Universite Pierre et Marie Curie
M2 de Statistiques

Année 2016 - 2017

# Projet d'Apprentissage : Prédire les clients qui ont réalisé des économies d'énergie

par

Tehe Yannick et Yahia Antony

# Introduction au challenge

Dans ce travail nous étudions l'influence d'un certain nombre de variables sur le fait qu'un individu fasse ou non des économies d'énergie.
Un individu $i$ quelconque de notre étude se décrit par un 110-uplet de la forme : $(i, x_{i,1}, ....., x_{i,108}, y_i)$. La valeur $x_{i,j}$ est la réponse à la question $X_j$ par l'individu $i$ et $y_i$ prend la valeur 1 si l'individu $i$ a fait des économies d'énergie et 0 dans le cas contraire.

Nous disposons d'un ensemble $E$ contenant des 110-uplets $(i, x_{i,1}, ....., x_{i,108}, y_i)$ de cardinal 85 528. Néanmoins il est important de préciser que certains $x_{i,j}$ ne sont pas renseignées. Nous leur attribuerons alors la caractéristique $NA$. L'ensemble $E$ correspond à nos données d'entraînement.

Nous disposons d'un ensemble $E^*$ contenant des 109-uplets de la forme $(i^*, x^*_{i,1}, ....., x^*_{i,108})$ avec $i^*$ l'individu numéro i de $E^*$. Les réponses au questionnaire sont définies de la même façon que sur $E$. On note aussi la présence de données manquantes. $E^*$ est donc notre échantillon test.

Le but de ce travail est d'utiliser l'information contenu dans $E$ afin de créer une fonction $f$ pouvant être vue comme un classifieur. $f : E^* \rightarrow \{0, 1\}$ . Nous devons entraîner des algorithmes afin de faire les prédictions les plus conformes à la réalité.
Le jeu étant défini, définissons aussi l'arbitre de ce jeu. Le score associé à $f$ sera simplement le pourcentage de bons classements que $f$ fait en prédisant sur $E^*$.

# Le Fil rouge de l'exposé

Nous débuterons cette étude par un pré-traitement de la base donnée. Cette étape est indispensable car pour l'instant les données dont nous disposons ne sont pas utilisable dans l'état. En effet nous observons que certaines réponses au questionnaire ont été mal saisies (problème d'encodage sur les réponses "True" ,"False" ) qui induisent un mauvais traitement par les différentes méthodes allant être utilisés par la suite.D'autre part la présence de valeurs manquantes dans les réponses au questionnaire induit un réel problème pour le fonctionnement de nos algorithmes de machine-learning.

Une fois cette étape terminée nous proposerons diverses techniques de classification qui nous ont semblé pertinentes.Pour chacune des méthodes nous détaillerons l'idée générale de l'algorithme ,pourquoi elle est adaptée pour le traitement de ce problème ainsi que les paramètres à régler pour obtenir les meilleures prédictions.enfin nous statuerons sur les performances de ces algorithmes en confrontant leurs résultats.

Pour terminer ce travail nous réfléchirons à des pistes d'amélioration. En effet nous nous demanderons comment il aurait été possible d'avoir de meilleurs résultats.

# Etude de la base de donnée

### Résolution du problème d'encodage des "True" et des "False"

Le travail de l'analyste statisticien ne se résume pas seulement à faire parler les données qu'on lui soumet, il doit veiller à ce que ces données soient de bonne qualité. Notamment vérifier s'il n'y a pas eu des problèmes de saisie qui pourraient fausser l'étude.Cette étape est indispensable car une fois que aurons confiance dans les données dont nous disposons nous pourrons mieux en user.

En recherchant les modalités de certaines réponses aux questionnaires nous nous sommes aperçus qu'un certain nombre de colonne contenait les quatre modalités suivantes : "False","false","True","true".Nous diagnostiquions un problème de saisie.Pour résoudre ce problème nous avons importé les données sous le logiciel R et ré-encodé par un algorithme les modalités "False" et "false" par 0 ; "True" et "true par 1. Ainsi les erreurs de saisies ont été corrigées

### Traitement des valeurs manquantes

A vu d'oeil nous avions remarqué que le jeu de données contenait un certain nombre de valeurs manquantes mais il nous était nécessaire d'évaluer leurs positions dans le jeu de donnée. En effet nous souhaitions identifier les variables comportant un nombre élevé de valeurs manquantes. Pour ce faire avec l'aide de la fonction Summary du logiciel R appliquée au tableau de données nous avons obtenu le pourcentage de valeurs manquantes par variable.Nous avons noté une forte hétérogénéité du nombre de valeurs manquantes par variables. Par exemple la variable "Q71" possédait 0.8% de valeurs manquantes tandis que la variable Q74 en possédait 39.5%.Nous avons alors regroupé les variables par tranche de 10% de valeurs manquantes. A partir de ce regroupement deux approches furent choisies:

En premier lieu il fut décidé de supprimer toutes les variables contenant plus de 20% de valeurs manquantes et remplacer dans chaque colonnes les valeurs manquantes à l'aide de la procédure suivante :

Soit $X_k$ la variable dans laquelle nous souhaitons remplacer les trous soit $I_k$ l'ensemble des indices de cette variable pour lesquelles il n'y a pas de valeurs manquantes.
supposons que $x_{i,k}$ ne soit pas renseignée on tire uniformément dans $I_k$ un indice j et on remplace $x_{i,k}$ par la valeur de $x_{j,k}$. A ce stade nous obtenons un tableau sans valeurs manquantes.

Une seconde procédure a été de garder les variables contenant moins de 30% de valeurs manquantes et de les remplacer par une méthode plus puissante.

Vu le grand nombre de variables et le nombre important de variables qualitatives un algorithme à questions nous paraissait adapté.Nous avions testé lors du challenge de modèle linéaire une bibliothèque de traitement de valeurs manquantes donnant des résultats satisfaisant : le Package "missForest" de R.

Nous avons naturellement pensé à l'utiliser dans ce problème pour le remplacement des

valeurs manquantes. Etant donnée la taille de nos données nous avons utilisé la fonction "missForest" de ce package avec un petit nombre d'arbre et d'itération pour optimiser la durée de compilation de celle ci. Par la suite nous nous sommes rendus compte que cette méthode donnait de meilleurs résultats de prédiction que la précédente.

**Transformation des variables qualitatives**

Pour pouvoir implémenter en python la plupart de nos algorithmes de machine learning nous devions obtenir un tableau avec uniquement des valeurs numériques. Pour cela il fallait transformer les variables qualitatives par la procédure suivante : Nous avons transformé chacune des variables qualitatives en variable de type factor en R puis converties leurs modalités en variables numériques.[0.2] En particulier la variable "C4" qui avait les modalités "IA" et "IB". Nous avons tranformé par ce procedé les IA en 1 et les IB en 2 .

# Modèles de prédiction

**Stratification**

Dans un premier temps nous souhaitions estimer les proportions de 0 et de 1 de l'échantillon test . Pour cela une première prédiction ne comportant que des 1 fut faite. Nous avons obtenu un score de 50% avec cette prédiction, nous en avions déduis que les 0 et les 1 sont en même quantité dans l'échantillon à prédire.Cependant cette proportion n'est pas la même que dans l'échantillon d'apprentissage dans lequel les proportions étaient environ de 80% de 0 et 20% de 0.

l'idée pour corriger cette différence et ne pas prédire de manière erronée a été de créer un nouvel échantillon d'entraînement de même stratification par la procédure suivante :
on partitionne l'ensemble des individus en deux classes:
-une première classe $C_0$ contenant uniquement les individus ayant pour réponse 0
-une deuxième $C_1$ contenant uniquement les individus ayant pour réponse 1
on tire ensuite la classe à pile ou face via une Bernoulli de paramètre 0.5 et on tire un individu de cette classe.On effectue un tirage sans remise .

A la fin de cette procédure on obtient une échantillon de même stratification que l'échantillon de test. A chaque compilation nous obtenons un nouvel échantillon d'apprentissage

```
#identification des lignes avec des 0 et des 1
ind0=[0]*85530
for i in range(85529):
    ind0[i]=i*(y_train[i]==0)

ind0
v1=[0]
v2=[3]
for i in range(85529):
    if ind0[i]>0:
        v1.append(ind0[i])
    elif (ind0[i]==0)&(i>3):
        v2.append(i)

ind0=v1
ind1=v2
len(ind1)

from random import uniform
v=[0]*len(ind1)
for i in range(len(ind1)):
    u=uniform(0,1)
    v[i]=(u<0.5)*ind0[i]+(u>=0.5)*ind1[i]

tir=v
```

Figure 1: script python pour la stratification $X_l$

**Prédiction par plus proche voisins**

l'idée de la classification par k plus proches voisins est de faire voter les plus proches voisins d'une observation. La classe de l'observation $X_i$ est déterminée en fonction de la classe majoritaire de ses k plus proches voisins. la notion de voisinage est basée sur des mesure de distance notamment la distance euclidienne dans le cadre de données numériques.

Dans le cas de notre étude nous avions affaire à beaucoup de variables qualitatives ne donnant pas de sens à une idée de distance euclidienne entre les différents individus.Pour traiter ce problème l'idée à été d'éclater en variables dichotomiques toutes nos variables qualitatives. Considérons par exemple une variable X à trois modalités A, B, C la transformation consiste alors à remplacer X par trois $X_A$, $X_B$,$X_C$ de même taille. Si nous considérons la nouvelle colonne $X_A$ elle contiendra des 1 là ou X avait des A et 0 ailleurs.

**Mise en oeuvre et résultats de la Classification par plus proches voisins**

Nous avons effectué la classification par plus proches voisins à l'aide de la fonction "KneighborsClassifier" de la bibliothèque sklearn.
L'idée pour nous a été de sélectionner à l'aide du *package* $sklearn.featureselection$ les meilleures variables pour faire une classification et de choisir k par une Cross-Validation Test-set sur l'échantillon constitué par les variables sélectionnées.

En effet nous avons découpé l'échantillon d'apprentissage en deux échantillons : un pour l'apprentissage faisant 2/3 de l'échantillon de départ et les 1/3 restant pour la validation et la prédiction . Nous avons ainsi après différents test déterminé le k optimal qui valait 10. malgré nos nombreux efforts pour le réglage des paramètres la classification par plus proches voisins nous a fourni des taux de bonnes prédictions tous inférieurs à 55% sur l'échantillon d'apprentissage et du même ordre sur l'échantillon Test.

**Prédiction par RandomForest**

la classification par RandomForest est un algorithme de classification par arbres de décision. En effet Le principe des algorithmes de classification arbre de décision est de partitionner les individus en produisant des groupes d'individus les plus homogènes possible du point de vue de la variable à prédire.

En particulier la RandomForest combine les décisions de plusieurs arbres entraînés sur un sous échantillon de E=(X,Y). Cette procédure s'effectue comme suit :

L'entrainement :

-on choisit un nombre d'arbre K (correspondant au paramètre nestimators dans la bibliothèque *sklearn* du langage python)

-Puis on tire aléatoirement et avec remise K échantillons de E=(X,Y)

-on construit ensuite sur chaque échantillon un arbre de décision

La prédiction:

on soumet nos nouvelles données à la prédiction de chaque arbre et l'on recueille le nombre de vote obtenue pour chaque on prédit ensuite la modalité qui a reçu le plus de voix. On parle de Tree-Bagging.

**Mise en oeuvre et résultats de la Classification par RandomForest**

Nous avons choisi la fonction RandomForstClassifier de sklearn qui nous a donné nos premiers résultats de classification satisfaisants sur l'échantillon d'apprentissage( de l'ordre 59% de taux de bonne classification).

Vu la grande instabilité des résultats obtenus par cette méthode nous nous sommes penchés sur le réglage de ses paramètres notamment le nombre d'arbres par une Cross-Validation(Test-set) elle même répétée plusieurs pour bien choisir celui-ci

```
# cross-validation sur le nombre d'arbre


modtest= RandomForestClassifier()

for i in [1,5,10,30,50,75,100,150,200,300]:
    modtest.set_params(n_estimators=i,max_features=80,max_depth =16,criterion="entropy")
    modtest.fit(x_train[tir[0:9000],:], y_train[tir[0:9000]])
    print (i, modtest.score(x_train[tir[9001:12000],:], y_train[tir[9001:12000]]))


#150 reste optimal pour le nombre d'arbre optimal
```

```
1 0.560853617873
5 0.594198066022
10 0.616872290764
30 0.638212737579
50 0.625208402801
75 0.634544848283
100 0.627542514171
150 0.634544848283
200 0.631543847949
300 0.636545515172
```

Figure 2: Résultats de la Cross-Validation Test-Set

Après le choix du nombre d'arbres, de l'observation et du bon sens nous ont permis de régler au mieux les autres paramètres. Par exemple le paramètre "Criterion" a été réglé à "entropy" quand les essais étaient faits sur des échantillons non équilibrés (un pourcentage de modalité plus élevé que l'autre) et celui de Gini étaient alterné avec "entropy" quand l'échantillon était équilibré.
Aussi le paramètre sample_weight permettait d'ajuster les poids des différentes modalités dans l'échantillon pour corriger les déséquilibres .
Après ces réglages la Randomforest nous fournissait des résultats de classification proche des 64% sur l'échantillon de validation et celui de test.

**Random forest avec les paramètres obtenus par cross-validation(test-set validation)**

```
]: #cross-valider param


from sklearn.ensemble import RandomForestClassifier
mod3= RandomForestClassifier(n_estimators=150,max_features=10,max_depth =84,criterion="entropy",class_weigt=[0.5,0.5])
mod3.fit(x_train[tir[0:9000],:], y_train[tir[0:9000]])

#évaluation sur un bloc plus petit (testset validation ):

print("Taux de bien classés sur les données de validation :")

print(mod3.score(x_train[tir[9001:15000],:], y_train[tir[9001:15000]]))
```

```
Taux de bien classés sur les données de validation :
0.645274212369
```

Figure 3: Taux bien classés après réglages des paramètres

Au bout de nombreux réglages le plus haut score obtenu sur l'échantillon de test via la randomForest a été de 64,5
Les algorithmes de prédictions par arbres de décisions nous fournissant nos meilleurs résultats jusqu'à lors et étant confrontés à une barrière de score.L'idée pour poursuivre a été d'utiliser une variante de la précédente(RandomForest) qui optimiserait son taux d'erreurs à chaque itération.
Nous avons donc opté pour une classification par GradientBoosting dont nous étayerons la procédure dans la suite.

**Classification par Boosting**

Le Boosting est aussi un algorithme de classification par arbres de décision mais a un fonctionnement différent de la randomforest pour la décision finale . la procedure est la suivante :
on construit K arbres en série c'est à dire que l'arbre K+1 a accès aux erreurs de l'arbre K et est construit pour les corriger.
La décision finale est prise comme dans le diagramme suivant :

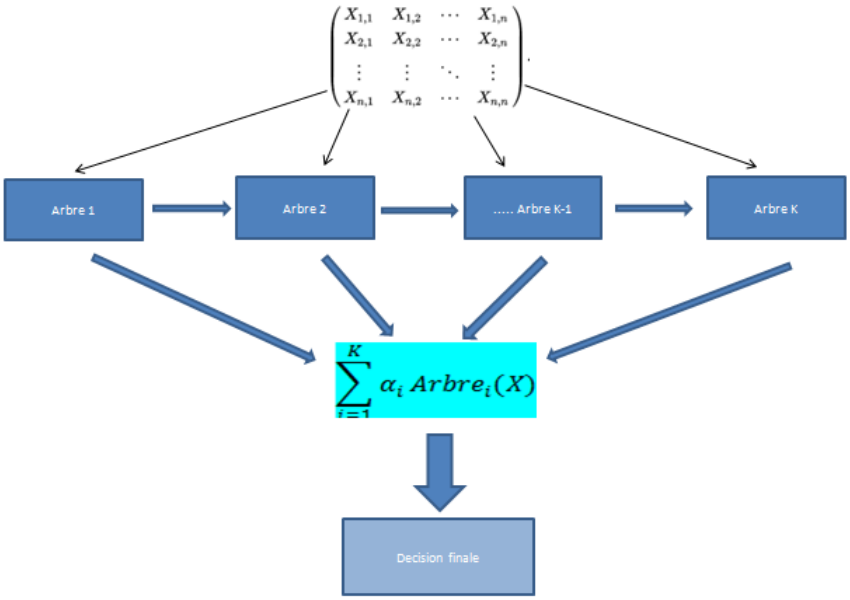


Figure 4: principe du Boosting

la décision est obtenu par une somme pondérée des différents arbres.Si c'est une classification avec les classes -1 et 1 la décision est le signe de la somme pondérée pour le cas 0 et 1 la décision est adaptée.

**Mise en oeuvre et résultats de la Classification par Boosting**

Nous avons choisi la fonction GradientboostingClassifier de la bibliotheque sklearn pour effectuer cette classification. Le choix des paramètres a été opéré par Validation croisée (validation test-set) après beaucoup de tests. le gradientBoosting bien paramétré nous a fourni des taux de classification de l'odre de 65% donc un peu meilleurs que celui de la randomforest.

**mise en oeuvre du Gradient Boosting pour booster la performance des arbres succéssifs :**

```
from sklearn.ensemble import GradientBoostingClassifier
mod5= GradientBoostingClassifier(n_estimators=100,max_features=50,max_depth =61,verbose=True)
mod5.fit(selection[tir[0:9000],:], y_train[tir[0:9000]])
#mod5_appr = mod5.predict(x_train[tir[0:8000],:])
print("Taux de bien classés sur les données d'apprentissage :")
#calcul du score sur deux echantillons de test (cross validation de la methode) avec
#la même strat

#Cross-validation des paramètres (non automatisée car de multiple test de paramètres non fortuits ont étés éffectués )
print(mod5.score(selection[tir[9001:12000],:], y_train[tir[9001:12000]]))
```

Figure 5: mise en oeuvre du gradient Boosting

### Blending de méthodes

Après avoir essayé plusieurs méthodes dont certaines ont été éliminées du plan de travail car non adaptées. nous avons obtenu un score proche de 65% pour nos meilleurs modèles . Des test nous ont montré que les prédictions de ces modèles étaient différentes pour un certain nombre de valeurs. Nous avons voulu alors à partir des prédictions de ces différents modèles construire une prédiction qui serait un mélange de ces prédictions à partir de règles de décisions logiques.
Le principe général a été de prédire les valeurs là ou une majorité de modèle étaient en accord et sur les valeur de désaccord de décider à partir du modèle le plus robuste ou les fixer à 1 ou 0. Des variantes astucieuses ont été essayées ce qui nous a menée à notre meilleur score qui de 65.9% donc environ 66%. nous proposons dans la figure suivante un exemple de ce qui a été essayé.

```
prm31=(mod3.set_params(max_depth=11,n_estimators=200)).predict(x_test)
prm32=(mod3.set_params(max_depth=15,n_estimators=175)).predict(x_test)
prm33=(mod3.set_params(max_depth=30,n_estimators=150)).predict(x_test)
prm34=(mod3.set_params(max_depth=50,n_estimators=150)).predict(x_test)
prm51=(mod5.set_params(max_depth=11,n_estimators=200)).predict(x_test)
prm52=(mod5.set_params(max_depth=15,n_estimators=175)).predict(x_test)
prm53=(mod5.set_params(max_depth=30,n_estimators=150)).predict(x_test)
prm54=(mod5.set_params(max_depth=50,n_estimators=150)).predict(x_test)
pred_meill=((pd.read_csv("0.659.csv",sep=",")).values)[:,1]
v=[0]*21260
v=prm31+prm32+prm33+prm34+prm51+prm52+prm53+prm54+pred_meill+pred

for i in range(len(prm31)):
    if v[i]>7:
        v[i]=1
    else :
        v[i]=pred_meill[i]
sum(v)
```

Figure 6: mise en oeuvre du blending

# Conclusion

Dans le cadre de cette étude, après un travail minutieux sur la base de donnée nous avons essayé plusieurs algorithmes de machine learning (Support à vaste Marge , Prédiction par plus proches voisins ,randomForest, Boosting ainsi que le blending des meilleurs modèles ). Nous nous sommes rendus compte que le Gradientboosting et le blending de méthodes ont étés les méthodes les plus efficaces après stratification des données d'entrainement et Cross-Validation pour ajuster les paramètres . Malgré tous ces éfforts nous n'avons pas pu passer le mur des 66%, nous nous sommes alors demandé pourquoi.plusieurs pistes de réponses s'offrent à nous : l'explication la plus vraisemblable est le traitement que nous avons réservé aux valeurs manquantes . En effet nous pensons que même les meilleurs algorithmes ne peuvent fournir de bons résultats quand ils sont entrainés sur des données de mauvaise qualité.
D'autre part le nombre élevé de variable a té une difficulté supplémentaire que nous aurions pu mieux traiter.