SY19 A18 TP7 Rapport

HAOJIE LU, JIACHENG ZHOU et HAIFEI ZHANG

8 Janvier,2019

1.Partie de classification

1.1 Introduction

Dans ce problème de classification, nous devons classifier les 3 différents types d'objets astronomiques à partir de ses variables. Pour cette partie nous avons fait les analyses ci-dessous :

-PCA

-K plus proche voisins

-Analyse discriminante linéarie (LDA) + Subset selection

-Analyse discriminante quadratique (QDA) + Subset selection

-Subset selection + Logistic Regression

-Classification naive bayésienne

-Arbre de décision avec application du bagging et des forêts aléatoires

-SVM et KSVM

1.2 Préparation

TOut d'abord, nous fasions le nettoyage de données.Etant que la colonne *rerun* et la colonne *objid* sont identiques pour tous les enregistements, nous supprimons ces deux colonnes.Pour pouvoir utiliser plus facilement dans la suite, nous avons séparé les données en deux parties : Un ensemble d'apprentissage (2/3 des données) et un ensemble (2/3 des données) de test. Egalement, on a choisi K=10 pour la validation croisée K-fold.

1.3 Sélection de modèles

PCA

Premièrement, nous faisons l'analyse en composant principal pour mieux analyser les données. Mais d'après l'analyse, tous les composants nous semblent importants.

KNN

Pour la méthode de KNN, nous retiendrons la classe la plus représentée parmi les k sorties associées aux k entrées les proches de la nouvelle entrée x. Premièrement, nous appliquons la méthode de la validation croisée pour obtenir un meilleur k. Par la méthode CV, le taux d'erreur reste presque invariant pour k entre 1 et 500. Donc, on choisissons alors K=100.

```
## Error knn = 0.2003599
```

LDA + subset selection

Nous faisons d'abord une sélection d'un sous-ensemble de variables en utilisant la fonction *Stepclass ()* de mainière "backward" dans le package klaR.Puis nous utilisons la fonction *lda()* dans le package *MASS* pour analyser linéariement les données que nous avons choisi.La moyenne des erreurs par 10-CV est ce que nous avons besoin pour la comparaison.

```
## method      : lda
## final model : class ~ u + r + i + z + run + camcol + specobjid + redshift +
##     plate
## <environment: 0x000000002ae335a0>
##
## correctness rate = 0.9326
```

```
## Error lda = 0.0688
```

QDA + Subset selection

Puis nous utilisons le modèle QDA par la fonction *qda()* dans le package *MASS* .Nous aussi faisons la sélection de prédicateurs. On trouve que il fonctionne beaucoup mieux que LDA.

```
## method      : qda
## final model : class ~ dec + u + g + run + camcol + field + redshift + fiberid
## <environment: 0x0000000025976b40>
##
## correctness rate = 0.9898
```

```
## error qda = 0.0108
```

Naive Bayes classifier

Pour cette méthode, nous faisons la modélisation en utilisant la fonction *NaiveBayes()* dans le package *E1071*. Nous calculons ensuite des taux d'erreur et la moyenne par 10-CV.

```
## error nbc = 0.0654
```

Subset selection + Logistic Regression

Dans cette méthode, nous faisons d'abord une sélection des variables. Car c'est une clarification de 3 types et donc nous utilisons la fonction *multinom()* de package **nnet*.

```
## final model : class ~ ra + u + r + i + z + run + redshift + mjd
```

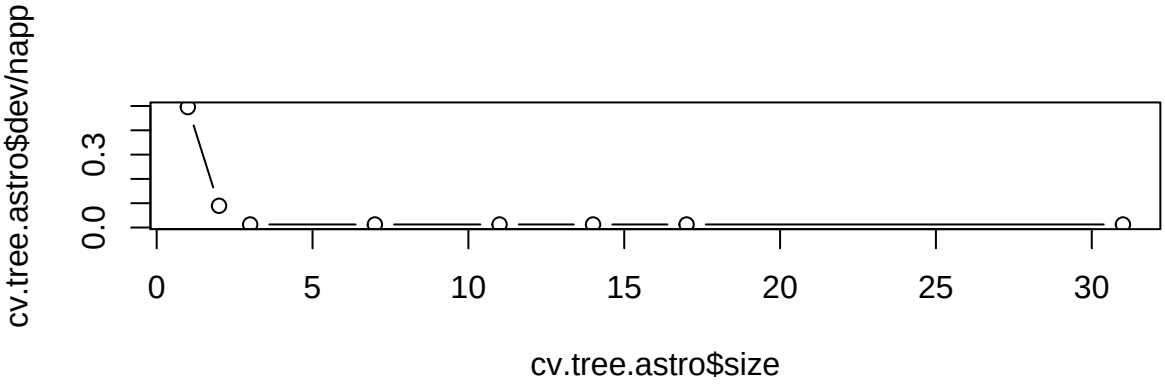
```
## error logr = 0.0089982
```

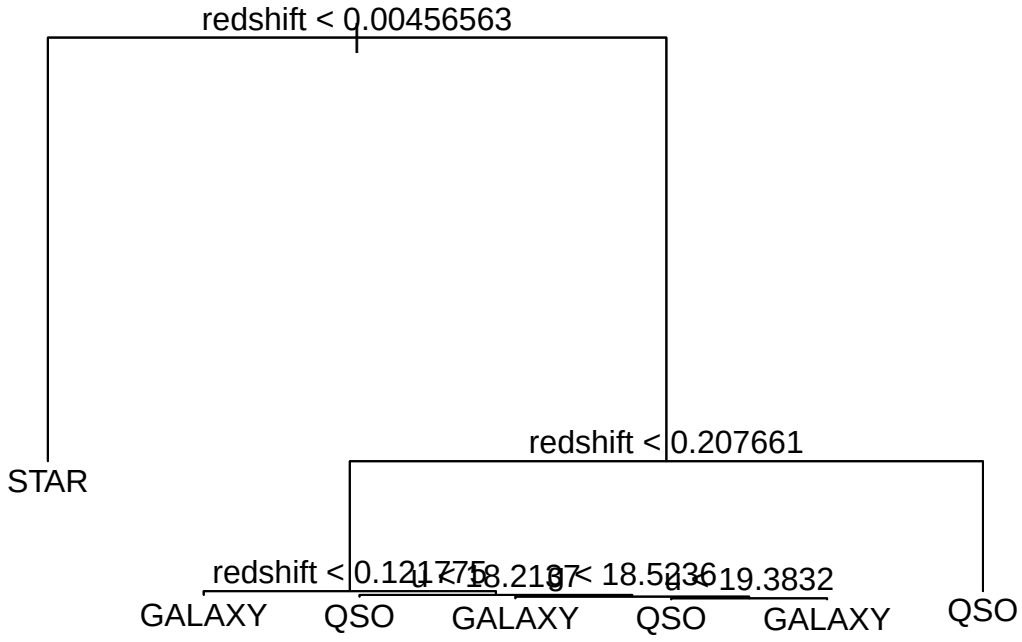
Decision Tree

Premièrement, nous construissons un arbre de décision sur ces données et nous calculons une erreur sur l'arbre original.

```
## error decision tree = 0.01019796
```

Puis nous appliquons à l'arbre précédent la procédure d'élagage, nous obtienons que l'erreur diminue un peu.






```
## Error of pruned tree with 4 branches = 0.01079784
```

Alors nous appliquons le bagging.

```
## error  bagging tree = 0.01259748
```

Nous changons le paramètre mtry = 4 et nous appliquons les forêts aléatoires sur ces données.

```
## error random Forest =  0.01259748
```

svm

Premièrement,nous utilisons la méthode SVM sans noyau.

```
## Le taux d'erreur du SVM sans noyau :  0.08278344
```

Nous utilisons la cross-validation pour trouver un meilleur cost paramètre C. Ca me fait très longtemps pour finir cette opération. Nous obtienons que le best_C = 100. Ensuite, nous essayons plusieur kenerls pour savoir s'il y a d'amélioration. Voici le tableau ci-dessous :

```
##                    linear polynomial radial sigmoid
## C-classification   0.0174     0.0784 0.0518  0.1858
## one-classification 0.5174     0.7790 0.6736  0.7758
```

Nous trouvons que le type='C-classification'avec kernel='linear' rende le moindre des erreurs.

```
## error svm with linear kernel = 0.0174
```

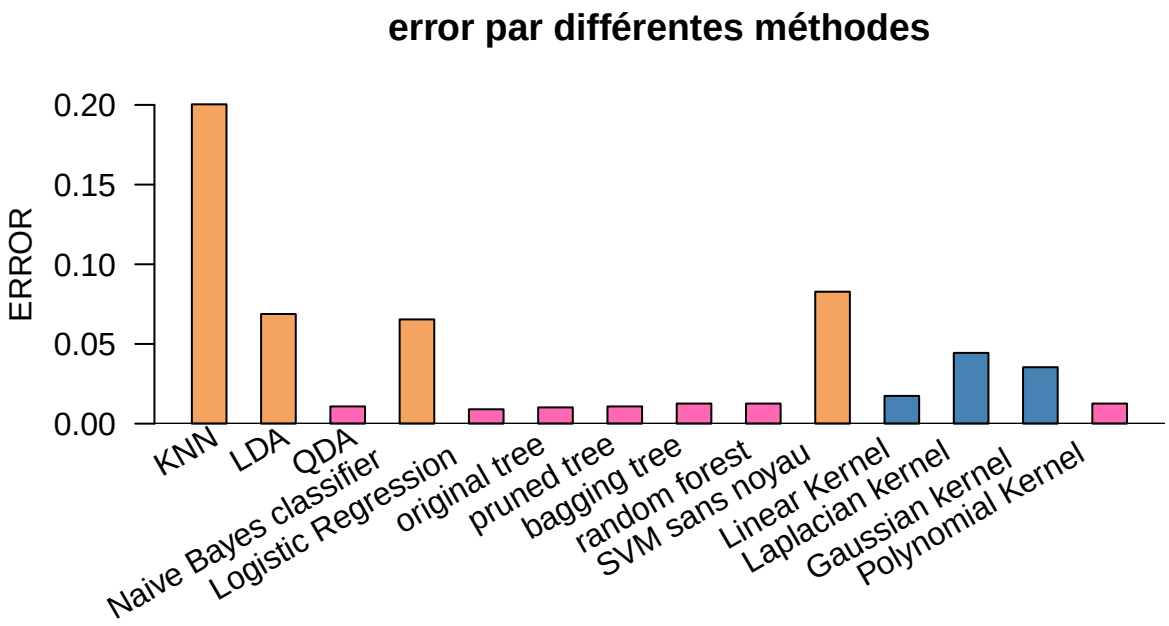
Ensuite, nous aussi utilisons la fonction *ksvm()* de package *kernelab* pour les refaire.

```
## error_ksvm_Laplacian =  0.04439112
```

```
## error_ksvm_Gaussian = 0.03539292
```

```
## error_ksvm_Polynomial = 0.01259748
```

Conclusion



Par le barplot de tous les taux d'erreurs, nous pouvons trouver que la methode Régression Logistique a la meilleure performance.

```
## La meilleure méthode :  Logistic Regression
```

```
##
## Son taux d'erreur :  0.0089982
```

2.Partie de regression

2.1 Introduction

Dans ce problème de régression, nous devons prédire la normalité du rendement de mais en France à partir des données climatiques fournies. *yield_anomaly* est la variable à prédire représentant l'anomalie de rendement de mais (une valeur positive indique un rendement plus élevé qu'attendu, une valeur négative indique une valeur perte de rendement par rapport à la valeur attendue), exprimée en tonne par ha. Pour cette partie nous avons fait les analyses ci-dessous :

-préparation des données

-sélection des modèles

-régression linéaire

-régression de Ridge, régression de Lasso et régression élastique

-régression polynomial et régression spline

-arbre de régression

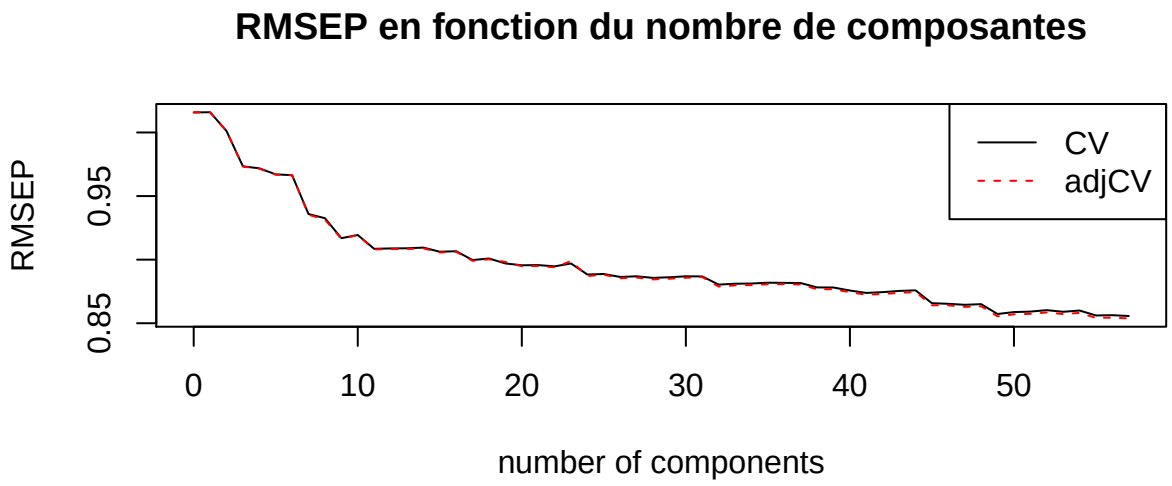
-SVR

2.2 Préparation des données

Pour pouvoir être utilisé plus facilement par la suite, nous avons séparé les données en détail : un ensemble d'apprentissage (2/3 des données) et un ensemble de test. Egalement, on a choisi K=10 comme un critère pour validation croisée K-fold.

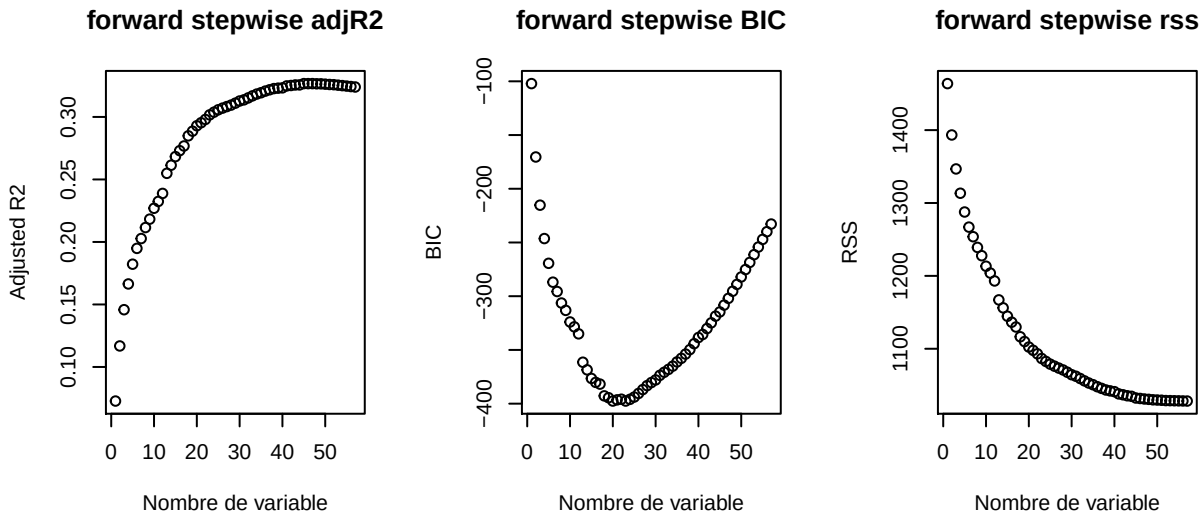
2.3 Sélection des modèles

Nous avons appliqué la méthode *PCA*, *subsets selection* pour analyser les modèles. Dans le cadre de *subsets selection*, nous n'avons pas utilisé la méthode *exhaustive* parce que 57 prédicteurs sont trop lourd pour cette meth-



ode.

Selon la graph, on peut voir que le meilleur modèle est de 55 ou de 57 variables. Après, on a utilisé la méthode de *subsets selection* avec *forward* et *backward*. On peut obtenir les critères *adjudant R2*, *BIC* et *rss*. Ici, on doit choisir le modèle du plus grand *adjusted R2*, le modèle du plus petit *BIC* et le modèle du plus petit *rss*.




```
## Nombre de variables pour le meilleur modèle par adjr2 :  45
```

```
## Nombre de variables pour le meilleur modèle par bic :  20
```

```
## Nombre de variables pour le meilleur modèle par rss :  57
```

En suite, on a fait la même chose en *backword stepwise*. Dans ce problème, nous avons choisi deux modeles : un modèle de toutes les 57 variables et autre de minimal BIC avec 18 variables.

2.4 Test de modèle

Régression linéaire

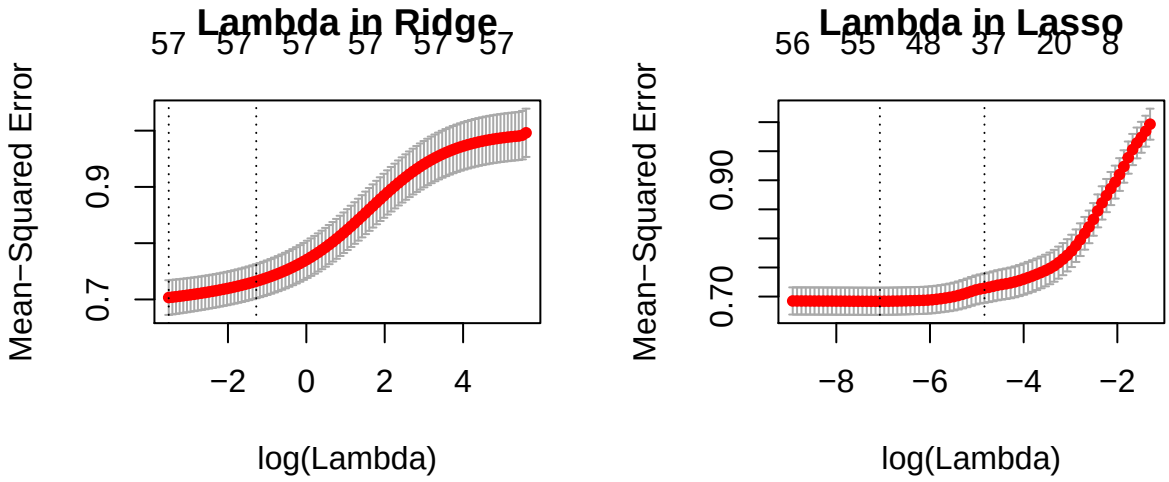
En utilisant la fonction *lm* avec les deux formulaire ci-dessus, on peut décider lequel modèles on doit choisir pour les autre méthodes ultérieures.

```
## MSE pour lm de 57 variables :  0.649683
```

```
## MSE pour lm de  20  variables :  0.6491643
```

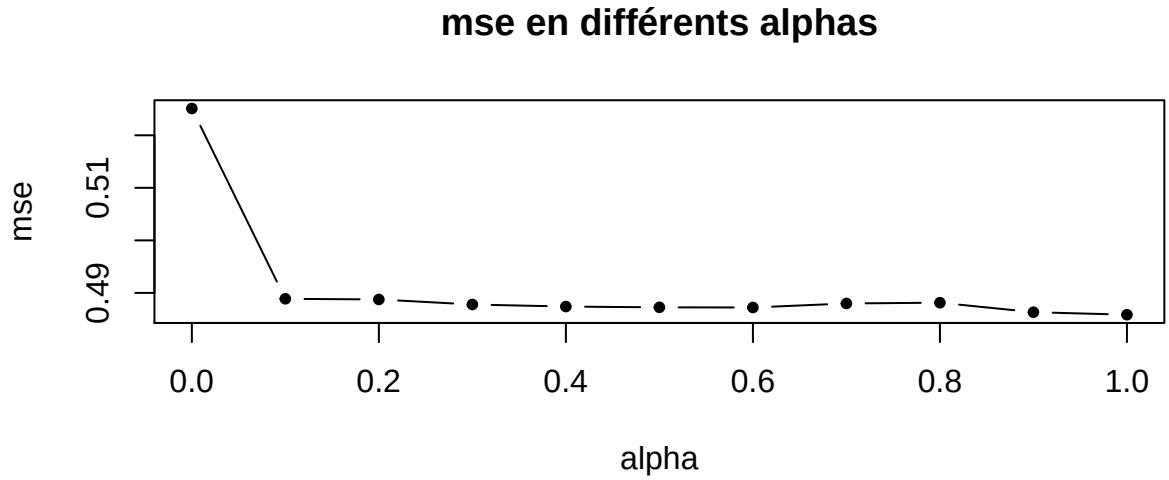
Ici, notre modèle est dérivé de l'ensemble de données d'apprentissage. La valeur de MSE moyenne est obtenue à partir du jeu de données de test. De toute évidence, les résultats montrent que l'utilisation de 57 variables est la meilleure.

Régression de Ridge, régression de Lasso et régression élastique



Sur le graphique ci-dessus, nous pouvons voir que la méthode de régression de ridge ne compresse pas les coefficients des variables à zéro. Avec la régression de lasso, lorsque ?? augmente, les coefficients de certaines variables moins importantes sont compressés à zéro. Par conséquent, la régression de lasso peut également être utilisée pour la sélection du modèle.

Pour la méthode régression de ridge, le paramètre alpha égale à zéro et pour régression de lasso est un. Mais on ne sait pas le comportement des valeurs d'alpha entre zéro et un. Nous avons appliqué la méthode de validation croisée pour déterminer la valeur d'alpha. C'est la méthode *régression élastique*.




```
## Le meilleur alpha :  1
## La mse minimale :  0.6456858
```

Régression polynomial et régression spline

Nous avons également essayé la régression polynomiale et la régression spline. Dans la régression par splines, nous avons appliqué *Natural Cubic Splines* et *B-Spline Basis*. Parmi ces méthodes, il existe un hyperparamètre *degree* ou *df* à déterminer. Par conséquent, nous avons utilisé une approche de validation croisée. Ci-dessous

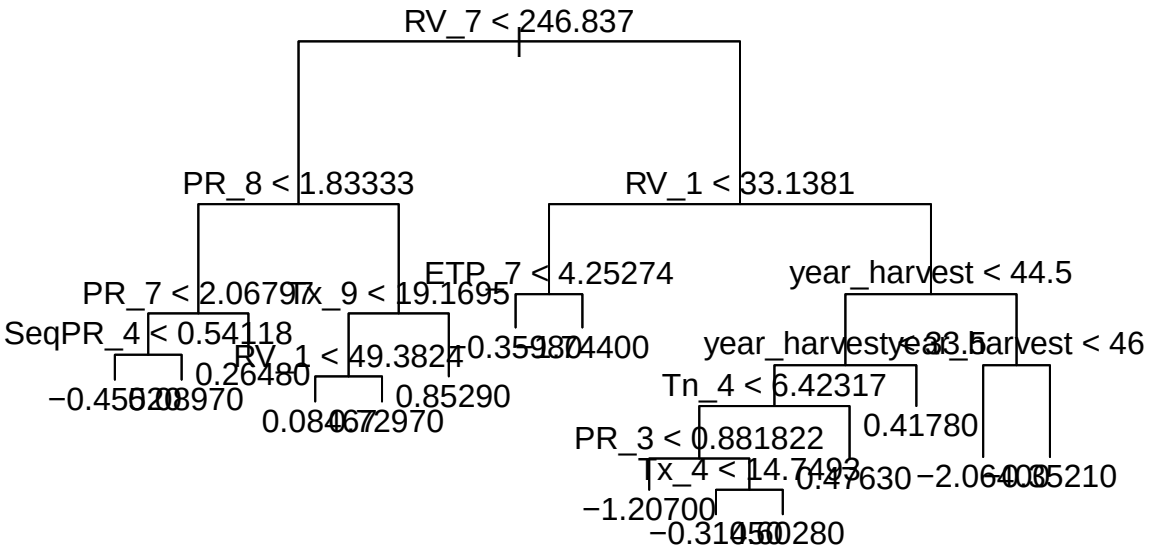
est le résultat que nous avons obtenu.

```
## La meilleure valeur de paramètre 'degree' pour régression polynomiale :  4
## La mse pour la régression polynomiale :  0.9188571
## La meilleure valeur de paramètre 'df' pour Natural Cubic Splines :  4
## La mse pour Natural Cubic Splines :  0.9146539
## La meilleure valeur de paramètre 'df' pour B-Spline Basis :  1
```

D'après les résultats, les effets de la régression polynomiale et de la régression spline ne sont pas bons.

Arbre de régression

Les méthodes arborescentes sont souvent utilisées pour classifier les problèmes, mais dans les problèmes de régression simples, l'arbres de régression constituent également une approche intéressante.




```
## La mse pour l'arbre par défaut :  0.7487649
## le meilleur nombre de feuille de l'arbre :  16
## La mse pour l'arbre pruine :  0.7487649
```

Regression de mixture

En utilisant la fonction *regmixEM* de la librairie *mixtools*, nous avons trouver le jeu de donnée est composé de deux composents principal. Avec les coefficients, on peut obtenir les predictions. Le resultat n'est pas mal.

```
## La mse de regmixEm :  1.11422
```

SVR

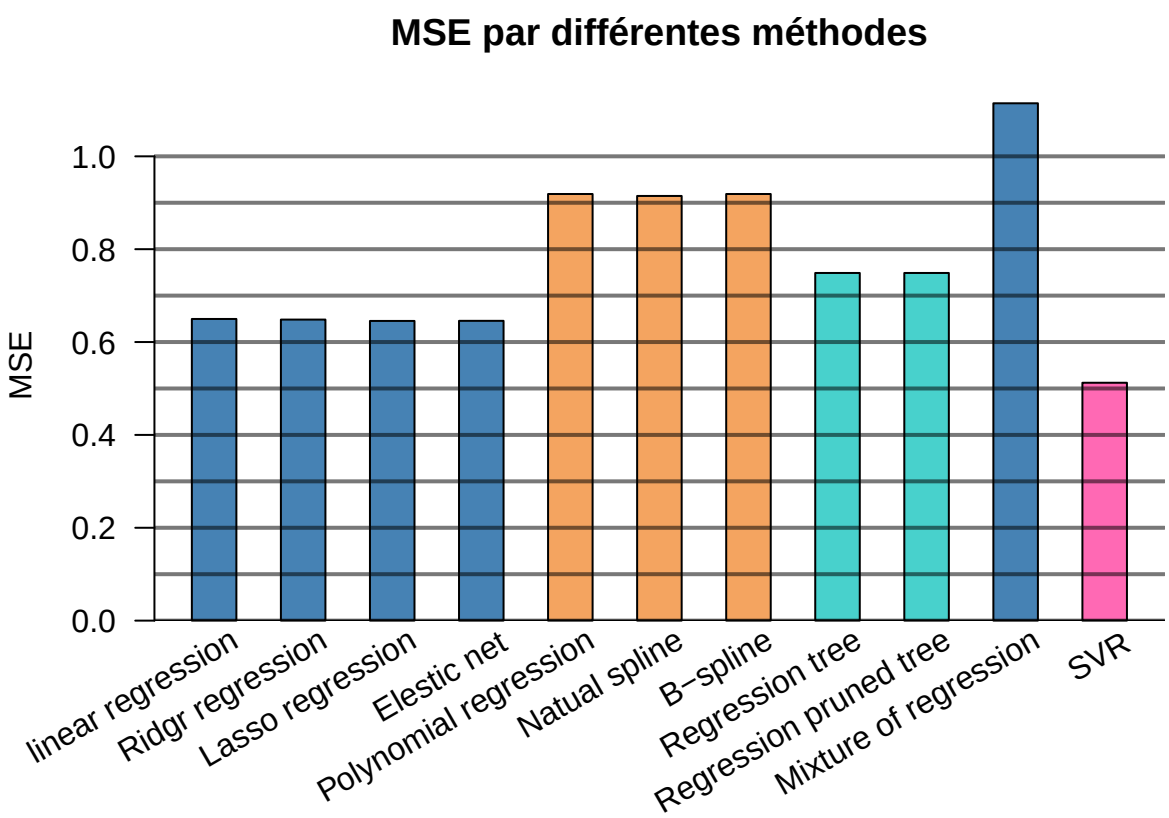
Enfin, nous avons utilisé la méthode SVR. Nous avons essayé plusieurs nouyau kernels : Gaussian, Laplacian et Polynomial.

Donc, nous utilisons le noyau de "laplacedot". Après, nous avons également utilisé une méthode de validation croisée pour déterminer le paramètre C parmis 0.01, 0.1, 1, 10,100 et 1000. Cette méthode prend beaucoup de temps à travailler, mais le résultat final est très bon.

```
## La mse de SVR :  0.5123366
```

2.5 Conclusion

Après nos exploitations, nous avons découvert ce que la méthode de *"SVR"* avec noyau de *"laplacedot"* possède le meilleure comportement pour ce problème. ci-dessous est la graph de MSE des différent méthode.



PARTIE 3 : classification d'images

Dans cette partie,on traite la classification d'images naturelles (au format JPEG) représentant des voitures,des chats et des fleurs. La tqche consiste à prédire le contenu de nouvelles images appartenant à l'un de ces trois types. Dans ce projet, on utilise deux méthodes classiques de Machine Learning et une méthode de réseaux de neurones.

3.1 Algorithme classique

Pré-traitement des données

Il est obligatoire de pré-traiter les donnés d'images car les algorithmes ne peuvent pas traiter les données en forme JPEG. Tout d'abord, on utilise la fonction *readImage()* dans la package "EBImage" qui permet de lire

les images et les transformer en forme de matrice avec 3 dimension *(hauteur x largeur x canal)*. Etant que les tailles de chaque image sont différents, on redimensionne la matrix en forme *(64 x 64 x 3)* par la fonction *resize()* de même package.

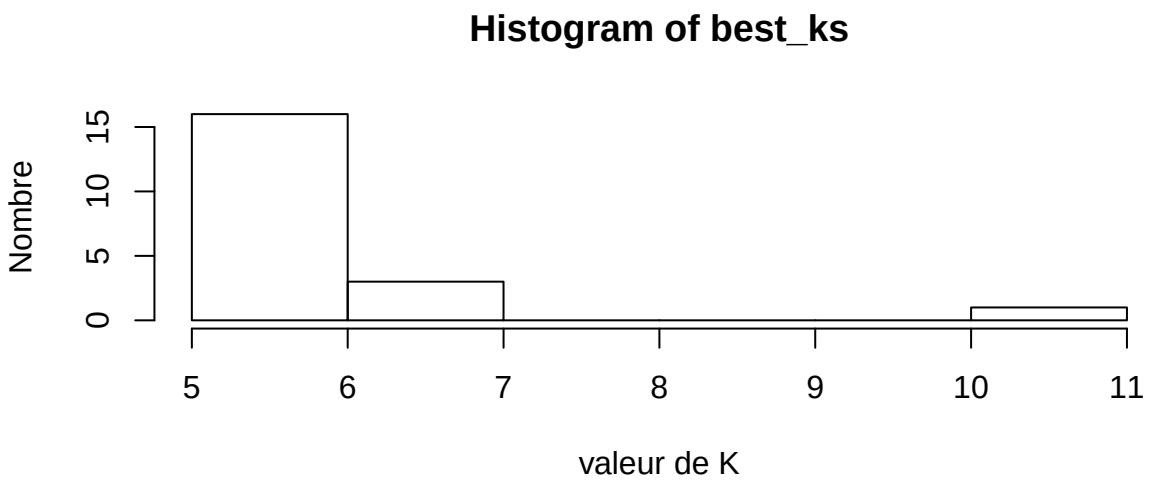
Mais les algorithmes traditionnels ne peuvent pas bien faire l'extraction de caractéristiques, donc on va faire manuellement et on utilise l'histogramme de gradient orienté(HOG). Son idée importante est que l'apparence et la forme locale d'un objet dans une image peuvent être décrites par la distribution de l'intensité du gradient ou la direction des contours. On utilise la fonction *HOG()* dans la package "OpenImageR" et on met cells=3 pourle nombre de division et orientations = 6 pour le nombre d'orientation. Donc, après cette opération, on peut obtenir un vecteur de length 54(3^2x6) avec les descripteurs HOG pour chaque image. A la fin,on assemble tous les vecteurs par *rbind()* pour obtenir une grande matrice qui garde les données de toute les image.

Maintenant, on peut faire notre modèle avec cette grande matrice. Ensuite, on va utilise deux méthodes classiques : KNN et SVM pour classifier les images.

Test des différents classifieurs

KNN

Pour l'algorithme KNN, un problème très important est du choix de paramètre K. Ici, on effectue une validation croisée 10-folds répétée 20 fois pour chercher le meilleur k.Le k=5 est le plus nombreux. Donc, on choisit k=5.



On aussi effectue une validation croisée 10-folds répétée 10 fois et le taux d'erreur moyen est :

```
## [1] 0.2145536
```

Comme on peut le remarquer, le pourcentage d'erreur obtenu est 21% ce qui reste élévé. Donc on fait un autre classifieur Support Vector Machine(SVM).

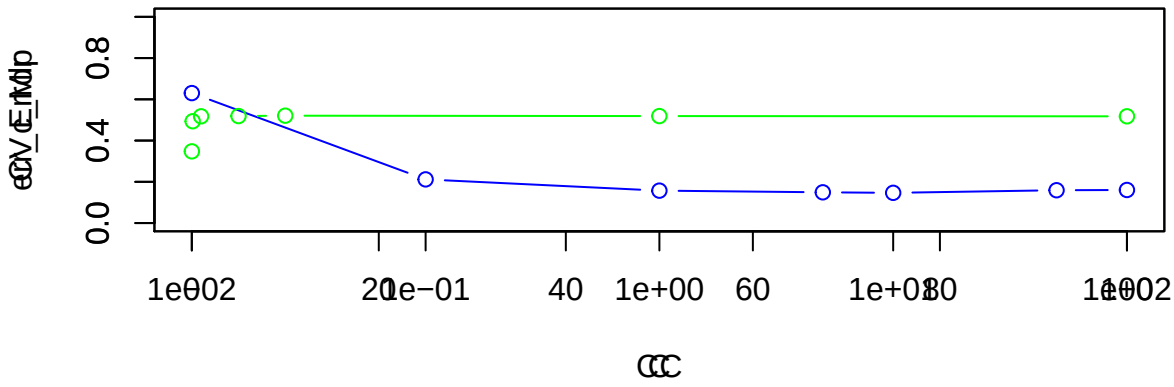
SVM et KSVM

On applique à présent SVM en effectuant une validation croisée en 10-folds. On utilise la fonction *svm()* dans le package *e1071*. Car notre projet est une classification de 3 types, il est important de metter la paramètre *probabilites= TRUE*.

```
## [1] 0.1515893
```

Le taux d'errur est environs 15% et il est meilleur que la méthode précédent. Mais on n'est pas encore satisfait pour cette résultat. Donc,On voudrais savoir si KSVM aura une meilleur preformance. On utilise la fonction *ksvm()* de package kernlab.

On cherche à trouver la meilleur fonction Kernel et on essaie 2 kernels les plus populaires : Gaussian et MLP kernel. On fait pas le kernerl polynomial car ça fait très long temps pour l'algotithme. Pour chaque kernel,on fait un choix sur le paramètre C. Voici la graphe ci_dessous :




```
## La meilleur kernel, C et le taux d'erreur : Gaussian, 10 ,  0.1465891
```

On peut trouve que la kernel MLP fait très mal. Mais la kernel Gaussian avec C=5 est un peu meilleur que svm et son taux d'erreur est envions 14.7%.

3.2 Réseaux de Neurones

Dans cette partie, on utilise la méthode CNN (Convolutional Neural Network). Tout d'abord, il faut lire les données et les transformer en en tenseurs/matrices 4D qui peut être reconnaît par CNN, de forme N x hauteur x largeur x canal . On aussi utilise les fonctions *readImage()* et *resize()* de package EBImage pour réaliser ça .La forme dans notre modèle est N x 32 x 32 x 3, où N est le nombre d'images. La méthode CNN permet de extraire des caractéristiques automatiquement, donc on n'a pas besoin de faiire ça manuellement comme la partie précédente.

On fait un modèle avec deux couches convolutionnelles de 2D masquée. La présentation de notre modèle CNN est dans ANNEXE.

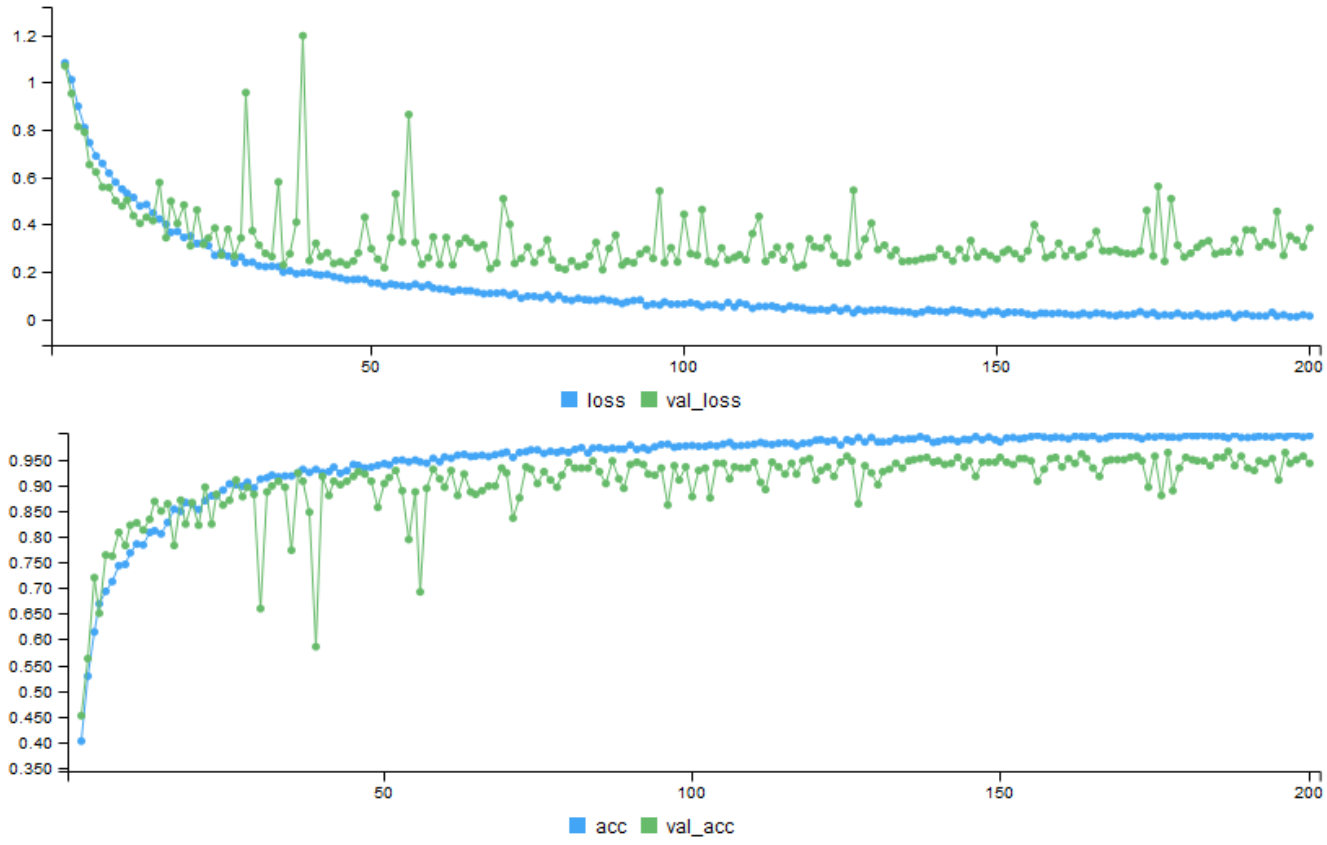
En plus, on met *epochs=200* et *batch_size=32*. On retire 20% d'images pour la validation,20% pour le test et la reste pour entraîner le modèle. Regardz la graphe générée pendant l'entraînement de modèle dans ANNEXE. La valeur perdue a une très bonne convergence, bien qu'il y a quelques petites vibrations. Le taux correct dans la validation atteins atteint à 96%. Mais sa performance dnas les données de test est ,qui est beaucoup pire que celle-ci de validation. C'est envrions 78%. Donc on choisit le HOG + SVM comme notre prédicteur final.

Annexe

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 32, 32, 32)	896
activation_1 (Activation)	(None, 32, 32, 32)	0
conv2d_2 (Conv2D)	(None, 30, 30, 32)	9248
activation_2 (Activation)	(None, 30, 30, 32)	0
max_pooling2d_1 (MaxPooling2D)	(None, 15, 15, 32)	0
dropout_1 (Dropout)	(None, 15, 15, 32)	0
conv2d_3 (Conv2D)	(None, 15, 15, 32)	9248
activation_3 (Activation)	(None, 15, 15, 32)	0
conv2d_4 (Conv2D)	(None, 13, 13, 32)	9248
activation_4 (Activation)	(None, 13, 13, 32)	0
max_pooling2d_2 (MaxPooling2D)	(None, 6, 6, 32)	0
dropout_2 (Dropout)	(None, 6, 6, 32)	0
flatten_1 (Flatten)	(None, 1152)	0
dense_1 (Dense)	(None, 512)	590336
activation_5 (Activation)	(None, 512)	0
dropout_3 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 3)	1539
activation_6 (Activation)	(None, 3)	0

Total params: 620,515
Trainable params: 620,515
Non-trainable params: 0

summary(model)



La trace des itérations de CNN