Rapport TP

PAIGNEAU Hugo - XIE Yijue - CAI Bolun

1. Classification

Nous avons analysé les données pour faire des classifications dans les données d'astronomie. Il y a 5000 observations avec 17 variables explicatives et la variable à classifier CLASS . Les catégories de la classification comprennent la galaxie, l'étoile et le quasar.

1.1 Analyse de données

Après observation, nous avons constaté que les données des galaxies et des étoiles représentaient près de la moitié des 5000 données, tandis que les données du quasar n'en représentaient que 8%.

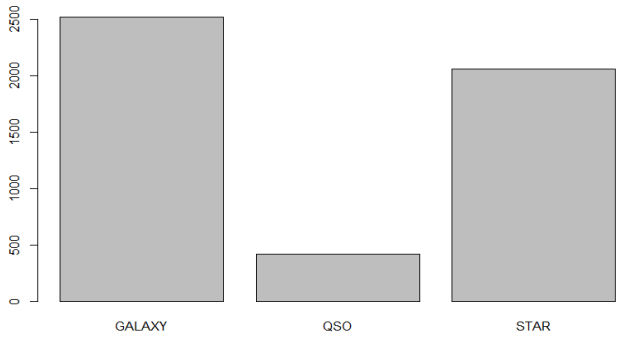


Fig1 : Répartition des 3 classe en nombre de lignes sur la dataset

De plus, nous avons également observé certains attributs de la classe id, qui n'ont aucun effet sur notre classificateur. Nous les supprimons alors, y compris : rerun, run, camcol, champ, objid, specobjid. Afin d'analyser la corrélation de toutes les données, nous utilisons la fonction cor dans le package corrplot pour calculer facilement et rapidement le coefficient de corrélation entre les variables continues et l'exprimer dans l'image. Nous pouvons voir sur la figure 2 que ces données sont très linéairement corrélées. Ainsi, à l'avenir, nous supprimerons l'association linéaire entre les données, comme les méthodes de PCA ou d'un subset selection, etc.

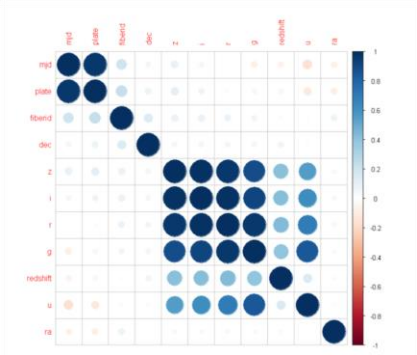


Fig2 : Graphique de corrélation entre les variables à prédire

1.2 Subset selection

Comme il n'y a que 14 degrés de variable, vous pouvez utiliser le mode exhaustif pour rendre les résultats de selection de subset plus précis. Nous utilisons R2 et BIC, et combiner ces deux résultats dans une matrice avec les variables d'origine sans subset sélection. Chaque fois que nous effectuons une analyse de classification, nous utilisons ces trois modèles dans différentes situations pour comparer la prédiction le plus précise dans les mêmes conditions.

```
[[1]]
class ~ .

[[2]]
class ~ dec + u + g + r + i + z + redshift + plate + mjd + fiberid
<environment: 0x0000000017724520>

[[3]]
class ~ ra + dec + u + g + r + i + z + redshift + plate + mjd +
    fiberid
<environment: 0x00000000174151a8>
```

Fig3 : Meilleurs modèles

Après les tests, nous avons constaté que les données d'origine étaient plus précises. On peut conclure que parce que les variables n'ont que 14 dimensions, lorsque nous supprimons certaines variables, cela conduira à trop peu de paramètres et le modèle ne peut pas apprendre suffisamment de formes de données complexes.

1.3 LDA, QDA, RDA, Bayes Naives

LDA s'utilise principalement quand on considère que les classes ont une matrice de covariance identique. Mais maintenant les données sont bien corrélés. On aura sans doute pas un bon taux d'erreur. Pareil pour Naive Bayes, il a un taux d'erreur de 3.6%.

Nous combinons ces quatre modèles avec RegSubset et KCV respectivement. Dans une 10-Fold validation croisée, nous utilisons trois données différentes après le subset pour les tests. L'erreur maximale est à près de 10% de RDA et l'erreur minimale est de 3,56% des Bayes Naives.

1.4 MDA

Le package mclust est un cluster basé sur un modèle statistique: c'est-à-dire que nous supposons que différents clusters proviennent de distributions différentes et que l'ensemble des données est une distribution de modèle mixte. Dans un classificateur supervisé multidimensionnel, nous pouvons utiliser une analyse discriminante basée sur les mélanges (MDA). En supposant que chaque classe est une distribution de mélange gaussien. Nous utilisons ici mclustda sans paramètre et l'algorithme MDA avec paramètres EDDA. Le premier correspond à une même matrice de covariance pour toutes les classes et ce dernier correspond l'analyse discriminante de décomposition des valeurs propres. Nous utilisons cvMclustDA pour tester toutes les données avec des catégories kcv égales à 10. Nous utilisons également la McLustDA plus générale. Chaque classe représentée par cette méthode utilise un mélange fini de distributions gaussiennes, et le nombre de composants et de matrices de covariance peut varier. Nous avons une erreur de 3,2% avec MclustDA normal et une erreur de 1,8% avec EDDA.

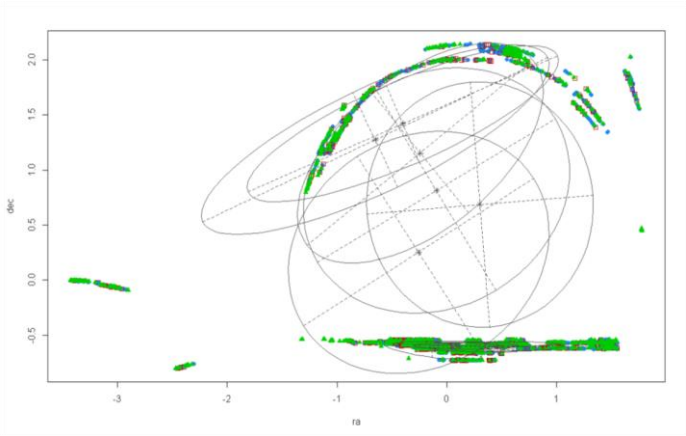


Fig4 : ScatterPlot entre les variables ra (ascension et declination)

Comme nous pouvons clairement l'observer, les coordonnées équatoriales ne diffèrent pas significativement entre les 3 classes. Il y a des valeurs aberrantes pour les étoiles et les galaxies, mais pour la plupart, les coordonnées sont dans la même plage.

1.5 PCA

Nous utilisons PCA pour réduire la corrélation entre les données. Nous avons constaté à partir de l'image du tracé que les cinq groupes de données u, g, r, i, z ont une corrélation élevée. Nous avons donc décidé d'effectuer un traitement de corrélation sur ces cinq groupes de données. Comme le montre la figure 5, trois variables peuvent couvrir plus de 90% du volume de données. Nous avons donc sélectionné trois nouvelles dimensions pour le fonctionnement ultérieur du SVM.

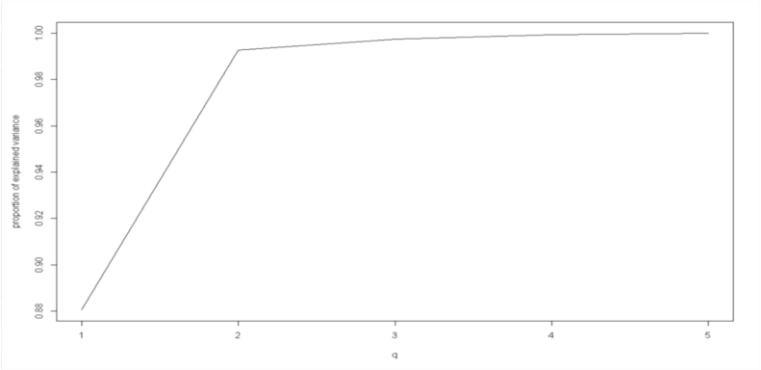


Fig5 : Proportion de la variance expliquée selon le nombre de composantes principales

1.6 Random forest et Bagging

Nous avons essayé des algorithmes utilisant random forest et bagging. Ce sont des méthodes plus interprétables que les autres dans le cas de classification. Nous avons effectué une analyse et une classification des arbres sur tous les paramètres et obtenu un bon résultat de 1%. De plus, nous essayons également de traiter les données en K-CV avec PCA. Prenez le paramètre mtry = sqrt (p) = 4, et l'erreur reste à 1%. Nous pouvons voir sur la figure 6 lorsque le nombre d'arbres dépasse 100, l'erreur commence à se stabiliser.

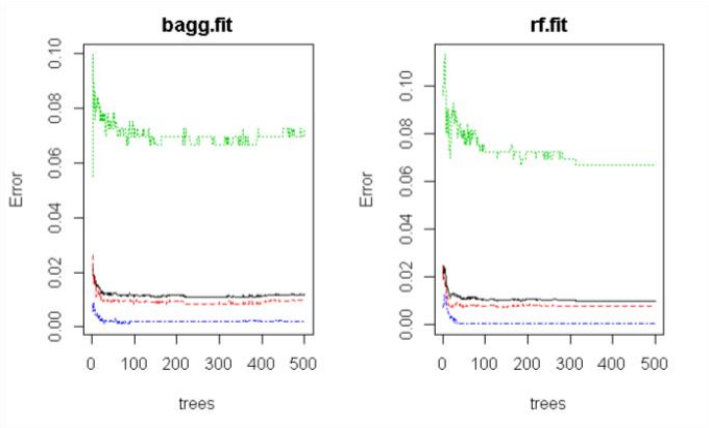


Fig6 : Out Of Bag error selon le nombre d'erreur pour les modèle de bagging et de random Forest

1.7 SVM

Dans le processus de mise à normalisation des données, nous avons des doutes que le SVM a sa propre propriété pour mettre à l'échelle les données. Est-ce la même chose que le processus de normalisation que nous voulons effectuer sur les données ? Après une certaine recherche, nous avons constaté que l'attribut d'échelle fourni par défaut peut être à nouveau valider de façon croisée dans notre validation croisée prédéfinie. L'attribut Scale dans SVM est une opération de boîte noire. Il peut effectuer un traitement non central ou non-scale sur certaines variables. Cela a plus de choix que notre traitement manuel de normalisation, et la précision des paramètres obtenus en imbriquant KCV dans KCV est également plus élevée.

Nous avons d'abord effectué une classification SVM avec noyau de vanilladot sur toutes les données, avec un taux d'erreur de 1,9%. Dans le cas du nombre de répétitions de la validation croisée est 10, nous avons d'abord déterminé qu'il fonctionne sous le noyau de polydot, rbfdot et vanilladot. Le résultat de vanilladot est meilleur, environ 1,4%. Après pour le coût de la violation des contraintes, sélectionnez un tableau de (10, 100, 1000, 5000, 10000) en appliquant les données transformées de PCA dans SVM. Trouvez une valeur optimale de c égal à 1000, moment auquel l'erreur est de 0,6%.

1.8 Conclusion

Nous représentons graphiquement toutes les erreurs de test. Dans la figure, nous pouvons conclure que bien que la forêt aléatoire ait de meilleures capacité d'interprétation, la boîte noire svm est plus efficace.

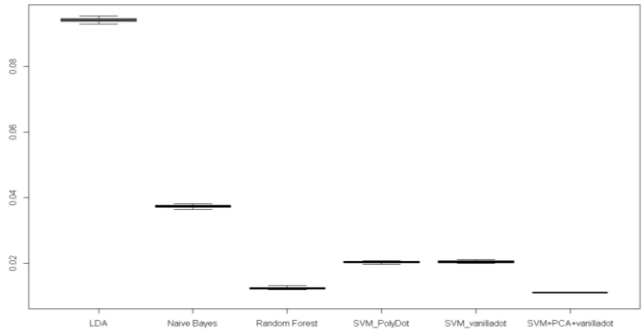
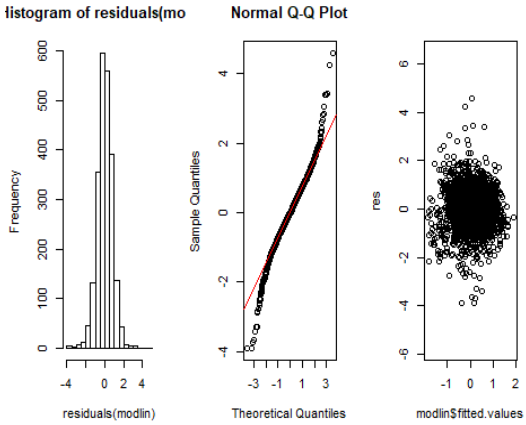


Fig7 : Récapitulatif du pourcentage d'erreur par modèle sous forme de boxplot

2 - Culture du maïs, la régression pour prédire l'anomalie du maïs

2.1 - Analyse de données

Les données concernent le rendement du maïs en France, dans les différents département sur plusieurs années. L'objectif est de prédire le rendement à partir de données climatiques. Il y a 2300 individus et 58 variables : 57 prédicteurs. On choisit dès à présent de supprimer la colonne X contenant un identifiant unique n'apportant pas d'informations particulières. On a p < n.



Les hypothèses du modèle lineaire ne semblent pas etre valable car les residus ne sont pas gaussiens (test de Shapiro-Wilk). Cependant dans le cas qu'un échantillon de grande taille, ce modèle reste robuste et peux convenir. C'est pour cela que nous le testons par la suite.

On sépare les données en 80% de données d'entraînements et 20% de données de test de manière aléatoire. Par la suite nous évaluerons le MSE, la variance et le biais par validation croisée afin de pouvoir comparer les modèles en eux.

2.2 - Selection de variable

On commencé par observer les corrélations entre mes variables explicatives et l'anomalie du mais. On a pu remarqué que certaines variables étaient fortement corrélées ! Nous avons utilisé la méthode Forward selection et obtenons donc 2 modèles (24 et 25 paramètres) qui viennent s'ajouter au modèle complet.

2.3 - Régularisation

On va utiliser la régularisation pour tenter d'aider le modèle à mieux généraliser sur les données de test. Sélection de la pénalité par validation croisée. La figure 8 illustre le biais et la variance. Imaginez que le centre rouge est le véritable paramètre de population que nous estimons, β, et les plans sont les valeurs de nos estimations résultant de quatre estimateurs différents - biais et variance faibles, biais et variance élevés, et leurs combinaisons.

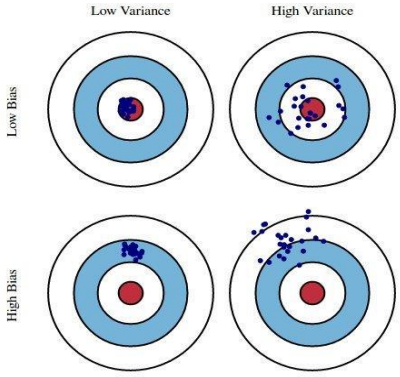


Fig8 : Trade-Off Biais-Variance

Le biais et la variance devraient être faibles, car des valeurs élevées entraînent de mauvaises prévisions du modèle.

On a donc essayé en utilisant Ridge, Lasso et Elastic Net Comme exemple, voyez Ridge:

$$Bias(\hat{\beta}_{ridge}) = -\lambda(X'X + \lambda I)^{-1}\beta,$$
$$Var(\hat{\beta}_{ridge}) = \sigma^2(X'X + \lambda I)^{-1}X'X(X'X + \lambda I)^{-1}.$$

Fig9 : Calcul de la variance et du biais pour la régularisation Ridge

De là, vous pouvez voir que lorsque λ devient plus grand, la variance diminue et le biais augmente. Il faut choisir judicieusement ce lambda : nous avons performé une Cross-Validation sur ce paramètre, le tout en essayant de minimiser l'AIC et le BIC. Voici un résumé des résultats :

Model	Variables	Ridge	Lasso	ElasticNet (alpha = 0.5)
1	Tous	0.704225	0.709299	0.702211
2	25	0.739872	0.744367	0.772829
3	24	0.757673	0.761229	0.788923

Nous obtenons des valeurs similaires pour les 3 méthodes.

2.5 - Réduction de dimension

Nous avons choisi ici la PCR. L'idée de base de la PCR est de calculer les principales composantes, puis d'utiliser certaines de ces composantes comme prédicteurs dans un modèle de régression linéaire ajusté en utilisant la procédure typique des moindres carrés. Cette méthodes est vraiment proche de la PCA

En utilisant la PCR, on découvrira peut-être que 5 ou 6 composantes principales sont suffisantes pour expliquer 90% de la variance de vos données. Dans ce cas, il est mieux d'exécuter la PCR avec ces 5 composants au lieu d'exécuter un modèle linéaire sur toutes les variables. Ici, on obtient un résultat peu satisfaisant puisqu'il faut utiliser les 57 prédicteurs.

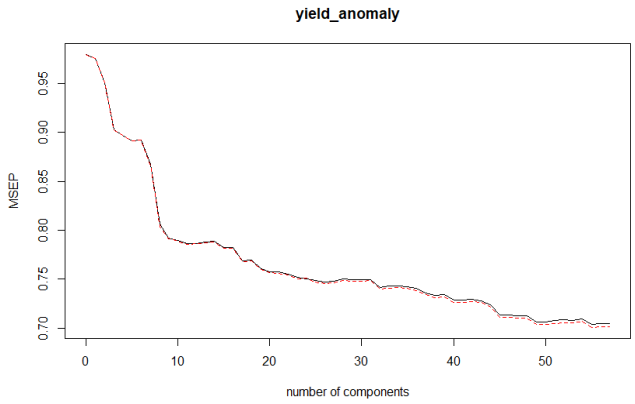


Fig10 : PCR sur les données entières

2.6 - Selection du modèle

2.6.1 - La régression linéaire simple

Il est toujours bon d'explorer ses données en utilisant un modèle facilement interprétable dès le début. Pour celà, la régression linéaire est l'outil idéal. On Utilise encore le K-Fold Cross-Validation sur chacun des 3 modèles différents. Nous obtenons un meilleur MSE pour le modèle complet encore une fois : 0.7022144. Nous avons aussi essayé de transformer nos variables en les combinant par elles-mêmes, en les multipliant par rapport aux autres, du même mois, et en transformant la variable du département en plusieurs variables binaires. Le MSE n'est que faiblement amélioré, nous passons à un MSE de 0.7012898.

On ne peut pas utiliser la regression polynomial car les temps de calculs seraient trop importants ici.

2.6.2 - Splines et GAM

Dans le cas d'un modèle multi-dimensionnel, nous n'utilisons les splines basiques, naturelles et smooth. On peut faire appel aux tensors mais la dimension augmente trop (curse of dimensionality) et on perd en interpretabilité. Pour palier ce soucis, nous utilisons les GAM (Generalized Additive Models). Ce modèle est bon pour expliquer mais reste souvent moins performant pour prédire. On obtient des résultats un peu plus intéressant avec le modèle complet encore une fois.

Model	GAM	Résultat
	DF	5
1	MSE	0.6797196
2	MSE	0.7336208
3	MSE	0.7573317

2.6.3 - RandomForest

Nous avons procédé de manière analogue à l'astronomie en créant un modèle simple, sans optimisation de paramètre d'abord, puis en optimisant le nombre d'arbre et gradant le mtry fixe (racine de 57). On calcul l'erreur grâce à la méthode vue en cours "out-of-bag error". Nous obtenons un nombre d'arbre égal à 960, puis nous optimisons ce mtry avec 960 arbres : 34.

Les scores obtenus sont moins bons que pour les GAM.

2.6.4 - SVM

Nous procédons aussi comme dans la partie précédent. Nous scalons nos données dans la fonction ksvm avec SCALE=TRUE pour normaliser nos données. On s'attend à un résultat "boite-noire". Nous choisissons aussi une double nested 10-CV pour faire varier nos paramètres Cost et obtenir de meilleures performances. Nous avons passé au peigne fin 4 noyaux : vanilladot, rbfdot, laplacian kernel et polydot.

Meilleur C	Kernel	Résultat
3000	vanilla	0.679719
1000	RBF	0.573627
50	poly(très instable)	0.884993
6	laplace	0.493328

On obtient de très bons résultats pour laplace :

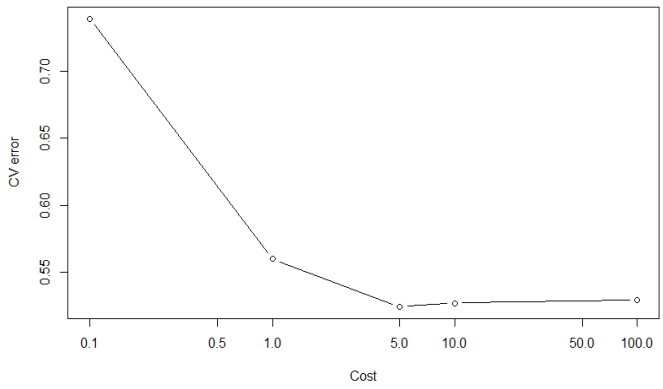


Fig11 : Optimisation du coût pour un SVM à noyau laplacien

Nous choisissons donc le kernel de laplace, et nous optimisons de la même manière sigma. La variance ne semble pas trop élevé et le biais est réduit. N ous choisissons de séparer notre modèle en 3 partie : train, test et validation. Nous testons notre modèle sur des données de validation qui sont vierge de toute modification afin d'obtenir un MSE moins biaisé.

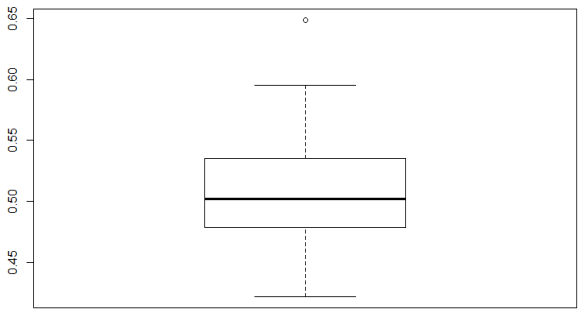


Fig12 : LaplacianKernel variance avec C=5 et un sigma optimisé

2.6.4 - Multilayer Perceptron

Pour finir, nous avons essayé un réseau de neuronne sur cette dataset (MLP) et utilisons la librairie keras qui nous offre plein de possibilités. Nous avons considéré un modèle de réseau de neuronne à 4 couches cachés avec activation ReLu. Nous constatons un certain overfitting si nous laissons le modèle comme cela (le paramètre loss augment au bout d'un certain temps). C'est pourquoi nous employons un kernel_regulizer qui permet de régulariser les données. Nous ajoutons aussi un callback d'early stop pour minimiser les risques d'overfitting. Ansi, nous obtenons un MSE à 0.629012 ce qui est un bon score, mais moins bon que le SVM laplace. Nous avons ausssi ici un Deep Neural Network modèle avec beaucoup de paramètre, ce qui le rend vraiment flexible, et peu interprétable.

2.7 - Conclusion

Nous avons utilisé différentes méthodes de régularisation, transformation ou encore selection de variables en vain. Le modèle sur lequel nous allons porter notre regard est celui de la boite noire SVM avec un noyau de laplace et des paramètres optimisés, qui nous donne un MSE extrêmement faible.

3 Classification d'images en 3 catégories

3.1 Analyse de données

Après avoir examiné les données, on trouve que les données sont des images colorées en tailles différentes. Pour l'ordinateur ils sont des matrice de dimension $x \times y \times 3$ avec des x et y différents. Par conséquent il faut régler les tailles des données avant entraîner le modèle. Dans notre cas nous les avons réglées en $200 \times 200 \times 3$ Puis enlevé des points qui ne portent pas trop d'information, c'est-à-dire des points ayant une variance très petite parmi des images. Pour certaines méthodes il faut aussi transformer des images en gris.

La repartition du nombre d'image par classe semble tout de fois correct.

3.2 Choix du modèle

3.2.1 Support Vector Machine

Support Vector Machine (SVM) est un outil très fort en problèmes de classification non-linéaire. Pour appliquer SVM sur ce problème, les images sont transformées en gris, donc des matrices 200×200. Puis la matrice est transformée en un vecteur de dimention 10000, qui sont des "vraies entrées". Comme il y a trop de dimensions dans les données d'entrées, nous avons pensé à diminuer la dimension par retirer des pixels non-informatif (les pixels dont la variance est très petit), mais nous avons trouvé que tous les pixels ont des variances similaires. C'est différent que le problème d'expressions que nous avons traité, parce que dans cette situation les images sont plus complexes. Le résultat de SVM n'est pas satisfaisant. Le taux d'erreur était environ 0.4. Donc nous décidons d'aller chercher une meilleure solution.

3.2.2 Multilayer Perceptron

Multilayer Perceptron (MLP) est un réseau de neurones très simple. Il est consisté avec des couches des neurones. Pour chaque couche, une "caractéristique" est extraite et utilisée pour déterminer des caractéristiques de la couche suivant. Il est plus raisonnable d'utiliser un MLP pour traiter des images, car même si pour les êtres-humain, nous classifions des images par caractéristiques. En MLP nous prétraitons des images comme en SVM, donc la première couche de neurones est composé de 10000 neurones. Néanmoins, la précision de validation du modèle MLP varie beaucoup. Il n'augmente pas avec les epochs comme prévu. Nous avons ajouté plus de couches mais cela ne fait rien. Un modèle MLP est bon en terme d'extraction des caractéristiques donc il est bon pour les classifications des images simples tel que des chiffres écrits. Il ne prend pas en compte des relations parmi des pixels ajacents.

3.2.3 Convolutional Neural Networks

Convolutional Neural Networks (CNN) est très célèbre pour son abilité d'apprentissage en traitement des images. Chaque fois il prend en compte des pixels ajacents donc il est parfait pour traiter des données 2-dimensionnels comme image. Grâce au package `keras`, nous pouvons construire un CNN avec facilité.

3.2.3.1 Structure de CNN

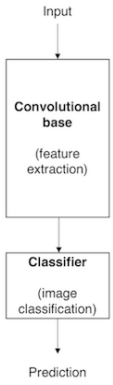


Fig13 : Structure du Convolutionnal Neural Network

Nous avons construit un CNN assez simple, mais très fort. Il est composé de couches de convolution avec un noyau de 3×3.

```
## Model: "sequential"
## _________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =================================================================
##  conv2d (Conv2D)             (None, 100, 100, 16)      448
## _________________________________________________________________
##  conv2d_1 (Conv2D)           (None, 100, 100, 16)      2320
## _________________________________________________________________
##  conv2d_2 (Conv2D)           (None, 50, 50, 16)        2320
## _________________________________________________________________
##  conv2d_3 (Conv2D)           (None, 50, 50, 16)        2320
## _________________________________________________________________
##  dropout (Dropout)           (None, 50, 50, 16)        0
## _________________________________________________________________
##  conv2d_4 (Conv2D)           (None, 25, 25, 32)        4640
## _________________________________________________________________
##  conv2d_5 (Conv2D)           (None, 25, 25, 32)        9248
## _________________________________________________________________
##  conv2d_6 (Conv2D)           (None, 13, 13, 32)        9248
## _________________________________________________________________
##  conv2d_7 (Conv2D)           (None, 13, 13, 32)        9248
## _________________________________________________________________
##  dropout_1 (Dropout)         (None, 13, 13, 32)        0
## _________________________________________________________________
##  conv2d_8 (Conv2D)           (None, 7, 7, 64)          18496
## _________________________________________________________________
##  conv2d_9 (Conv2D)           (None, 7, 7, 64)          36928
## _________________________________________________________________
##  conv2d_10 (Conv2D)          (None, 4, 4, 64)          36928
## _________________________________________________________________
##  conv2d_11 (Conv2D)          (None, 4, 4, 64)          36928
## _________________________________________________________________
##  flatten (Flatten)           (None, 1024)              0
## _________________________________________________________________
```

```
## dropout_2 (Dropout)              (None, 1024)              0
## _________________________________________________________________
## dense (Dense)                    (None, 64)                65600
## _________________________________________________________________
## dropout_3 (Dropout)              (None, 64)                0
## _________________________________________________________________
## dense_1 (Dense)                  (None, 3)                 195
## =================================================================
## Total params: 234,867
## Trainable params: 234,867
## Non-trainable params: 0
## _________________________________________________________________
```

3.2.3.2 Traitement des données

Pour faire la validation au même temps d'apprentissage, nous avons divisé des données en deux groupes : l'un pour entraînement et l'autre pour validation. Tous les images sont mises en $200 \times 200 \times 3$ pour l'apprentissage. Pour le CNN, il n'est pas nécessaire d'éliminer la dimension couleur donc il garde plus d'informations. Nous avons aussi trouvé des autres images sur internet pour tester.

Nous avons aussi essayé de transformer en augmentant les images (uniquement celles pour le training) : Pour celà, nous avons transformé nos images à l'aide du package OpenImageR. En fait, nous dupliquons nos photos avec des augmentations aléatoires tells que du parallélisme, rotations, ou la fonction `ZCAwhiten.`

3.2.3.3 Apprentissage

L'apprentissage d'un CNN est en époques et en chaque époque, seulement un lot d'images est utilisé. Le nombre d'époque ne doit être ni trop grand car il prend trop de temps, ni trop petit car le modèle peut être surappris. 30 est un nombre juste pour ce problème. Pour éviter le surapprentissage, nous avons défini une fonction de rappel (early_callback) qui termine l'apprentissage lorsque la fonction de perte de validation commence à augmenter. Le résultat est satisfaisant, comme nous avons obtenu une précision de validation de plus de 92%.

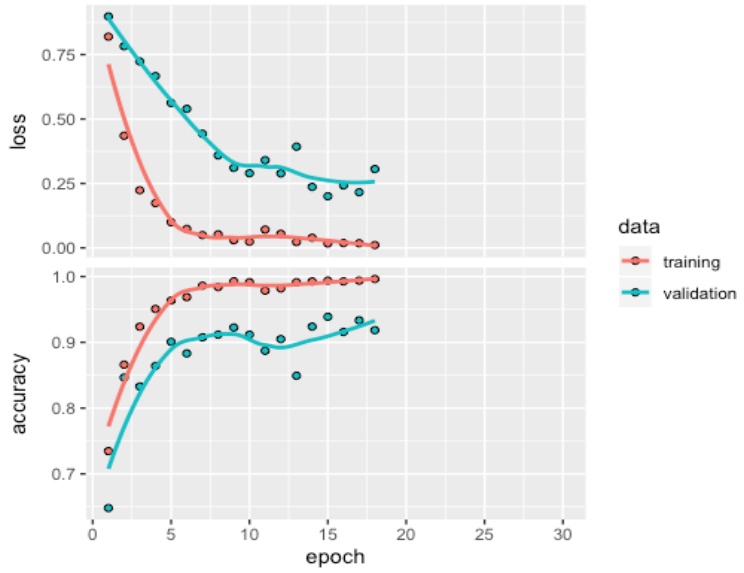


Fig14 : Apprentissage du modèle CNN montrant la loss et l'accuracy en fonction de l'epoch

3.3 Conclusion

Après avoir essayé les trois modèles, il est sans nul doute que le CNN est le meilleur pour la classification des images. Il aurait été intéressant et utile d'utiliser un modèle pré-entraîné afin d'améliorer la prediction (par exemple en utilisant un modèle qui a été entraîné sur un large jeu de données).