# Forest Cover Type Prediction

**Kevin Zagalo**
kevin.zagalo@etu.upmc.fr

**Ismail Benkirane**
ismail.benkirane@etu.upmc.fr

Projet pour le cours *Apprentissage Statistique* du LIP6 [0], Sorbonne Université — Janvier 2019

---

**Résumé**

Ce projet a pour but de proposer et tester des modèles pour l'étude de la base de données *Covertype* [1]. Il s'agit d'un problème de classification multi-classe avec 7 classes : Spruce/-Fir, Lodgepole Pine, Ponderosa Pine, Cottonwood/Willow, Aspen, Douglas-fir, Krummholz, et 581 012 instances de 54 attributs sans données manquantes.

Les attributs sont les suivants :

| Nom | Unité | Description |
|---|---|---|
| `Elevation` | mètres | Altitude |
| `Aspect` | degrés | Orientation |
| `Slope` | degrés | Pente |
| `Horizontal_Distance_To_Hydrology` | mètres | Distance horizontale au point d'eau le plus proche |
| `Vertical_Distance_To_Hydrology` | mètres | Distance verticale au point d'eau le plus proche |
| `Distance_To_Roadways` | mètres | Distance horizontale à la route la plus proche |
| `Distance_To_Fire_Points` | mètres | Distance horizontale au départ de feu le plus proche |
| `Hillshade_9am` | entier entre 0 et 255 | Ombrage à 9h au solstice d'été |
| `Hillshade_Noon` | entier entre 0 et 255 | Ombrage à 12h au solstice d'été |
| `Hillshade_3pm` | entier entre 0 et 255 | Ombrage à 15h au solstice d'été |
| `Wilderness_Area` | 4 colonnes binaires | Wilderness area designation |
| `Soil_Type` | 40 colonnes binaires | Type de sol |
| `Cover_Type` | entier entre 1 et 7 | Classe |

0. Laboratoire d'Informatique de Paris 6 : : `https://www.lip6.fr`
1. `https://archive.ics.uci.edu/ml/datasets/Covertype`

# Table des matières

# 1 Analyse préliminaire et pré-traitement des données

On choisit d'utiliser la bibliothèque `pandas` pour charger les données, surtout pour l'analyse préliminaire. `pandas` fournit une panoplie de fonctions pour visualiser les données. `groupby`, `boxplot` et `hist` nous seront forts utiles pour choisir les données que nous exploiterons. Le *notebook* contenant le code joint au rapport nécessite aussi les bibliothèques `matplotlib`, `numpy`, `sklearn` et `plotly`.

Avant toute modification des données et/ou élaboration de méthodes, nous tenterons de mieux comprendre les données pour éventuellement les modifier, c'est-à-dire :

— exhiber des corrélations
— supprimer des données inutiles
— ajouter des données qui seraient plus pertinentes
— modifier la façon de "qualifier" les données qualitatives

Tout d'abord on constate sur la figure 1 que les données sont inégalement réparties selon les classes. Cela peut vouloir dire plusieurs choses : soit nos données sont mal échantillonnées, soit les types `1` et `2` sont effectivement largement plus répandues.

C'est quelque chose dont nous n'avons pas la maitrise, une discussion avec un expert sur le sujet serait préférable. On prendra donc cela en compte dans nos méthodes.

La suite consistera globalement à faire la même chose sur le reste des données grâce aux méthodes de la bibliothèque `pandas`.

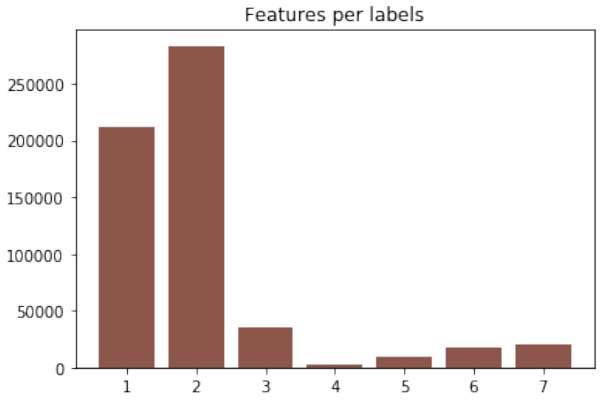
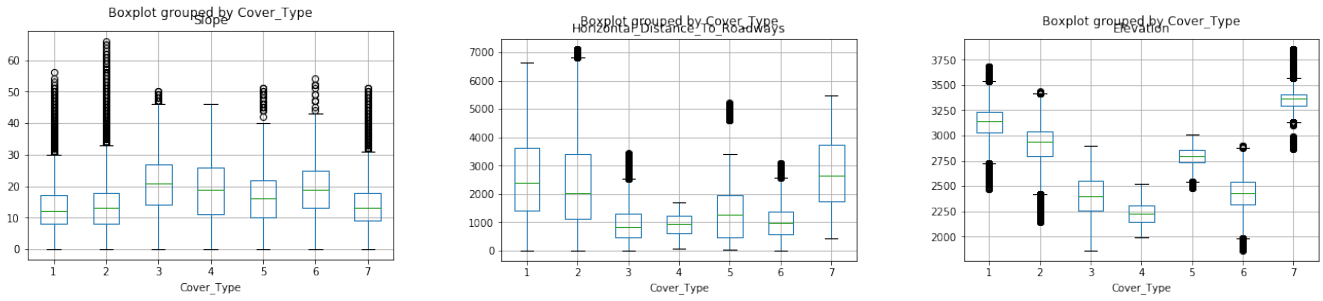



Figure 1 – Histogramme des données par types de forêts

On observe dans la figure 2 l'importance des attributs `Elevation`, `Slope` et `Distance_To_Roadways`.




(a) `Slope` (b) `Distance_To_Roadways` (c) `Elevation`

Figure 2 – Boxplot des données numériques

La première partie consistera donc à voir si on peut réduire le nombre de paramètres, la deuxième à modifier les données pour une meilleure analyse, et enfin mettre les données de train, de validation

et de test. On trouve déjà quelques idées dans [BD99].

## 1.1 Réduction des paramètres

Nous le ferons en deux parties : une première fois pour les variables qualitative et la seconde pour les valeurs numériques. On préfèrera garder dans le DataFrame `df_covtype` des entiers plutôt que des vecteurs binaires, quitte à les y remettre dans les données de train et de test ensuite. Cela facilitera grandement l'analyse préliminaire.

L'attribut `Wilderness_Area`n'apporte pas beaucoup d'information, mais elle permet de distinguer la classe 4, qui a du mal a être identifiée par les méthodes testées. En effet, les forêts de classe 4 ne sont que dans 'Cache la poudre'. Nous allons donc garder l'attribut comme un 4-vecteur binaire :
- 1 : Rawah - 2 : Neota - 3 : Comanche Peak - 4 : Cache la Poudre

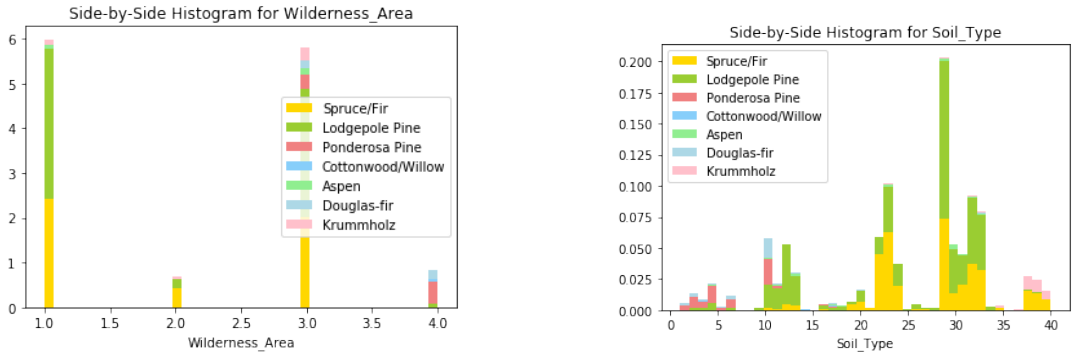



Figure 3 – Histogrammes pour `Wilderness_Area`et `Soil_Type`

Pour réduire presque de moitié les paramètres de `Soil_Type`, nous considérerons uniquement les familles de sols, et le fait qu'ils soient rocheux, friable ou autre, pour atteindre le nombre de 24 paramètres. On trouve ces informations dans le fichier *covtype.info* fourni avec les données.

- 1 to 40 : based on the USFS Ecological Landtype Units for this study area

Nous aurons plutôt deux vecteurs binaires `soil_family` et `soil_group`, respectivement de taille 3 et 21 qui remplaceront la variable `Soil_Type`. On remarquera que le deuxième permet entre autres de mieux classer la classe 7, puisqu'elle n'est présente que dans les forêts rocheuses.

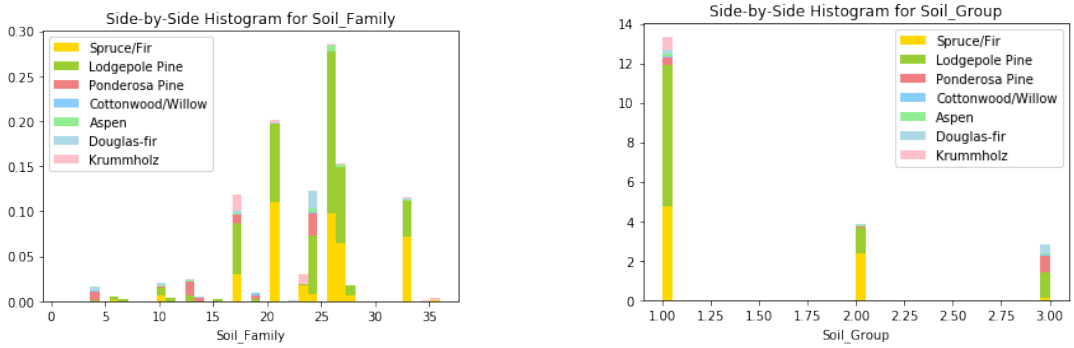



Figure 4 – Histogrammes pour `Soil_Family` et `Soil_Group`

## 1.2 Transformation des données

On commence par chercher des corrélations entre les données :

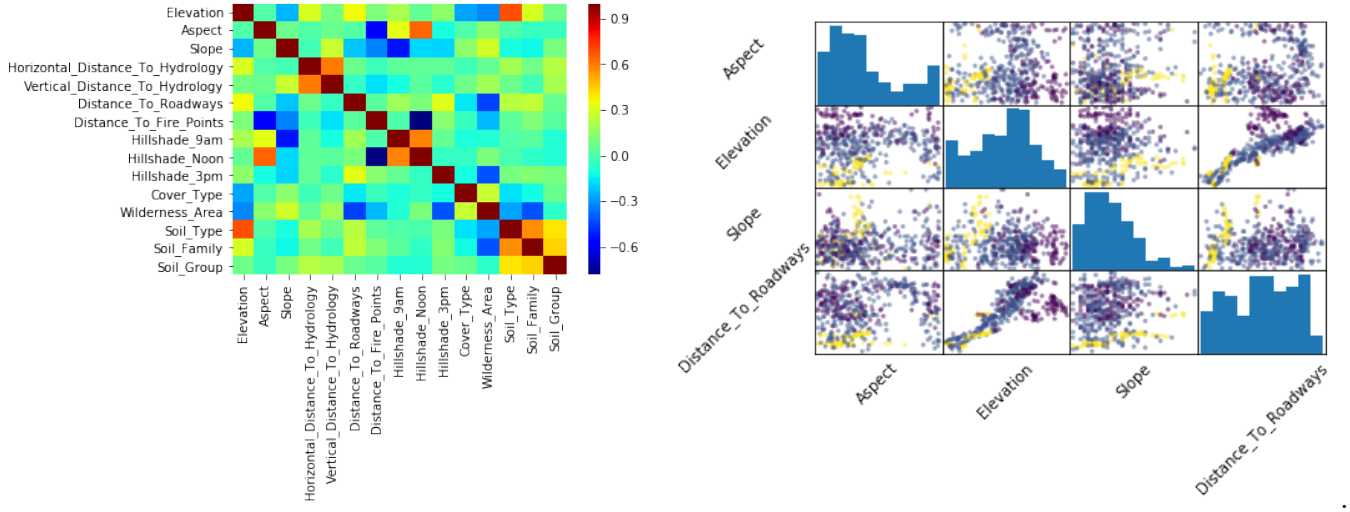



Figure 5 – Matrices de corrélations

On voit ici plusieurs choses :
— Plus la pente est grande, plus l'ombre a de la variance
— L'élévation est grandi linéairement en fonction de la distance aux routes
— L'azimuth et l'ombrage forment des sigmoid

On remplacera `Aspect` par

$$\texttt{Aspect\_Group} = (N, NW, W, SW, S, SE, E, NE)$$

ce qui permet de faire un apprentissage catégoriel plutôt que numérique.

On constate sur la figure 6 que la plupart des forêts sont orientées vers le nord-est, et donc il est logique de trouver que l'ombrage à 3 heures de l'après midi soit plus important que les deux autres. On voit que la distribution de `Hillshade_3pm` ressemble à une gaussienne, ce qui est exploitable.

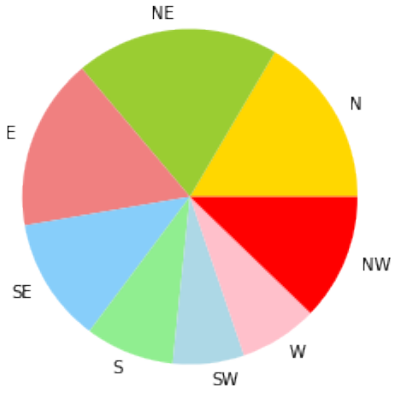



Figure 6 – Répartition de `Aspect`

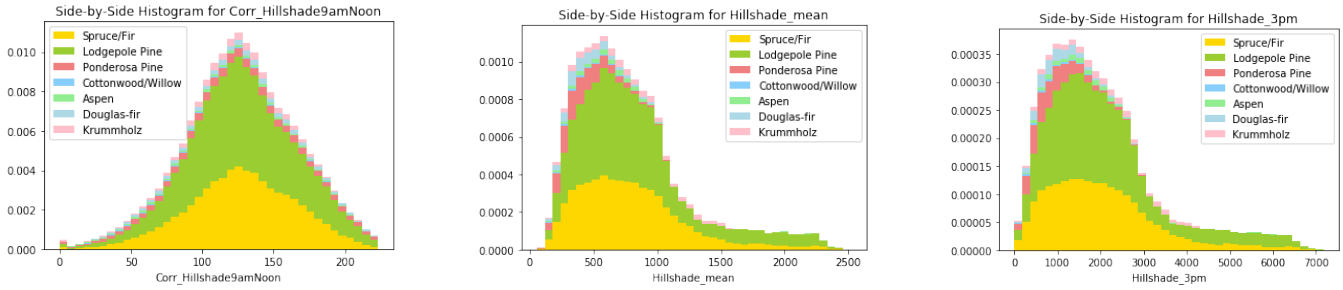



Figure 7 – Répartitions de `Corr_Hillshade9amNoon`, `Hillshade_mean` et `Hillshade_3pm`

On gardera donc `Hillshade_3pm` et deux nouveaux attributs `Corr_Hillshade9amNoon` = `Hillshade_9am` × `Hillshade_Noon`/255 et

$$\texttt{Hillshade\_mean} = \frac{\texttt{Hillshade\_9am} + \texttt{Hillshade\_Noon} + \texttt{Hillshade\_3pm}}{3}$$

L'idée vient du fait qu'en faisant cela, nous nous retrouvons avec une quantité proportionnelle à des distances comme l' `Elevation` ou des polynômes des quantités comme `Hillshade_3pm`.

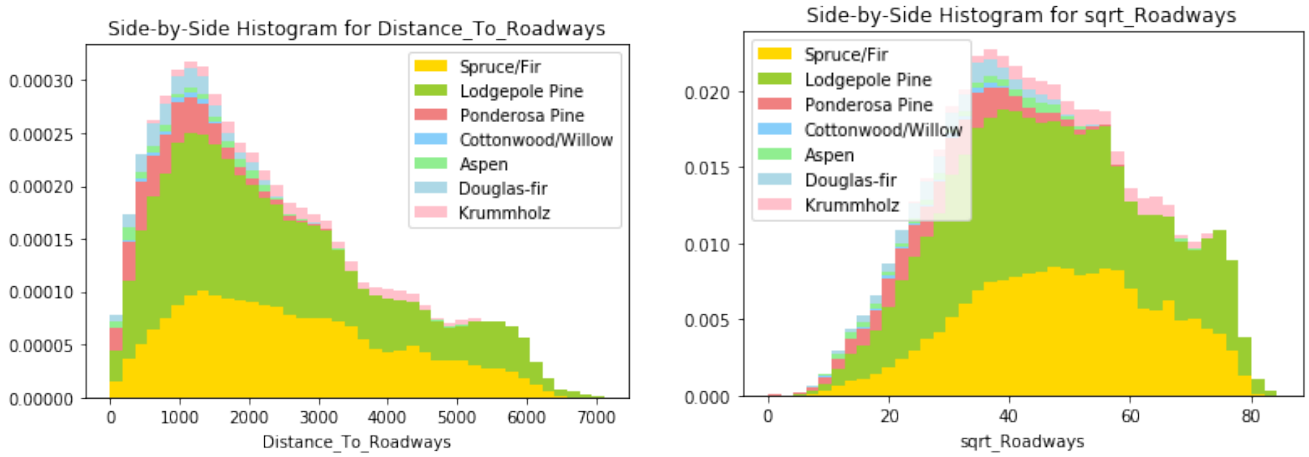



FIGURE 8 – Répartitions de `Distance_To_Roadwayset` `sqrt_Roadways`

Ensuite, nous voyons sur la figure que la distribution de l'attribut `Distance_To_Roadways` ressemble à une $\chi^2$. Si on en prend la racine carrée on se retrouve donc avec une distribution proche d'une gaussienne. Pour la distance à l'eau nous résumerons les deux paramètres à la distance euclidienne à l'eau, c'est-à-dire

$$\sqrt{\texttt{Vertical\_Distance\_To\_Hydrology}^2 + \texttt{Horizontal\_Distance\_To\_Hydrology}^2}$$

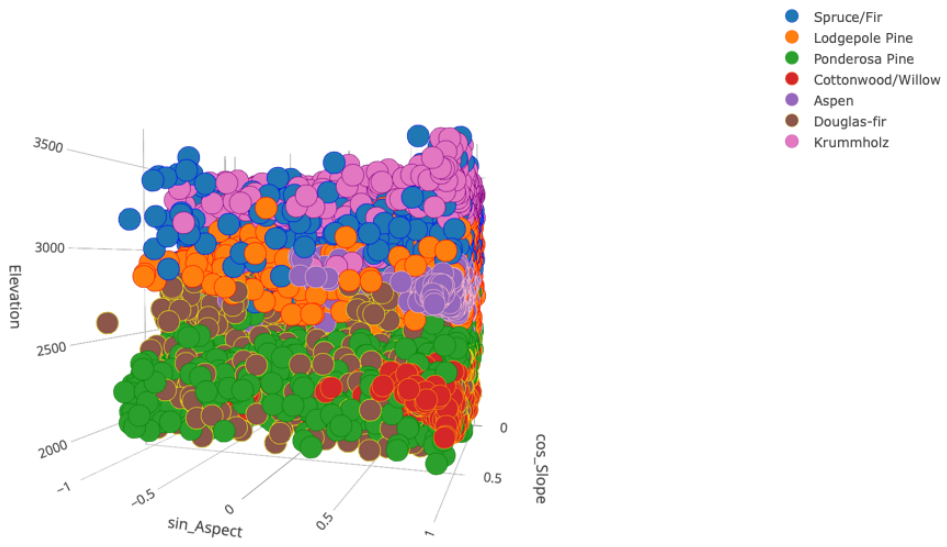



FIGURE 9 – Grapique de `cos_Slope`, `sin_Aspect` et `Elevation`

Enfin, nous prendrons les cosinus de nos données quantifiées en degrés et degrés azimuth. Nous aurons donc deux nouveaux attributs `cos_Slope` et `sin_Aspect` qui sont respectivement le cosinus de la pente des forêts et le sinus de l'orientation par rapport au nord. Une dernière étape consistera à normaliser les colonnes de nos données car elles n'ont pas du tout les mêmes échelles.

# 2 Test des méthodes

Nous étudierons les modèles suivant :
— Logistic Regression
— K-Nearest Neighbors
— Artificial Neural Network
— Random Forest
— Quadratic Discriminant Analysis

## 2.1 Logistic Regression

On cherche d'abord du côté de la régression logistique multinomiale. Comme elle est sensible aux trop grandes variances, on décide d'utiliser un paramètre de régularisation grâce à la pénalité *elastic net*. La loi a posteriori sachant que les données sont $(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^{(i)}, \mathbf{y}^{(i)})_{i=1}^{N}$ est une loi multinomiale

$$\Pr(\widehat{y} = k|\mathbf{w}; \mathbf{x}) = \frac{\exp(w_{0,k} + \mathbf{w}_k.\mathbf{x})}{\sum_{j=1}^{7} \exp(w_0 + \mathbf{w}_j.\mathbf{x})}$$

et sa fonction de coût est donc donnée par la log-vraisemblance

$$-\left[\frac{1}{N}\sum_{i=1}^{N}\left(\sum_{j=1}^{7} y_j^{(i)}(w_{0,j} + \mathbf{w}_j.\mathbf{x}^{(i)}) - \log\sum_{j=1}^{7} e^{w_{0,j}+\mathbf{w}_j.\mathbf{x}^{(i)}}\right)\right] + \lambda\left[\frac{1-\alpha}{2}\sum_{j=1}^{7}||\mathbf{w}_j||^2 + \alpha\sum_{j=1}^{7}||\mathbf{w}_j||\right]$$

où $N$ est le nombre d'observations. Le second terme de l'équation représente le terme de régularisation *elastic net*. Dans la pénalité *elastic net*, $\alpha$ varie de 0 à 1. Quand $\alpha = 0$, on a une régularisation L2, quand $\alpha = 1$, on a une régularisation L1 ("Lasso"). Quelques test montrent qu'il n'est pas nécessaire de faire varier alpha. Nous nous contenterons donc d'une régularisation L2. Des test préalables montrent que le terme de régularisation est de l'ordre de $10^{-3}$.

```
from sklearn.linear_model import LogisticRegression

log_params = [{'C':[0.1,0.2,0.3],'solver':['sag','newton-cg']}]
log = GridSearchCV(LogisticRegression(random_state=seed,penalty='l2',multi_class='multinomial'),
                   param_grid=log_params,cv=10,n_jobs=-1)

log.fit(x_train, y_train)
log.grid_scores_
```

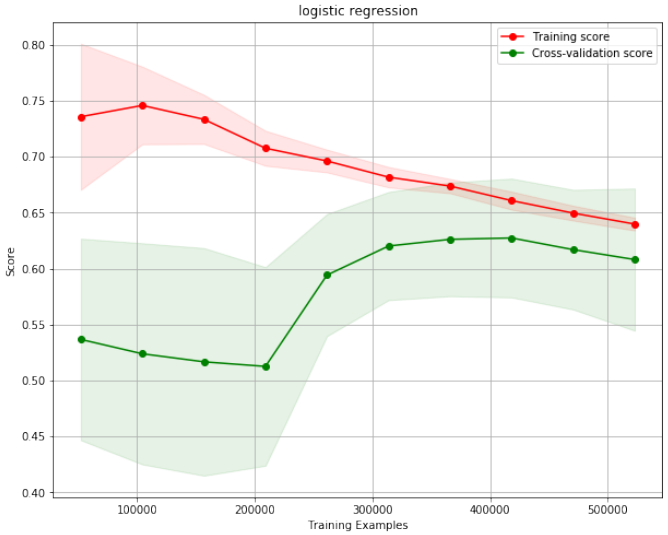
```
[mean: 0.63858, std: 0.00158, params: {'solver': 'sag', 'C': 0.1},
 mean: 0.63858, std: 0.00158, params: {'solver': 'newton-cg', 'C': 0.1},
 mean: 0.63836, std: 0.00154, params: {'solver': 'sag', 'C': 0.2},
 mean: 0.63836, std: 0.00155, params: {'solver': 'newton-cg', 'C': 0.2},
 mean: 0.63838, std: 0.00157, params: {'solver': 'sag', 'C': 0.3},
 mean: 0.63836, std: 0.00154, params: {'solver': 'newton-cg', 'C': 0.3}]
```

```
print ('Best accuracy obtained: {}'.format(log.best_score_))
print ('Parameters:')
for key, value in log.best_params_.items():
    print('\t{}:{}'.format(key,value))
```

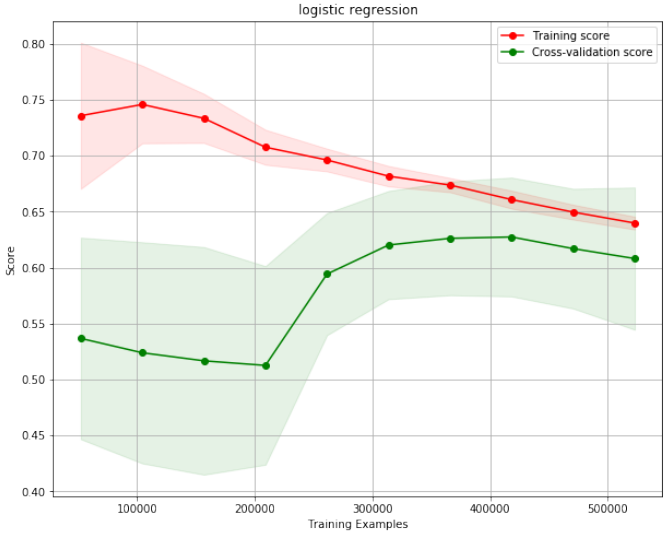
```
Best accuracy obtained: 0.6385785379190968
Parameters:
        solver:sag
        C:0.1
```

Premièrement, la régression logistique multinomiale ne converge pas sur les données brut. Ensuite, nous testons sur les données modifiées ce qui donne la learning curve de la figure 10(a). Nous arrivons à une accuracy de **63.8%**.

Cependant le classifier ne détecte pas la classe 5. Nous faisons donc un PCA en projetant les données sur une base plus propice à voir cette classe. Et le résultat est .....




(a) Sans PCA




(b) Avec PCA

FIGURE 10 – Learning Curve de la Logistic Regression sur les données modifiées

## 2.2 Random Forest

On fait varier les paramètres de notre premier modèle sur les données brutes. On commence par les paramètres `n_estimators`, qui est le nombre d'arbres, et `max_depth` qui la profondeur des arbres utilisés dans le modèle.
On constate par différents test [2] qu'aller au delà de 15 et 50 respectivement ne change pas grand chose.

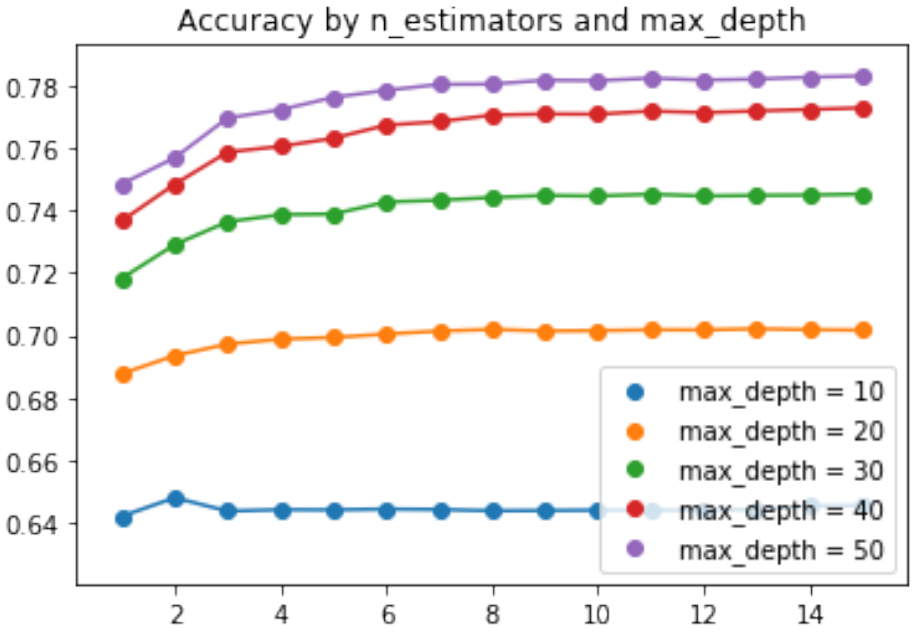



Figure 11 – Accuracy en fonction des paramètres `n_variables` et `max_depth`

# Références


[BD99] Blackard, Jock A. ; Dean, Denis J. : Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. In : *Computers and Electronics in Agriculture* (1999)

[CD14] Crain, Kevin ; David, Graham : Classifying Forest Cover Type using Cartographic Features. (2014)


2. cf. le notebook joint au rapport