Rapport du TP n°7 SY19

Compte tenu du nombre de pages limité dans le rapport, nous ne montrerons pas nos démarches en exhaustivité, mais préférons montrer des méthodes différentes pour la classification et la régression. Bien sûr, certaines des méthodes sont valables dans les deux cas, par exemple pour les transformations de variables manuelles.

Suite à un différent avec mon binôme qui n'a pas apporté de contribution, j'ai décidé de rendre le TP seul. Je vous remercie d'éventuellement en tenir compte pour la notation.

I. Classification

A) Sélection de variables

Nous disposons de 5000 entrées correspondant à des objets astronomiques de trois types (étoile, galaxie, quasar) avec 17 prédicteurs. L'objectif est de trouver un modèle performant pour prédire l'appartenance de nouvelles données à ces 3 classes.

En utilisant la fonction levels(), nous pouvons directement éliminer deux prédicteurs qui sont constants sur l'ensemble du dataset, et qui ne vont par conséquent pas apporter d'informations pour la variable à prédire. Ces prédicteurs sont : "objid" et "rerun". De plus, MJD signifie "Modified Julian Date"[1] et à priori, la date de l'observation n'est pas en lien avec le type de l'objet. Cela reste une hypothèse, et il faudrait avoir plus de détails sur le dataset pour pouvoir l'affirmer.

Afin d'évaluer la performance de nos différents modèles de classification, j'ai défini une fonction retournant le nombre d'observations correctement classifiées :

```
perf <- function(y_true, y_pred) {
  print(table(y_true,y_pred))
  cat("Taux de réussite :")
  mean(y_true==y_pred)
}
```

En explorant le dataset, nous remarquons que certaines variables sont fortement corrélées. Le graphique de corrélation n'est pas affiché ici afin de respecter les 12 pages.

Pour éliminer les prédicteurs redondants, nous réalisons une ACP sur notre dataset afin de réduire le nombre de variables. Après analyse de notre ACP, il apparaît que 90% de l'information de notre dataset est contenu dans 6 à 9 variables. Cependant, lors des tests

[1] http://scienceworld.wolfram.com/astronomy/ModifiedJulianDate.html

sur différentes méthodes, le modèle complet donne systématiquement de meilleurs résultats.

Selon le domaine d'application, nous aurions pu privilégier un modèle parcimonieux avec des performances légèrement plus faibles, mais plus facilement interprétable.

B) Transformation de prédicteurs

Ici, nous allons essayer d'ajouter de nouveaux prédicteurs à notre modèle, comme variables transformées ou combinaisons des différentes variables. Voici nos nouveaux prédicteurs :

```
data$A <- data$u*data2$g*data2$r*data2$i*data2$z
data$B <- data$A^2
data$C <- data$A^3
data$D <- data$ra^2
data$E <- data$ra^3
data$F <- data$dec^2
data$G <- data$dec^3
```

Ce nouveau modèle s'est finalement avéré avoir des performances égales ou inférieures aux modèles avec le sous-ensemble de prédicteurs initial.

C) Tests de différentes méthodes

J'ai commencé par évaluer les méthodes de Gaussian Mixture Models LDA, QDA et Naives Bayes, puis SVM et Random Forest. Ces deux derniers, se sont avérés être nos modèles les plus performants.

LDA:

En cross validation, nous avons obtenu une erreur de 90,49% +/- 0.004201%. Afin d'obtenir un intervalle de confiance sur la précision, nous avons évalué l'erreur du modèle en utilisant 10 passes de cross validation.

QDA:

J'ai utilisé le même processus pour entraîner et évaluer les performances du modèle QDA. Le taux de précision obtenu est de 97,20% +/- 0.003994%.

Interprétation des résultats :

Remarquons que les méthodes QDA et LDA supposent toutes les deux que les données des 3 classes suivent des lois normales multivariée. LDA suppose également que ces 3 lois normales multivariées ont la même matrice de covariance.

Or, en calculant la matrice de covariance pour chacune des classes et en faisant la différence absolue moyenne entre chaque couple de matrices, nous obtenons des résultats élevés:

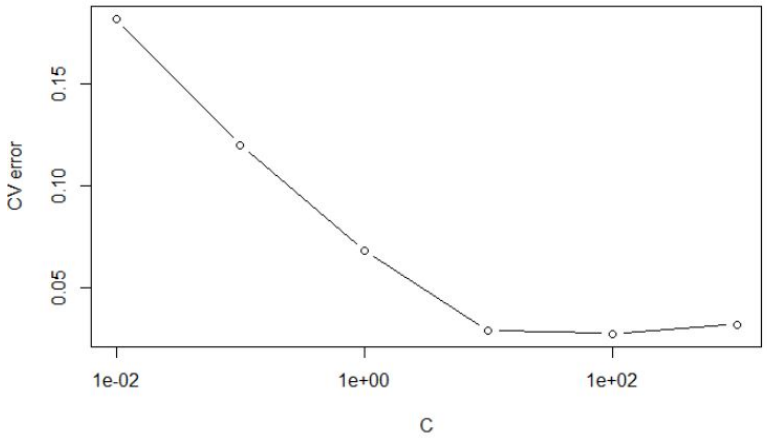
Entre 1 et 2	1.783835e+34
Entre 2 et 3	5.951597e+32
Entre 1 et 3	1.843351e+34

Ce qui peut expliquer pourquoi la méthode QDA a un taux de précision supérieur à la méthode LDA.

SVM:

Le modèle obtenu par SVM est l'un des meilleurs, derrière Random Forest que nous allons voir dans la section suivante.

Nous avons déterminé le paramètre C par cross validation, en utilisant le fait que la fonction SVM du package "kernlab" permet de directement faire la CV.



Le paramètre C optimal semble être 100, et nous obtenons de meilleures performances avec un kernel polynomial.

Avec 10 passes de cross validation sur l'ensemble des données du sujet, nous obtenons une performance de 98,24% +/- 0.0949.

Random Forest :

Notre second modèle optimal, est celui obtenu par Random Forest, avec lequel nous obtenons une performance de de **98,66%**, +/- 0.0794.

Récapitulatif des résultats :

Ci-dessous, un récapitulatif des performances de nos différents modèles :

LDA	90,49% +/- 0.004201%
QDA	97,20% +/- 0.003994%
SVM (C=100, kernel polynomial)	**99,22% +/- 0.003815%**
Random Forest	98,75% +/- 0.014072%

L'intervalle de confiance pour Random Forest est légèrement plus élevé, car le modèle est plus lourd et a pris plus de temps à être entraîné et évalué. Nous avons donc effectué seulement 5 passes de Cross Validation pour Random Forest.

D) Améliorations possibles

Avec plus de temps, j'aurais aimé estimer les paramètres optimaux des noyaux de SVM. Actuellement, ce sont les paramètres par défaut. Aussi, je n'ai pas analysé en détail les données de base, afin de voir si des observations semblaient aberrantes et pourraient détériorer les performances de notre modèle.

II. Régression

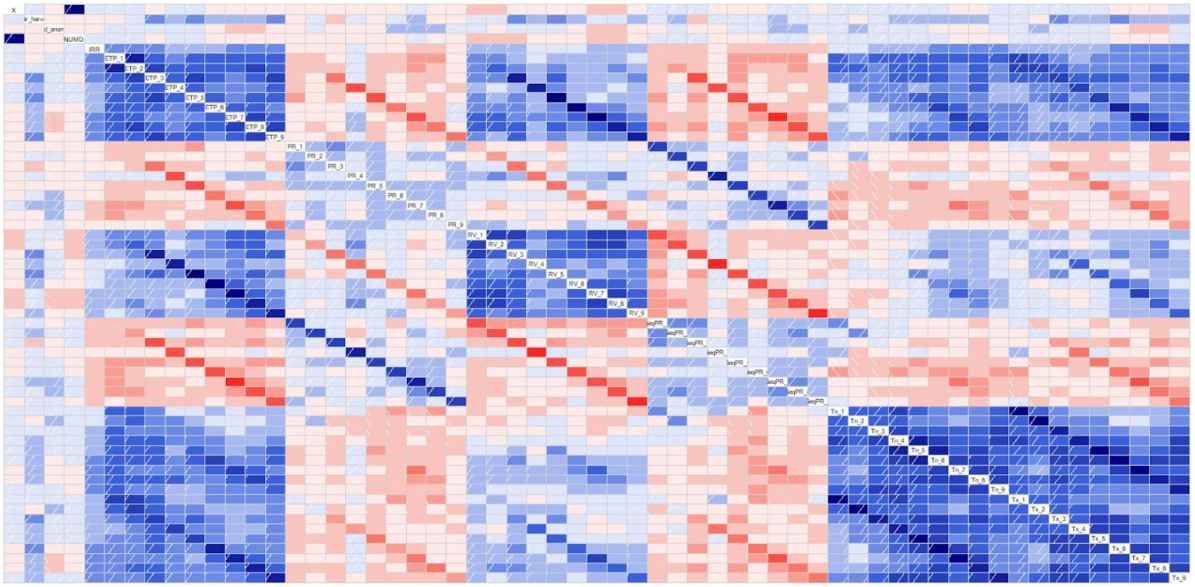
Pour ce problème de régression, nous ne disposons pas d'informations sur le premier prédicteur, qui n'a pas de nom quand on ouvre le fichier mais_train.csv, mais R y attribue le nom "X" quand on charge le fichier. Le sujet ne nous donne pas d'indication sur cette colonne. Avec NROW(levels(factor(data$X))), nous pouvons remarquer que il y a autant de valeurs uniques que d'observations. En combinant cette information avec la position de cette colonne, nous pouvons penser qu'il s'agit d'un identifiant, et que cette colonne peut être éliminée.

A) Sélection de variables

Nous avons étudié la corrélation entre les différents prédicteurs, qui est assez importante dans ce jeu de données.

Par exemple, ETP_5 et RV_5 sont corrélées à 88%.

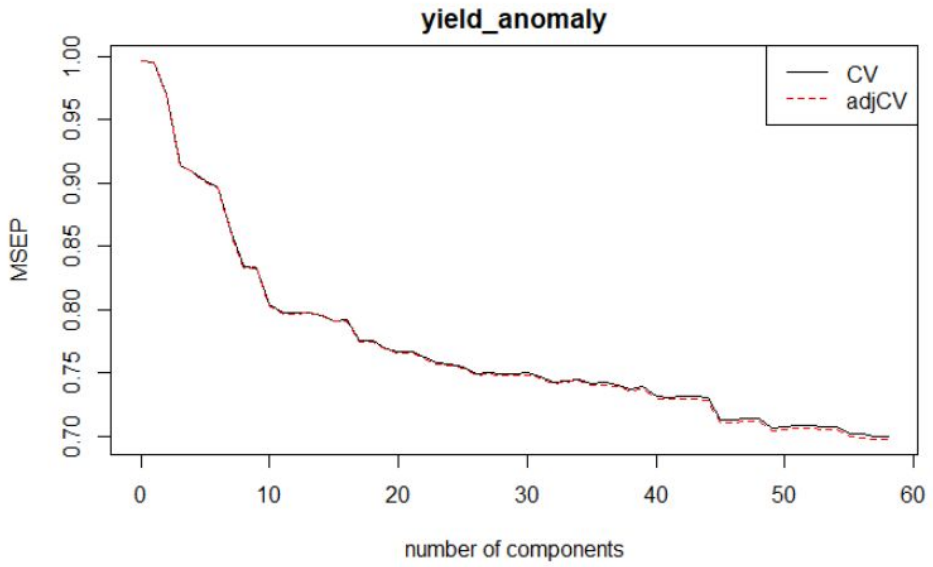
Pour étudier de façon plus graphique ces corrélations, nous avons étudié le graphique ci-dessous, (généré en utilisant la fonction corrgram de la librairie du même nom)



Les couleurs sombres indiquent une corrélation forte. Cette visualisation nous permet d'identifier certains patterns, comme les diagonales sombres. En étudiant la signification du jeu de données, nous pouvons comprendre pourquoi certaines variables sont corrélées, en voici deux exemples :

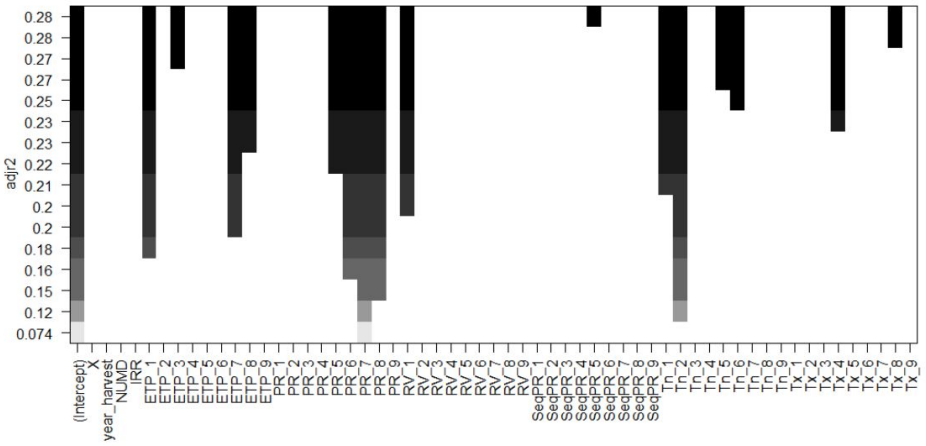
- ETP_X et RV_X sont souvent beaucoup corrélées, et cela a du sens : plus l'ensoleillement est important, plus l'évapotranspiration est importante.
- De même pour Tn_y et Tx_y : plus la température minimale est importante, plus la température maximale est importante. En effet, ces deux variables exhibent la même information : la température globale sur un mois et un département donné.

Par conséquent, nous avons utilisé l'ACP, cette méthode étant particulièrement utile lorsque les variables, dans le jeu de données, sont fortement corrélées. La corrélation indique qu'il existe une redondance dans les données. En raison de cette redondance, nous utilisons l'ACP pour essayer de réduire les variables d'origine en un nombre plus petit de nouvelles variables.

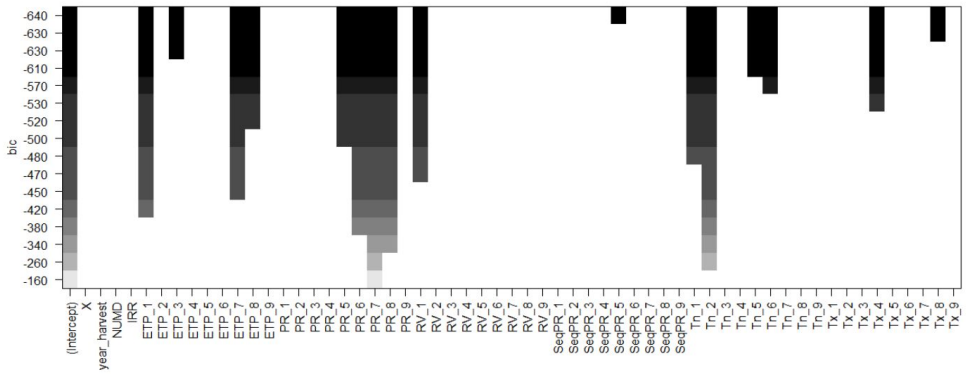


A priori, le modèle complet donne de meilleurs résultats.

Nous utilisons ensuite une autre méthode, regsubsets, pour étudier les prédicteurs les plus explicatifs. La méthode exhaustive n'étant pas possible compte tenu de nos 58 prédicteurs, nous avons utilisé les méthodes "forward" et "backward". Les résultats étant similaires, nous montrons ici seulement la méthode "forward". Nous utilisons le R^2 ajusté dans un premier temps, mais aussi le critère BIC pour mesurer la qualité des différents modèles tout en pénalisant le nombre de paramètres, afin d'obtenir un modèle plus facilement interprétable.



Puis avec un autre indicateur, le BIC :



Les résultats sont assez similaires, et nous pouvons déduire de ces méthodes, différents sous-modèles que nous allons tester :

Taille	Prédicteurs	Méthode d'obtention
58	Modèle complet	N/A
26	IRR+ETP_3+ETP_4+ETP_6+ ETP_7+ ETP_8+ ETP_9+ PR_2+ PR_5+ PR_6+ PR_7+ PR_8+ RV_1+ RV_3+ RV_6+ RV_7+ RV_8+ RV_9+ SeqPR_2+SeqPR_3+SeqPR_4+SeqPR_5+Tx_3+Tx_4+Tx_8+Tx_9	p-values en régression linéaire
16	ETP_1+ETP_3+ETP_7+ETP_8+PR_5+PR_6+ PR_7+ PR_8+RV_1+SeqPR_5+Tn_2+Tn_1+Tn_5+Tn_6+Tx_4+Tx_8	regsubsets selon R^2 ajusté
16	ETP_1+ETP_3+ETP_7+ETP_8+PR_5+PR_6+ PR_7+ PR_8+RV_1+SeqPR_5+Tn_2+Tn_1+Tn_5+Tn_6+Tx_4+Tx_8	regsubsets selon BIC
58	Modèle complet	Analyse en composantes principales

Lors du test de ces modèles, le modèle complet avait systématiquement les meilleures performances (en comparant à chaque fois l'erreur quadratique moyenne mais aussi les intervalles de confiance, afin d'éviter de classer un modèle plus performant qu'un autre à cause d'une erreur statistique).

B) Tests de modèles

Nous avons testé différents modèles, qui ont tous été réalisé avec 10 passes de cross validation sur 10 blocs. SVR et Random Forest sont les méthodes ayant donné les meilleurs résultats. Nous les comparons avec une régression linéaire, mais nous ne décrivons pas ici les autres méthodes que nous avons testé, compte tenu de leurs résultats médiocres, et afin de gagner de la place.

Modèle linéaire:

Le modèle linéaire classique nous a donné les erreurs suivantes selon les sous-ensembles de prédicteurs trouvés dans la partie précédente:

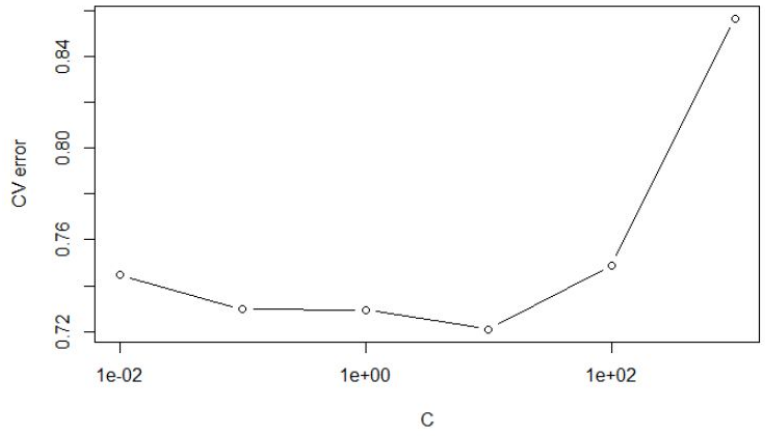
	R^2	R^2 ajusté	AIC	BIC
Modèle complet	**0.3071**	**0.3250**	**5675.7**	6020.1
Modèle selon p-values de RL	0.2766	0.2683	5829.9	5990.6
Modèle selon regsubsets	0.2840	0.2790	5786.0	**5889.3**

Il est probable qu'en étudiant la dispersion des résidus sur le modèle complet, certains patterns apparaissent, car beaucoup de variables sont corrélées entre elles.

Pour la suite, seule les performances du modèle complet seront affichés, car lors de nos tests, l'erreur était systématiquement la plus faible.

SVR:

Pour la régression à vecteurs de support, nous avons commencé par déterminer le paramètre C par cross validation.



Le paramètre C optimal semble être 10. Nous avons utilisé cette valeur pour tester notre modèle avec 10 cross validations, et nous obtenons en MSE la valeur suivante : 0.5826. Environ 95% des valeurs sont contenues dans l'intervalle suivant : [0.5813013;0.5838744].

Random Forest:

Pour la régression, notre modèle le plus performant est celui obtenu par Random Forest avec le modèle complet. Nous obtenons un MSE de 0.5583 +/- 0.0033.

C) Récapitulatif des résultats

Les valeurs à ajouter ou soustraire pour obtenir l'intervalle à environ 95% sont données après les erreurs.

Modèle	R^2	MSE
Linéaire	0.3071 +/- 0.0024	0.6950 +/- 0.0018
SVR	0.4472 +/- 0.0025	0.5537 +/- 0.0027
Random forest	**0.4584 +/- 0.0035**	**0.5583 +/- 0.0033**

D) Améliorations possibles

Nous aurions pu davantage étudier le dataset pour vérifier qu'il n'y ai pas d'observations aberrantes qui pourraient fausser notre modèle.
Nous aurions aussi pu utiliser un modèle certes moins performant, mais plus explicable, en utilisant les résultats de la PCA pour combiner les variables qui expliquent la même chose.
J'aurais aimé pouvoir tester tous les modèles obtenus par sélections de variables, sur toutes les méthodes, avec cross validation. Cependant, cela prenait trop de temps, notamment pour SVR et Random Forest.
Regsubset peut éventuellement s'utiliser en classification quand il y a deux classes, nous aurions donc pu l'utiliser en testant une classe contre une autre ou une classe contre les autres.
Enfin, nous aurions voulu tester des transformations de variables manuelles, comme ce que nous avons fait en classification. Par exemple, faire une moyenne des températures maximales et minimales.

III. Classification d'images

L'objectif est ici de classer des images selon ce qu'elles représentent : voiture, chat ou fleur.

A) Chargement des images

Pour charger les images en R avec Keras, nous utilisons la fonction flow_images_from_directory() qui permet de générer des batchs d'images depuis un certain dossier.

Cette fonction permet de charger les images tout en appliquant des modifications éventuelles : augmentation d'image ou changement de taille par exemple.

L'objet retourné par cette fonction est un générateur d'images, qui peut être utilisé de différentes façons, pour entraîner le réseau de neurones, pour faire de la prédiction, ou pour l'évaluer.

B) Description du réseau de neurones

Après plusieurs essais de réseaux plus ou moins complexes, nous avons décidé de garder notre premier essai qui était le plus simple, mais aussi le plus performant.

C'est un réseau de neurones à 3 couches.

La première couche est de type "**flatten**" et convertit nos images d'entrée, d'un tableau à 3 dimensions (1 dimension pour la classe à laquelle l'image appartient, et 2 dimensions pour la valeur des 20*20 pixels en hauteur et largeur, de 0 à 255), vers un vecteur d'une seule dimension de 20*20 = 400 pixels. Cette couche n'a aucun paramètre et se contente de formater nos données en alignant les valeurs des pixels.

Les deux couches suivantes sont appelées "**denses**" avec Keras. Elles sont complètement connectées entre elles : chaque neurone de la première couche est connecté avec chaque neurone de la seconde couche. La première couche possède 128 neurones et une fonction d'activation "**rectifieur**" qui retourne max(x,0), et la seconde seulement 3, correspondant à chaque classe à prédire. La fonction d'activation de cette dernière couche est de type "**softmax**", ce qui permet de retourner un tableau de scores de probabilités d'appartenance à chaque classe, au lieu d'une variable catégorique.

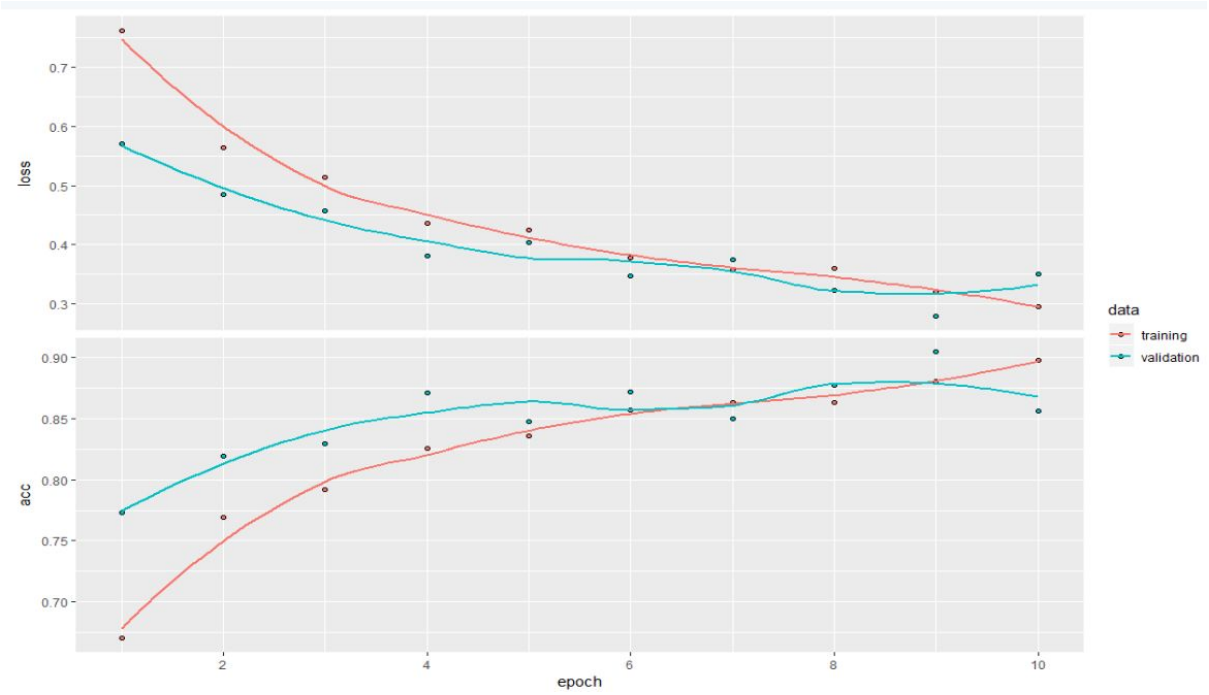
C) Entraînement et validation

Nous utilisons un optimizer basé sur l'algorithme Adam, qui est une extension de l'algorithme du gradient stochastique vu en cours. A la différence de ce dernier, le taux d'apprentissage est variable, et la méthode intègre une composante d'inertie[2].
Pour fonction d'objectif, nous utilisons "categorial_crossentropy" de keras qui correspond au negative log likelihood vu en cours.

Pour entraîner le réseau et optimiser la valeur des paramètres de chaque couche, nous utilisons une fonction fit_generator() qui va de pair avec nos générateur d'images.

Après plusieurs essais, nous déduisons que 10 est un nombre de passes convenable. Une passe correspond à un passage du dataset d'entraînement et de validation, avec corrections associées sur les poids des neurones.

[2] https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/



D) Comparaison avec un réseau de neurones pré-entrainé

Finalement, j'ai voulu comparer notre réseau de neurones avec des réseaux connus et pré-entraînés, comme par exemple le RestNet50. A la différence d'un réseau de convolution contenant de nombreuses couches connectées entre elles, un réseau de neurones résiduel comme le ResNet50 permet d'avoir des "raccourcis" entre différentes couches, ce qui peut accélérer le début de l'apprentissage.[3]

Ce réseau de neurones développé par Microsoft a gagné de nombreuses compétitions de classification d'images, il est donc très performant.

Voici une comparaison de la performance de notre réseau avec le ResNet50:

Modèle	Performance sur 60 images du dataset	Performance sur 60 Images tirées au hasard depuis une banque d'images[4]
ResNet50	100%	98,4%
Réseau SY19 - 10 epochs	88,33%	75,00%
Réseau SY19 - 5 epochs	83,33%	71,67%

[3]https://www.quora.com/What-is-the-deep-neural-network-known-as-%E2%80%9CResNet-50%E2%80%9D
[4] http://www.image-net.org/

La performance est calculée comme la moyenne des images correctement classifiées.

Sans surprise, le réseau ResNet50 a des performances très supérieures. Ce réseau s'approche en effet de l'état de l'art du deep learning, et dans certains cas d'applications, il a surpassé les performances d'une classification faite par un humain[5].

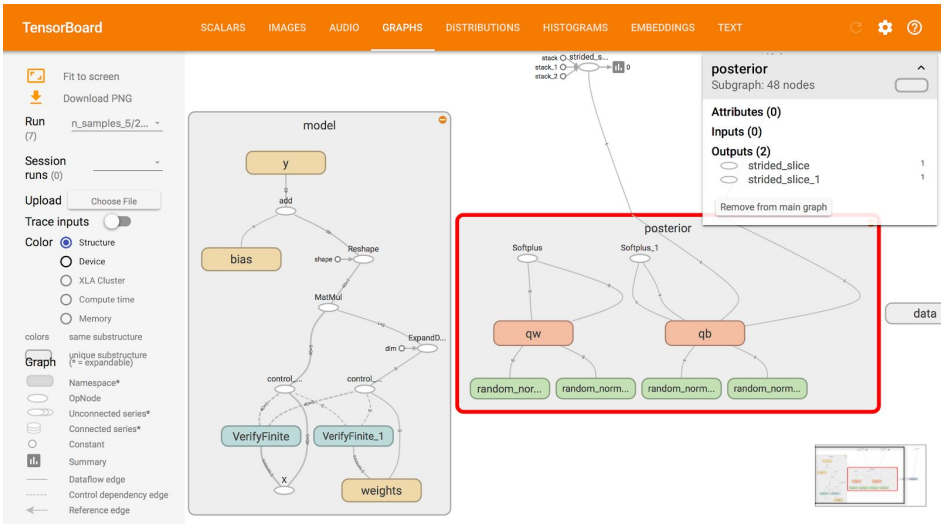
E) Améliorations possibles

Pour augmenter la variété des données d'entraînement, j'aurais aimé intégrer notre réseaux avec des images de différentes sources et de différents formats, par exemple en utilisant différentes banques d'images.
Avec plus de temps, j'aurais étudié plus en détail notre réseau, afin de le tuner et de comprendre quelles architectures ont les meilleurs performances, et pourquoi. J'aurais aussi dû effectuer une K-fold cross validation sur les performances de nos réseaux de neurones, mais cette méthode prend énormément de temps avec ce type de modèle. Certaines solutions existent pour distribuer ces opérations.

De plus, certaines images d'entraînement de mauvaise qualité auraient éventuellement pu être enlevées des données d'entraînement, par exemple flower_train_18.jpg ci-contre.

Cependant, il peut être intéressant de les laisser pour que le réseau soit capable de classifier des images de mauvaise qualité.

Enfin, j'aurais aimé utiliser les callbacks TensorBoard[6] afin d'avoir accès à davantage d'outils de visualisation. Ces callbacks permettent d'avoir par exemple des histogrammes d'activation des différentes couches, ou simplement des graphes dynamiques des différentes métriques, avec l'interface suivante :



[5] https://arxiv.org/abs/1502.01852
[6] https://keras.rstudio.com/articles/training_visualization.html#tensorboard