	0.285	NC	0.287	NC	0.240	NA	0.368	0.394	NA	0.301	NC
PCA + FDA	0.257	0.326	0.226	0.250	NA	0.305	0.263	0.274	NA	0.261	0.289
Demi-matrice brute	NC	NC	NC	NC	NC	NC	NC	NC	0.363	NC	NC
Demi-matrice nettoyée	0.284	0.336	0.273	0.286	0.247	NA	0.374	0.274	NA	NA	NC
Demi matrice + FDA	0.265	0.317	0.264	0.324	NA	0.345	0.308	0.253	NA	0.253	NC
Sobel nettoyé	NC	NC	NC	NC	0.248	NA	0.361	0.294	NA	NA	NC
Sobel + FDA	0.257	0.312	0.260	0.318	NA	0.339	0.304	0.244	NA	0.247	NC

Nous avons donc 4 classifieurs avec des performances similaires, la LDA, le SVM linéaire sur dataset FDA, le MLP sur dataset FDA et le classifieur Bayésien Naïf également sur dataset FDA. La LDA semble être le meilleur choix dans notre cas car elle possède la variance la plus faible et des performances très légèrement meilleures.

Malheureusement le CNN n'a pas permis d'obtenir les résultats escomptés, ainsi que chacun de nos datasets personnalisés (demi-matrice, contours, PCA et PCA+FDA)

3.5. Conclusion

C'est donc l'analyse discriminante linéaire sur le dataset original scalé et vidé de ses pixels noirs qui semble être le meilleur classifieur pour le jeu de données expressions. Nous aurions sans doute pu travailler plus sur la partie pré-processing des données, par exemple en appliquant des méthodes de détection de features directement, ou en supprimant plus de pixels (notamment ceux dont la variance inter-classe était très faible).

Si nous avions eu plus de temps ou des machines plus puissantes, nous aurions pu travailler plus sur le CNN qui reste à notre avis la meilleure solution pour les problématiques de traitement d'image de ce type.

4. Jeu de données Charactères

4.1. Analyse préparatoire

Le jeu de données caractères contient 10000 individus avec 16 prédicteurs quantitatifs et une variable qualitative à expliquer possédant un total de 26 classes (une pour chaque lettre de l'alphabet). Initialement créé à partir d'images (de hand-writing probablement), ce dataset bénéficie déjà d'une réduction de dimensions, ce qui va en faciliter le traitement.

De plus son grand nombre d'individus par rapport au nombre de prédicteurs rassure par rapport à la performance des classifieurs que nous allons pouvoir réaliser.

4.2. Nettoyage des données

Pour ce dataset, il n'est pas nécessaire d'effectuer un pré-traitement approfondi, mais nous allons tout de même scaler les données et utiliser un deuxième dataset réduit FDA sur les données pour constater s'il est tout de même possible d'améliorer les résultats par rapport au jeu de données brut.

4.3. Utilisation des classifieurs

Pour ce jeu de données, nous avons décidé d'utiliser d'abord les classifieurs classiques tels que la LDA, la QDA, le classifieur bayésien naïf, puis les random forests et enfin un perceptron multi-couches.

- **LDA :**

  La LDA est le premier classifieur que nous avons testé sur ce dataset. Nous avons obtenu un taux d'erreur moyen de 30.1%, ce qui est très mauvais pour un dataset de cette taille et aussi propre.

- **QDA :**

  La QDA a cette fois été beaucoup plus performante que la LDA et nous avons obtenu un taux d'erreur de 12.2% à la fois sur les données brutes et sur le dataset FDA.

- **Classifieur bayésien Naïf :**

  Le classifieur bayésien naïf a été globalement très mauvais, tout comme la LDA, et affiche un taux d'erreur moyen de 36.2% sur les données brutes et 29.5% sur le dataset FDA.

- **Random Forests :**

  Nous avons utilisé l'algorithme des random forests sur nos datasets, et à notre surprise nous avons obtenu les meilleurs résultats sur le dataset brut plutôt que sur le dataset avec réduction de dimension, avec comme paramètres mtry = 4 (soit la racine carrée du nombre de variables du dataset) et ntree =1000. Le taux d'erreur moyen obtenu est très faible (5.2%)

- **SVM :**

  Étant donné la taille du dataset, il nous a été impossible d'utiliser la fonction tune.out du package e1071 pour paramétrer le SVM. Néanmoins, en utilisant une fonction personnelle pour effectuer un grid search des hyper-paramètres du SVM, nous n'obtenons que des résultats relativement mauvais pour nos dataset.

- **Les K plus proches voisins :**

  L'algorithme des k plus proche voisins nous a permis d'obtenir des résultats très satisfaisants appliqué sur les données brutes et sur le dataset FDA. Le nombre de voisin optimal Kopt a été choisi à l'aide de la méthode du coude, nous avons obtenu un taux d'erreur de 9,5% sur les données initiales et un taux d'erreur de 7,4% sur le dataset FDA avec 3 comme valeur pour le k optimal.

- **Perceptron multi-couches :**

  Enfin, le perceptron multi-couches nous a donné des résultats honorables, proches de ceux de la random forest. Utilisé sur le dataset réduit par FDA, vec 500 neurones dans la couche cachée et 26 dans la couche de sortie, en 30 rounds, avec un batch-size de 10 et un learning rate de 0.01, nous arrivons à un taux d'erreur moyen de 5.5%

4.4. Résultats

Voici un tableau illustrant les différents résultats (taux d'erreur) obtenus en appliquant les classifieurs sur le jeu de données brut ou transformé :

Dataset	SVM				LDA	QDA	Random Forest	Naive Bayes	CNN	MLP	KNN
	linéaire	poly	sigmoid	radial							
Initial	0.156	0.206	NC	NC	0.301	0.123	0.052	0.362	NA	0.066	0.095
FDA	0.191	0.089	0.278	0.237	NA	0.123	0.129	0.29	NA	0.055	0.074

Les résultats obtenus sur ce dataset sont d'assez bonne qualité, en particulier pour la random forest, le perceptron multi-couches et les k plus proches voisins.

4.5. Conclusion

Étant donné que la random forest donne des performances légérement meilleures pour une variance identique au perceptron multi-couche, c'est ce classifieur que nous allons conserver pour faire les prédictions sur ce dataset.

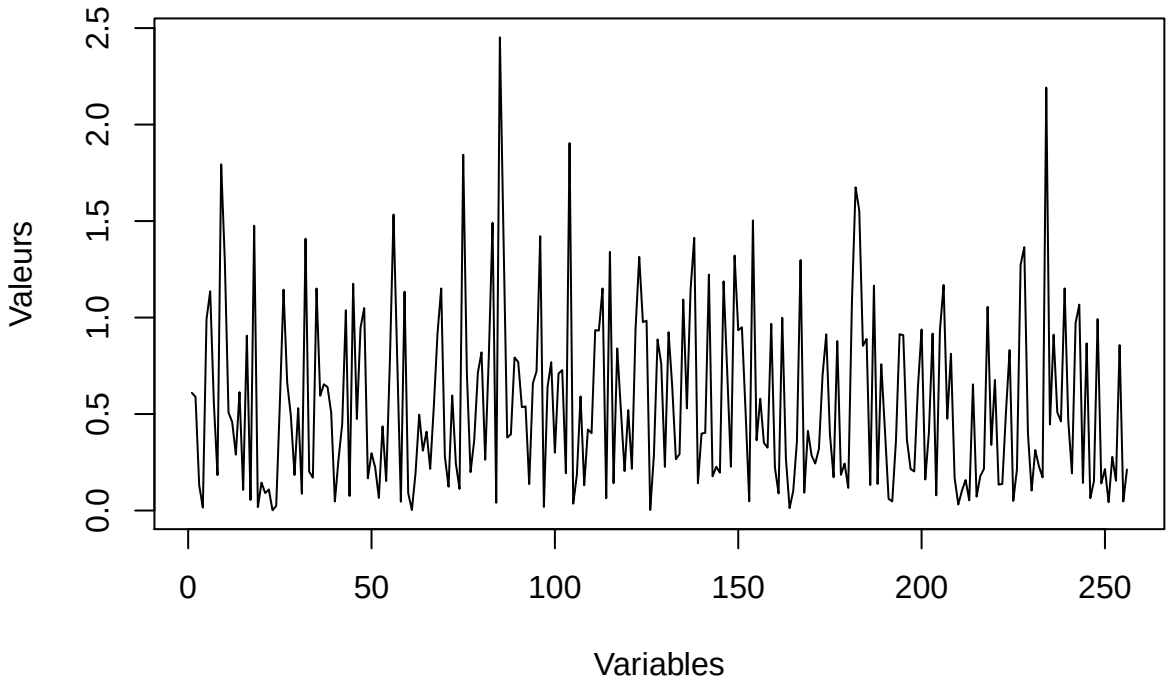
5. Jeu de données Parole

5.1. Analyse préparatoite

Ce jeu de données contient 2250 individus pour 256 prédicteurs et une variable à expliquer à 5 modalités. Ce dataset représente le signal sonore émis par une personne lorsqu'elle prononce une syllabe (parmi 5 possibilités).

Étant donné le grand nombre de prédicteurs, nous allons probablement encore devoir utiliser une réduction dimensionnelle pour améliorer les performances de notre classifieur.

En plottant le signal, on peut remarquer des différences visuelles entre les différentes modalités de la variable à expliquer. Ces différences visuelles nous ont fait penser à la possibilité d'utiliser un CNN en plus des autres classifieurs.



5.2. Nettoyage des données

Pour le prétraitement des données de ce jeu de données, nous avons encore une fois commencé par créer un nouveau dataset scalé et réduit par FDA, et un autre dataset pour le réseau de neurones à convolution, en disposant le signal sous forme de matrice d'une largeur faible et égale à un multiple de la période du signal (dans notre cas nous avons essayé les largeurs 8,9 et 10).

Enfin, nous avons également essayé pour le plaisir un dataset contenant les valeurs absolues du signal et un contenant les moyennes du signal, qui se sont tous deux avérés rapidement non concluants.

5.3. Utilisation des classifieurs

- **CNN :**

  Nous avons tout d'abord voulu essayer une méthode un peu exotique et ainsi nous avons testé d'utiliser un réseau de neurones convolutif sur notre matrice du signal en utilisant une fenêtre de convolution très grande.

  Malheureusement, cette tentative ne fut pas concluante car la fréquence d'enregistrement du signal ne doit pas être exactement un multiple de la fréquence du signal ou ne doit pas respecter le théorème de Shannon. Les résultats n'étaient donc pas concluants.

  Nous avons donc repris les deux autres datasets et essayé les classifieurs classiques.

- **LDA :**

  La LDA nous donne un très bon résultat sur ce dataset, avec un taux d'erreur de seulement 8.2%.

- **QDA :**

  La QDA, quant à elle, fournit des résultats plutôt médiocres sur le jeu de données brutes avec 36.4% d'erreur mais seulement 8.3% sur le dataset réduit par FDA.

- **Classifieur bayésien Naïf :**

  Le classifieur Bayésien Naïf donne des résultats honorables sur ce dataset avec 12.3% d'erreur sur les données brutes et 8.3% sur les données FDA.

- **Random Forests :**

  Les random forests donnent elles aussi de très bons résultats sur les deux datasets, avec pour plus faible taux d'erreur 8.2% sur les données brutes tout en ayant une variance également très faible (8*10e-7).

- **SVM :**

  Les SVM s'avèrent donner des résultats encore meilleurs que les autres classifieurs, avec entre 7.4 et 7.5% d'erreur sur les données brutes pour les noyaux sigmoid, radial et linéaire, et légèrement plus pour le dataset FDA (entre 7.9% et 8.1%).

- **Les K plus proches voisins :**

  L'algorithme des k plus proche voisins nous a permis d'obtenir des résultats très satisfaisants appliqué sur les données brutes et sur le dataset FDA. Le nombre de voisin optimal Kopt a été choisi à l'aide de la méthode du coude, nous avons obtenu un taux d'erreur de 9,6% sur les données initiales et un taux d'erreur de 8,4% sur le dataset FDA avec 15 comme valeur pour le k optimal.

- **Perceptron multi-couches :**

  Enfin, le dernier classifieur testé est le perceptron multi-couches, qui donne encore une fois de très bons résultats, mais légèrement moins que les SVM (8.3% d'erreur sur les deux datasets).

5.4. Résultats

Voici un tableau illustrant les différents résultats (taux d'erreur) obtenus en appliquant les classifieurs sur le jeu de données brut ou transformé :

Dataset	SVM				LDA	QDA	Random Forest	Naive Bayes	CNN	MLP	KNN
	linéaire	poly	sigmoid	radial							
Initial	0.074	0.451	0.075	0.074	0.082	0.364	0.082	0.123	NA	0.083	0.096
FDA	0.081	0.086	0.079	0.081	NA	0.083	0.085	0.083	NA	0.083	0.084

On peut constater que tous les classifieurs testés ont donné des résultats honorables sur au moins un des deux datasets, et qu'il est difficile de les départager les uns des autres.

5.5. Conclusion

Pour ce dernier dataset, nous allons nous tourner vers le classifieur SVM avec noyau sigmoid sur données brutes, avec comme paramétres cost=50 et gamma =0.0001. Les SVM à noyau linéaire et radial donnaient un taux d'erreur très légèrement inférieur mais leur variance est deux fois plus élevée que celle du SVM à noyau sigmoid, ce qui appuie notre décision finale.

6. Difficultés rencontrées

Ce projet fut particulièrement complexe à gérer, notamment pour la partie réseaux de neurones et nettoyage des données, de plus il fut extrêmement chronophage et gourmand en ressources matérielles.

Malgré des dizaines d'heures passées nous n'avons pas pu optimiser au maximum nos fonctions et hyper-paramètres, bien que nous soyons tout de même satisfaits des résultats obtenus. Nous avons effectivement dû faire des choix stratégiques d'optimisation car il nous était bien évidemment impossible de tester toutes les combinaisons possibles sur nos machines.

7. Conclusion

Ce TP, clôturant l'UV SY19, fut particulièrement enrichissant. Il nous a permis de mettre en application et d'approfondir nos connaissances sur les classifieurs utilisés en machine learning et en deep learning, notamment les réseaux de neurones qui sont aujourd'hui des outils indispensables du data scientist.

L'hétérogénéité des datasets proposés nous a permis de rencontrer les difficultés typiques du métier et de nous familiariser encore plus avec R, notamment pour la partie pré-processing des données et la formation autonome via l'utilisation de nouveaux packages, ainsi que pour la partie réflexion sur la cohérence des modèles et gestion du temps.