Régression Logistique

Modélisation prédictive

Ricco Rakotomalala

Université Lumière Lyon 2

Préambule

- La régression logistique est une méthode prédictive statistique ancienne, bien connue.
- Elle s'applique au classement binaire, mais peut être facilement élargie au cadre multiclasse (régression multinomiale notamment, mais pas que…).
- Elle connait un engouement ces dernières années en machine learning grâce à de « nouveaux » algorithmes de calcul (d'optimisation) et des librairies performantes.
- Les travaux autour de la régularisation rendent efficaces son utilisation dans le cadre de l'apprentissage en grandes dimensions (nb. Var. Expl. >> nb. Obsv.).
- Elle a des accointances avec les réseaux de neurones (perceptron simple). Elle est présentée sous cet angle dans certaines librairies (ex. Keras).

Plan

1. Fondements probabilistes
2. Estimation des coefficients de la régression
3. Importance des variables
4. Sélection de variables
5. Régularisation
6. Régression multiclasse
7. Conclusion
8. Références

Hypothèse fondamentale de la régression logistique

FONDEMENTS PROBABILISTES

Théorème de Bayes

Probabilités conditionnelles - On se place dans le cadre binaire Y ∈ {+, -}

Estimer la probabilité conditionnelle P(Y/X)

$$P(Y = k/X) = \frac{P(Y = k) \times P(X/Y = k)}{P(X)}$$

$$= \frac{P(Y = k) \times P(X/Y = k)}{\sum_{l=1}^{K} P(Y = l) \times P(X/Y = l)}$$

Dans le cas à 2 classes

$$\frac{P(Y=+/X)}{P(Y=-/X)} = \frac{P(Y=+)}{P(Y=-)} \times \frac{P(X/=+)}{P(X/Y=-)}$$

Cette quantité est facile à estimer à partir des données

Quelle hypothèse introduire pour rendre l'estimation de ce rapport possible ?

On parle de méthode semi-paramétrique parce qu'on ne fait pas d'hypothèses directement sur la distribution mais sur un rapport de distribution → l'hypothèse est moins restrictive.

Hypothèse fondamentale de la régression logistique

$$log\left[\frac{P(X/Y=+)}{P(X/Y=-)}\right] = b_0 + b_1X_1 + \cdots + b_pX_p$$

Cette hypothèse couvre une très large classe de distributions

- Loi normale (idem Analyse discriminante)
- Loi exponentielle
- Lois discrètes
- Loi gamma, Beta, Poisson
- Mélange de variables explicatives binaires (0/1) et numériques 

Moralité

1. Champ d'application théoriquement plus large que l'Analyse Discriminante (paramétrique)
2. Sa capacité à proposer une interprétation intéressante des coefficients est précieuse

Equation LOGIT

Autre écriture à une constante près

$$LOGIT = log\frac{\pi}{1-\pi} = a_0 + a_1X_1 + \cdots + a_pX_p$$

Où $\begin{cases} \pi = P(Y = +/X) \\ 1 - \pi = P(Y = -/X) \end{cases}$

- Rapport de probabilité appelé « odds » (rapport de chances) [ex. odds = 2, l'individu a 2 fois plus de chances d'être « + » que d'être « - »]
- -∞< LOGIT = log-odds < +∞, indique une propension à être positif, on peut le voir aussi comme une sorte de « score »

Première relecture de la règle d'affectation :

- Y = + ⇔ P(Y = + / X) > P(Y = - / X)
- Devient : Y = + ⇔ LOGIT > 0

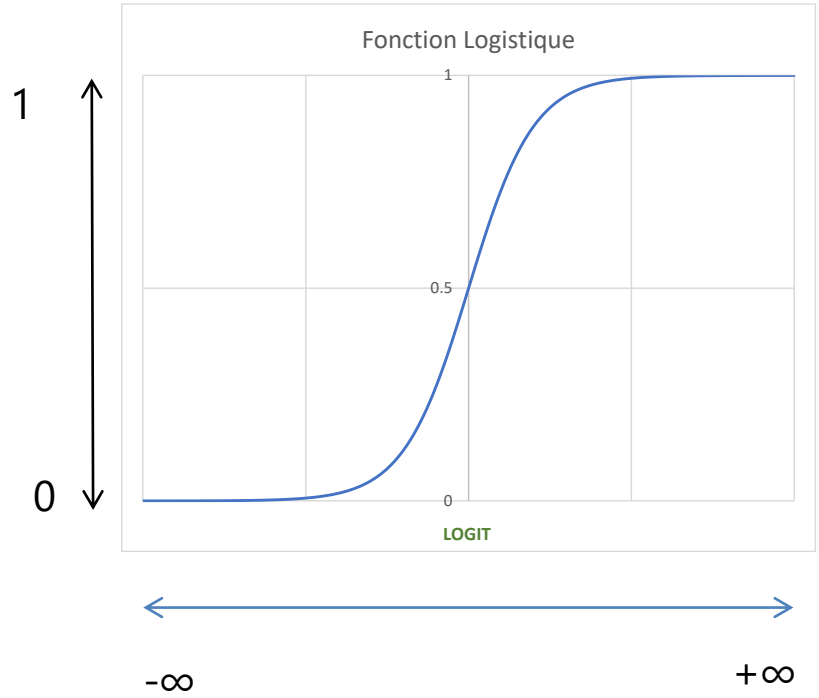
Fonction logistique

Fonction de transfert – Transformation logistique

Revenir à la probabilité d'appartenance π

$$\pi = \frac{e^{LOGIT}}{1 + e^{LOGIT}}$$

Ou bien $$\pi = \frac{1}{1 + e^{-LOGIT}}$$



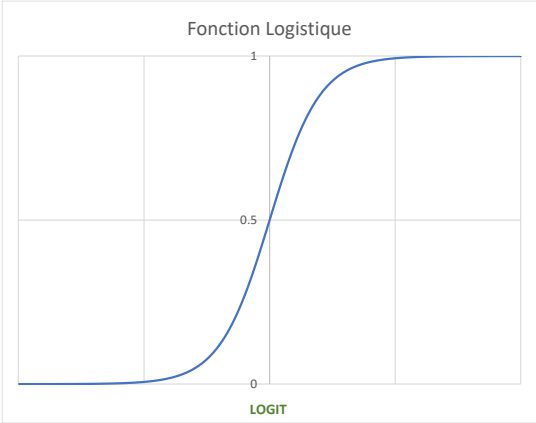
Avec π, on dispose d'un vrai « score » c.-à-d. une véritable estimation de la probabilité l'appartenance à la modalité cible ($0 \leq \pi \leq 1$).

Seconde relecture de la règle d'affectation :

- Y = + $\Leftrightarrow$ LOGIT > 0
- Devient Y = + $\Leftrightarrow$ π > 0.5

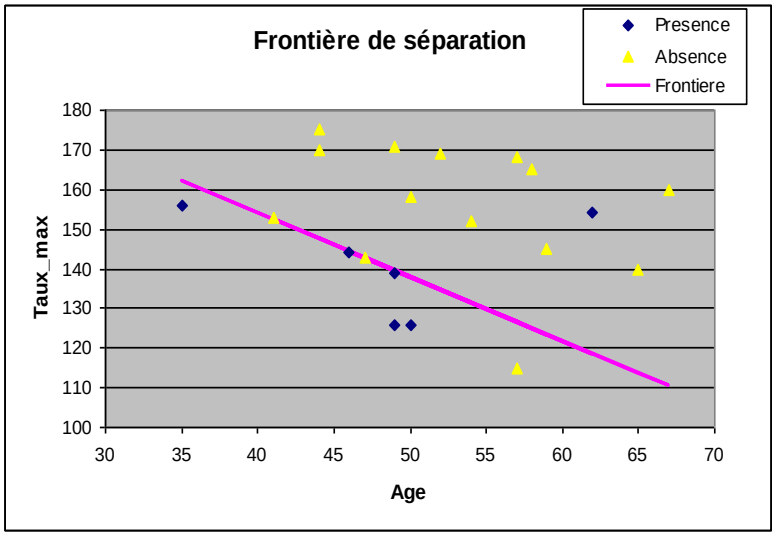
Régression logistique = Classifieur linéaire

La régression logistique est une régression non-linéaire dans le sens où on transforme une combinaison linéaire (le LOGIT) à l'aide d'une fonction de transfert non-linéaire (fonction logistique).



Mais...

Elle induit une séparation linéaire (on parle de classifieur linéaire – Linear classifier) dans l'espace de représentation (cf. Linear Models dans Scikit-learn par ex.)



Ex. de frontière. Cf. Livre, Section 11.3.

$$16.254 - 0.120\,Age - 0.074\,Taux_{Max} = 0$$

Exemple – Breast Cancer Wisconsin Dataset

Par défaut, la 2nde modalité par ordre alphabétique est « + » dans Scikit-Learn.

Base « benchmark » très connue. Prédire la nature {begnin, malignant} de cellules extraites de tumeurs à partir de leurs caractéristiques (taille, forme, épaisseur, …).

```
#info - var. toutes comprises entre [0,10]
#classe - binaire {begnin, malignant}
D.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 699 entries, 0 to 698
Data columns (total 10 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   clump        699 non-null    int64
 1   ucellsize    699 non-null    int64
 2   ucellshape   699 non-null    int64
 3   mgadhesion   699 non-null    int64
 4   sepics       699 non-null    int64
 5   bnuclei      699 non-null    int64
 6   bchromatin   699 non-null    int64
 7   normnucl     699 non-null    int64
 8   mitoses      699 non-null    int64
 9   classe       699 non-null    object
dtypes: int64(9), object(1)
memory usage: 54.7+ KB
```

```
#préparation - matrice X
X = D[D.columns[:-1]]

#vecteur y
y = D.classe
```

```
#scikit-learn -- version
import sklearn
sklearn.__version__
'0.24.2'

#régression logistique - param. par défaut
#sans préparation des données
from sklearn.linear_model import LogisticRegression
reg = LogisticRegression()
reg.fit(X,y)

#affichage des coefficients – coef_ est une matrice, d'où coef_[0]
pandas.DataFrame(reg.coef_[0],index=D.columns[:-1],columns=["coef"])
```

	coef
clump	0.523153
ucellsize	0.019895
ucellshape	0.322347
mgadhesion	0.234200
sepics	0.070728
bnuclei	0.397893
bchromatin	0.400704
normnucl	0.144779
mitoses	0.503595

```
#constante
reg.intercept_

array([-9.52436669])
```

Algorithmes d'entraînement

ESTIMATION DES COEFFICIENTS

Rappel sur le principe de la maximisation de la vraisemblance

$$LOGIT = a_0 + a_1 X_1 + \cdots + a_p X_p$$

Trouver les valeurs estimées des coefficients $\hat{a} = (\widehat{a_0}, \widehat{a_1}, \cdots, \widehat{a_p})$ qui « maximise leur vraisemblance » sur un échantillon.

Quel intérêt pour un EMV (estimateur du max. de vraisemblance) ?

- Asymptotiquement sans biais
- Variance minimale
- Asymptotiquement normal

⇨ Excellentes qualités. Outil privilégié pour l'inférence statistique (test, intervalle de confiance).

Estimation des coefficients via la maximisation de la vraisemblance

Le **modèle binomial** :

(1) Parce que Y est binaire {+, - } ou Y ∈ {1, 0} pour simplifier

(2) Si Y est d'une autre nature, on utiliserait d'autres modèles de distribution (ex. multinomial, …)

Pour l'individu n°i, on modélise la probabilité d'appartenance aux classes

$$\pi_i^{y_i} \times (1-\pi_i)^{(1-y_i)} \quad \begin{cases} y_i = 1 \Rightarrow \pi_i \\ y_i = 0 \Rightarrow 1-\pi_i \end{cases}$$

La vraisemblance pour un échantillon de taille n (observations i.i.d.)

$$L = \prod_{i=1}^{n} \pi_i^{y_i} \times (1-\pi_i)^{(1-y_i)}$$

La log-vraisemblance (log-likelihood)

$$LL = \sum_{i=1}^{n} y_i \times \log(\pi_i) + (1-y_i) \times \log(1-\pi_i)$$

Plus loin dans le critère à optimiser

Présenter l'apprentissage comme la minimisation d'une fonction de coût

Très en vogue dans les librairies actuelles (cf. Keras par exemple)

Log loss (logistic loss, binary cross-entropy)

$$J = -LL = -\sum_{i=1}^{n} y_i \times \log(\pi_i) + (1 - y_i) \times \log(1 - \pi_i)$$

Déviance

$$D = -2 \times LL$$

Fonction de perte quadratique

$$Q = \frac{1}{2}\sum_{i=1}^{n} (y_i - \pi_i)^2$$

Pourquoi pas ?

$y_i \in \{0, 1\}$

$0 \leq \pi_i \leq 1$

Algorithme d'optimisation de la fonction de coût

Newton-Raphson – Approche « classique »

Pas de solution directe, on passe par une approche itérative pour minimiser une fonction de coût J(.).

Pour passer de l'étape (t) à (t+1) :

$$a^{t+1} = a^t + H^{-1}(a) \times \nabla(a)$$

H(.) est la matrice hessienne, matrice des dérivées partielles secondes, de dim. (p + 1, p + 1)

$$H(a) = \left[\frac{\partial^2 J}{\partial a . \partial a'}\right]$$

∇(.) est le vecteur gradient, vecteur des dérivées partielles premières, de dim. (p + 1, 1)

$$\nabla(a) = \frac{\partial J}{\partial a}$$

« Hyperparamètres » de l'algorithme ⇨

- Nombre d'itérations max.
- Stagnation de la fonction de coût J(.)
- Stabilisation des estimations successives de â

Avantages

De H(.) peut être extraite la matrice des variances covariances des coefficients estimés â

Inconvénients

Temps de calcul et occupation mémoire de H(a) durant le processus d'optimisation sur les grandes volumétries (« n » grand) et surtout les données à très grandes dimension (« p » très grand)

Descente du gradient

Optimiser la fonction de coût sans avoir à passer par la matrice Hessienne (ou toute approximation de H(.), de toute manière de taille [p +1, p + 1], ex. BFGS)

Principe de la descente de gradient :

$$a^{t+1} = a^t + \eta \times \nabla(a)$$

Taux d'apprentissage (learning rate)

Intérêt. On s'affranchit de la matrice hessienne H(a) [gains en mémoire, en temps de calcul]. Le taux d'apprentissage (êta) permet de lisser les corrections.

Pour la régression logistique, le vecteur gradient s'écrit :

$$\nabla(a_j) = \frac{1}{n}\sum_{i=1}^{n}(x_{ij}) \times (y_i - \pi_i)$$

La correction à chaque étape, dépend :

(1) « Force » du signal envoyé par x_j (c'est mieux si les variables sont exprimées sur les mêmes unités, ou standardisées / normalisées).

(2) Amplitude de l'erreur

Descente de gradient – Taux d'apprentissage

η détermine la vitesse de convergence du processus d'apprentissage (trop faible, lenteur ; trop élevé, oscillation)

On peut améliorer le dispositif en le faisant évoluer au fil des itérations (élevé d'abord pour accélérer, faible à la fin pour plus de précision)

Exemple : Scikit-learn

class `sklearn.linear_model.` **`SGDClassifier`**(*loss='hinge', *, penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=1000, tol=0.001, shuffle=True, verbose=0, epsilon=0.1, n_jobs=None, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5, early_stopping=False, validation_fraction=0.1, n_iter_no_change=5, class_weight=None, warm_start=False, average=False*) [source]

learning_rate : *str, default='optimal'*
The learning rate schedule:

- 'constant': `eta = eta0`
- 'optimal': `eta = 1.0 / (alpha * (t + t0))` where t0 is chosen by a heuristic proposed by Leon Bottou.
- 'invscaling': `eta = eta0 / pow(t, power_t)`
- 'adaptive': eta = eta0, as long as the training keeps decreasing. Each time n_iter_no_change consecutive epochs fail to decrease the training loss by tol or fail to increase validation score by tol if early_stopping is True, the current learning rate is divided by 5.

  New in version 0.20: Added 'adaptive' option

eta0 : *double, default=0.0*
The initial learning rate for the 'constant', 'invscaling' or 'adaptive' schedules. The default value is 0.0 as eta0 is not used by the default schedule 'optimal'.

power_t : *double, default=0.5*
The exponent for inverse scaling learning rate [default 0.5].

On le détermine nous-même, mais la « bonne » valeur tient un peu du pifomètre.

$$\eta = \frac{1}{\alpha \times (t + t_0)}$$

« t » est le numéro d'itération
α est un paramètre supplémentaire
« t_0 », mystère, il faut voir la littérature

$$\eta = \frac{\eta_0}{t^{0.5}}$$

A vous de fixer η_0
0.5 étant également paramétrable

Descente de gradient stochastique

Si « online » ou « mini-batch », mieux vaut mélanger au hasard les individus avant de les faire passer.

Exemple : Scikit-learn

class `sklearn.linear_model.`**`SGDClassifier`**(*loss='hinge', *, penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=1000, tol=0.001, shuffle=True, verbose=0, epsilon=0.1, n_jobs=None, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5, early_stopping=False, validation_fraction=0.1, n_iter_no_change=5, class_weight=None, warm_start=False, average=False*) [source]

This estimator implements regularized linear models with stochastic gradient descent (SGD) learning: the gradient of the loss is estimated each sample at a time and the model is updated along the way with a decreasing strength schedule (aka learning rate). SGD allows minibatch (online/out-of-core) learning via the `partial_fit` method. For best results using the default learning rate schedule, the data should have zero mean and unit variance.

Descente de gradient classique (batch). Calcul du gradient après le passage de toutes les observations.

Online. Calcul du gradient au passage de chaque observation, correction des coefs., etc.

Mini-batch (traitement par lots). On fait passer un groupe d'obs. (nb = paramètre de l'algorithme), calcul du gradient, correction des coefs. Etc.

Plus rapide, meilleure convergence avec moins d'itérations sur la base entière (epochs), parce que les ajustements sont plus fins, plus fréquents, avec une forme d'échantillonnage.

(cf. aussi fit(.) de Keras)

Bilan - Algorithme d'optimisation

Le choix de l'algorithme joue clairement sur la vitesse d'apprentissage et la capacité à appréhender les grandes bases (voir « Régression Logistique sur les grandes bases »).

2 grandes familles :

(1) Newton-Raphson et ses variantes (BFGS, L-BFSG)

(2) Descente de gradient et ses variantes (ADAM, ADADELTA, …)

Exemple : KERAS

Available optimizers

- SGD
- RMSprop
- Adam
- Adadelta
- Adagrad
- Adamax
- Nadam
- Ftrl

Ex. Adam optimization is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments. According to Kingma et al., 2014, the method is "computationally efficient, has little memory requirement, invariant to diagonal rescaling of gradients, and is well suited for problems that are large in terms of data/parameters".

Exemple – Breast Cancer Dataset

```
#centrage réduction des variables
from sklearn.preprocessing import StandardScaler
std = StandardScaler()
Z = std.fit_transform(X)

#descente de gradient stochastique - reg. logistique
from sklearn.linear_model import SGDClassifier
sgd = SGDClassifier(loss='log')
sgd.fit(Z,y)

#coef.
print(pandas.DataFrame(sgd.coef_[0],index=D.columns[:-1],columns=["coef"]))
```

	coef
clump	1.495493
ucellsize	0.122822
ucellshape	0.624090
mgadhesion	1.982787
sepics	0.355997
bnuclei	1.876699
bchromatin	0.750848
normnucl	0.341101
mitoses	0.771152

Attention !

```
#diviser les coefs. par les écarts-type pour dé-standardiser
#un peu différents de « reg » parce qu'algo différent
#de LogisticRegression ('lbfgs' par défaut dans « sklearn 0.24 »)
res = sgd.coef_[0] / std.scale_
print(pandas.DataFrame(res,index=D.columns[:-1],columns=["coef"]))
```

	coef
clump	0.531499
ucellsize	0.040279
ucellshape	0.210146
mgadhesion	0.694901
sepics	0.160887
bnuclei	0.521317
bchromatin	0.308152
normnucl	0.111783
mitoses	0.449953

Quel intérêt alors ?

```
#temps de calcul LogistiRegression (lbfgs)
%timeit reg.fit(X,y)

8.34 ms ± 61.2 µs per loop (mean ± std. dev. of
7 runs, 100 loops each)

#temps de calcul de SGD
%timeit sgd.fit(Z,y)

1.97 ms ± 23.4 µs per loop (mean ± std. dev. of
7 runs, 1000 loops each)
```

Et tests de significativité

IMPORTANCE DES VARIABLES

Importance des variables – Coefficients standardisés

Problème. Les coefficients indiquent le sens (signe) et la force (valeur absolue) de l'impact des variables dans la régression (au sens de la variation du LOGIT). Mais telles quelles, elles ne permettent pas d'évaluer le mérite relatif des variables lorsque ces dernières sont exprimées dans des unités différentes.

Solution. Passer par les coefficients standardisés

- Soit en centrant et réduisant les explicatives présentées à l'algorithme.
- Soit en normalisant après coup les coefficients obtenus (cf. Livre, section 3.5.2)

Certains algorithmes de toute manière sont meilleurs lorsque les variables sont standardisées (ex. descente du gradient)

$$\beta_j = a_j \times \sigma_j$$

Mêmes valeurs des coefficients (β_j) que si on avait lancé la régression sur variables centrées et réduites.

Intérêt : nous disposons d'une « variation en écart-type » c.-à-d. impact de la variation d'un écart-type d'une variable sur le LOGIT, les (β_j) sont comparables.

Coefficients standardisés (exemple)

```
#coef. standardisés de LogisticRegression
regStd = LogisticRegression()
regStd.fit(Z,y)

#coefficients standardisés
beta = pandas.DataFrame(regStd.coef_[0],index=X.columns,columns=['beta'])
print(beta)
```

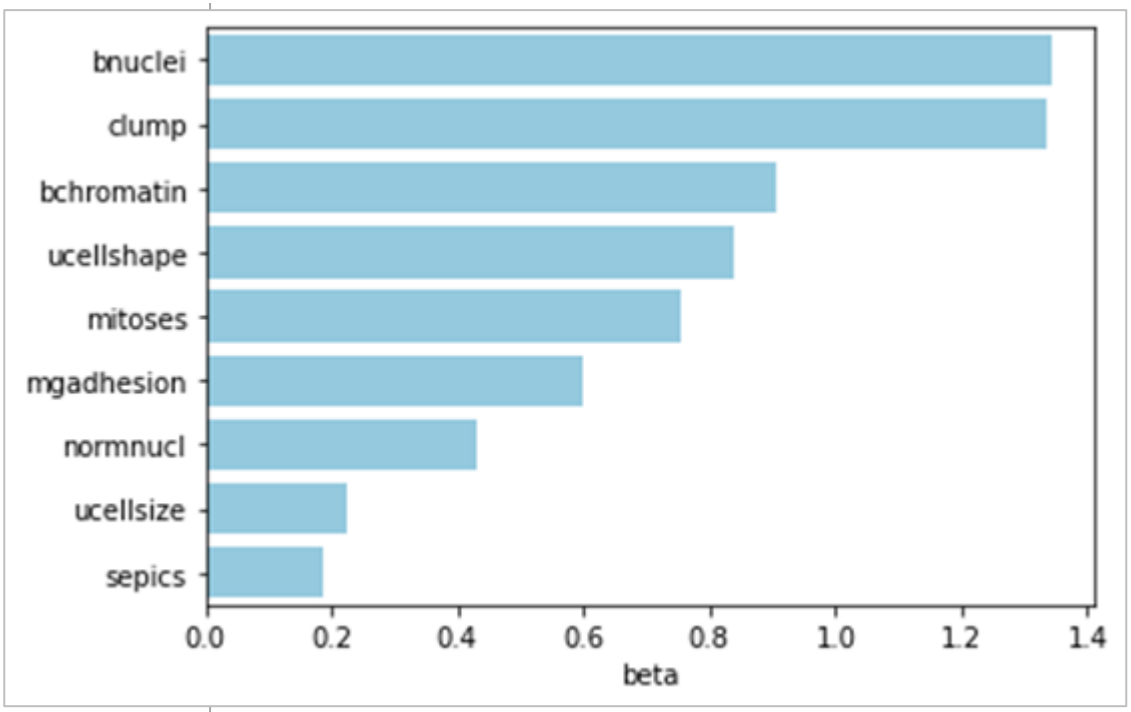
```
                beta
clump       1.336242
ucellsize   0.222793
ucellshape  0.837304
mgadhesion  0.598962
sepics      0.187225
bnuclei     1.341467
bchromatin  0.904485
normnucl    0.432049
mitoses     0.754527
```

```
#graphique après tri
beta_sorted = beta.sort_values(by='beta',ascending=False)
print(beta_sorted)
```

```
                beta
bnuclei     1.341467
clump       1.336242
bchromatin  0.904485
ucellshape  0.837304
mitoses     0.754527
mgadhesion  0.598962
normnucl    0.432049
ucellsize   0.222793
sepics      0.187225
```

```
import seaborn as sns
sns.barplot(x='beta',y=beta_sorted.index,data=beta_sorted,color='skyblue')
```

Le rôle des variables peut être hiérarchisé.



Test de significativité – Normalité asymptotique

$(\widehat{a_j})$ est asymptotiquement normal et, via la matrice hessienne, nous pouvons obtenir la matrice de var-covar des coefficients $(\widehat{\Omega}_{\hat{a}})$, dont la variance $(\hat{\sigma}^2_{\widehat{a_j}})$ sur la diagonale principale.

```
#régression logistique sous R
reg <- glm(classe ~ clump+ucellsize+ucellshape+mgadhesion+sepics+
bnuclei+bchromatin+normnucl+mitoses, data = D, family = binomial)
print(summary(reg))

##
## Coefficients (avec test à 5%, en rouge non-significatifs)
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -9.670977   1.051294  -9.199  < 2e-16 ***
## clump        0.531186   0.131825   4.029 5.59e-05 ***
## ucellsize    0.005841   0.186450   0.031  0.97501
## ucellshape   0.332579   0.207574   1.602  0.10911
## mgadhesion   0.240317   0.114826   2.093  0.03636 *
## sepics       0.069365   0.150751   0.460  0.64542
## bnuclei      0.400130   0.089381   4.477 7.58e-06 ***
## bchromatin   0.410683   0.156142   2.630  0.00853 **
## normnucl     0.144749   0.102275   1.415  0.15698
## mitoses      0.550705   0.302635   1.820  0.06880 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 900.53  on 698  degrees of freedom
## Residual deviance: 117.01  on 689  degrees of freedom
## AIC: 137.01
##
## Number of Fisher Scoring iterations: 8
```

Sachant que :

$$\frac{\widehat{a_j}}{\hat{\sigma}_{\widehat{a_j}}} \rightsquigarrow \mathcal{N}(0,1)$$

Test de significativité :

$$\begin{cases} H_0 : a_j = 0 \\ H_1 : a_j \neq 0 \end{cases}$$

Remarque : *On pourrait même étendre l'idée au test d'un groupe de coefficients, mais l'opération nécessite la manipulation, pas très facile, de $(\widehat{\Omega}_{\hat{a}})$ (cf. Livre, section 3.3).*

Test de significativité – Rapport de vraisemblance

```
#régression logistique avec l'ensemble des var.
reg <- glm(classe ~ clump+ucellsize+ucellshape+mgadhesion+sepics+bn
uclei+bchromatin+normnucl+mitoses, data = D, family = binomial)
# propriétés de l'objet
print(attributes(reg))

## $names
##  [1] "coefficients"      "residuals"         "fitted.values"
##  [4] "effects"           "R"                 "rank"
##  [7] "qr"                "family"            "linear.predictors"
## [10] "deviance"          "aic"               "null.deviance"
## [13] "iter"              "weights"           "prior.weights"
## [16] "df.residual"       "df.null"           "y"
## [19] "converged"         "boundary"          "model"
## [22] "call"              "formula"           "terms"
## [25] "data"              "offset"            "control"
## [28] "method"            "contrasts"         "xlevels"

#régression sans "mitoses"
reg2 <- glm(classe ~ clump+ucellsize+ucellshape+mgadhesion+sepics+b
nuclei+bchromatin+normnucl, data = D, family = binomial)

#écart des déviances
LR = reg2$deviance - reg$deviance
print(LR)

## 4.534219

#p-value
print(pchisq(LR,1,lower.tail=FALSE))

## 0.03322361
```

Principe du test de rapport de vraisemblance : Comparer les log-vraisemblance (les déviances plus précisément) avec (D_a) et sans (D_s) la variable à éprouver (cf. Livre, Section 3.2).

$$LR = D_s - D_a$$

$$LR \rightsquigarrow \chi^2(1)$$

« 1 » parce qu'un coef. en moins dans le 2nd modèle

Test de significativité globale – Rapport de vraisemblance

Le principe peut être étendu à l'évaluation du rôle d'un groupe de variables (cf. Livre, section 3.2.3), y compris du rôle de l'ensemble des variables ! Est-ce que la régression apporte de l'information ou pas (section 3.2.4) ?

La constante n'est pas concernée par le test parce qu'elle n'est pas associée à une variable.

$$\begin{cases} H_0 : a_1 = \cdots = a_p = 0 \\ H_1 : \exists j \,/\, a_j \neq 0 \end{cases} \Rightarrow LR \rightsquigarrow \chi^2(p)$$

```
#régression logistique
reg <- glm(classe ~ clump+ucellsize+ucellshape+mgadhesion+sepics+bn
uclei+bchromatin+normnucl+mitoses, data = D, family = binomial)
print(summary(reg))
##
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -9.670977   1.051294  -9.199  < 2e-16 ***
## clump        0.531186   0.131825   4.029 5.59e-05 ***
## ucellsize    0.005841   0.186450   0.031  0.97501
## ucellshape   0.332579   0.207574   1.602  0.10911
## mgadhesion   0.240317   0.114826   2.093  0.03636 *
## sepics       0.069365   0.150751   0.460  0.64542
## bnuclei      0.400130   0.089381   4.477 7.58e-06 ***
## bchromatin   0.410683   0.156142   2.630  0.00853 **
## normnucl     0.144749   0.102275   1.415  0.15698
## mitoses      0.550705   0.302635   1.820  0.06880 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 900.53  on 698  degrees of freedom
## Residual deviance: 117.01  on 689  degrees of freedom
```

LR = 900.53 – 117.01

LR = 783.52

p = 9

→ p-value ≈ 7.5^{E-163}

Stratégies de sélection des variables pertinentes

SÉLECTION DE VARIABLES

Approche « statistique »

Principe

- S 'appuie sur les tests de significativité, basé sur la normalité asymptotique
- Pilotée par l'hyperparamètre « α » (risque d'acceptation ou de rejet à tort d'une variable)

Stratégie (les plus simples)

- **Forward** : Départ = 0 variables. Recherche de la variable additionnelle la plus significative à chaque étape, ajout si p-value < α
- **Backward** : Départ = p variables. Recherche de la variable la moins pertinente, retrait si p-value > α

Backward → petites bases, Forward → grandes bases

Choix de la valeur de l'hyperparamètre « α », outil de pilotage

Risque de se tromper en acceptant ou en conservant une variable

- α faible, plus exigeant ➔ moins de variables sélectionnées
- α élevé, moins exigeant ➔ plus de variables sélectionnées

Exemple sous Tanagra

N°	Selected atts
1	clump
2	ucellshape
3	bnuclei
4	bchromatin

Backward, α = 0.01

Detailed results

N°	Current Reg.	Moved	Sol.1	Sol.2	Sol.3	Sol.4	Sol.5
1	AIC : 137.01 CHI-2 : 783.52 d.f. : 9 p-value : 0.0000	ucellsize Chi-2 : 0.001 p : 0.9750	ucellsize Chi-2 : 0.001 p : 0.9750	sepics Chi-2 : 0.212 p : 0.6454	normnucl Chi-2 : 2.003 p : 0.1570	ucellshape Chi-2 : 2.567 p : 0.1091	mitoses Chi-2 : 3.311 p : 0.0688
2	AIC : 135.01 CHI-2 : 783.51 d.f. : 8 p-value : 0.0000	sepics Chi-2 : 0.222 p : 0.6373	sepics Chi-2 : 0.222 p : 0.6373	normnucl Chi-2 : 2.074 p : 0.1498	mitoses Chi-2 : 3.350 p : 0.0672	ucellshape Chi-2 : 4.280 p : 0.0386	mgadhesion Chi-2 : 4.541 p : 0.0331
3	AIC : 133.23 CHI-2 : 783.29 d.f. : 7 p-value : 0.0000	normnucl Chi-2 : 2.451 p : 0.1174	normnucl Chi-2 : 2.451 p : 0.1174	mitoses Chi-2 : 3.449 p : 0.0633	ucellshape Chi-2 : 4.885 p : 0.0271	mgadhesion Chi-2 : 5.306 p : 0.0213	bchromatin Chi-2 : 7.433 p : 0.0064
4	AIC : 133.76 CHI-2 : 780.77 d.f. : 6 p-value : 0.0000	mitoses Chi-2 : 3.979 p : 0.0461	mitoses Chi-2 : 3.979 p : 0.0461	mgadhesion Chi-2 : 6.199 p : 0.0128	ucellshape Chi-2 : 8.674 p : 0.0032	bchromatin Chi-2 : 9.102 p : 0.0026	clump Chi-2 : 17.745 p : 0.0000
5	AIC : 137.57 CHI-2 : 774.96 d.f. : 5 p-value : 0.0000	mgadhesion Chi-2 : 6.524 p : 0.0106	mgadhesion Chi-2 : 6.524 p : 0.0106	ucellshape Chi-2 : 10.427 p : 0.0012	bchromatin Chi-2 : 10.847 p : 0.0010	bnuclei Chi-2 : 19.159 p : 0.0000	clump Chi-2 : 24.482 p : 0.0000
6	AIC : 142.45 CHI-2 : 768.08 d.f. : 4 p-value : 0.0000	-	ucellshape Chi-2 : 14.117 p : 0.0002	bchromatin Chi-2 : 14.576 p : 0.0001	clump Chi-2 : 23.345 p : 0.0000	bnuclei Chi-2 : 25.588 p : 0.0000	-

Modèle après sélection de variables

Attributes in the equation

Attribute	Coef.	Std-dev	Wald	Signif
constant	-8.998787	0.8748	105.8140	0.0000
clump	0.584057	0.1209	23.3450	0.0000
ucellshape	0.525738	0.1399	14.1170	0.0002
bnuclei	0.437393	0.0865	25.5878	0.0000
bchromatin	0.550018	0.1441	14.5758	0.0001

Et si on avait fixé α = 0.05 ?

Approche par optimisation

Principe

Trouver un arbitrage entre l'aptitude du modèle à coller aux données (matérialisée par la déviance D, qui diminue toujours à mesure que le nombre de variables augmente) et sa complexité (représenté par le nombre de coefficients estimés)

Formule

$$C = D + k \times (p + 1)$$

« $k = 2$ », critère Akaike (AIC)

« $k = \ln(n)$ », critère BIC (Schwartz). Moins de variables car pénalise plus la complexité dès lors que $\ln(n) > 2$

Stratégie

Parmi les plus simples et populaires, « Forward » vs. « Backward »

Arrêt dès lors que le critère ne décroît plus lors de l'ajout (ou le retrait) d'une variable

```r
#stepAIC de la librarie MASS - critère AIC avec k = 2
sel <- stepAIC(reg,direction="backward",k=2)

## Start:  AIC=137.01
## classe ~ clump + ucellsize + ucellshape + mgadhesion + sepics +
##     bnuclei + bchromatin + normnucl + mitoses
##
##              Df Deviance    AIC
## - ucellsize   1   117.01 135.01
## - sepics      1   117.22 135.22
## <none>            117.01 137.01
## - normnucl    1   119.06 137.06
## ...

## Step:  AIC=135.01
## classe ~ clump + ucellshape + mgadhesion + sepics + bnuclei +
##     bchromatin + normnucl + mitoses
##
##              Df Deviance    AIC
## - sepics      1   117.23 133.23
## <none>            117.01 135.01
## - normnucl    1   119.15 135.15
## ...

## Step:  AIC=133.23
## classe ~ clump + ucellshape + mgadhesion + bnuclei + bchromatin +
##     normnucl + mitoses
##
##              Df Deviance    AIC
## <none>            117.23 133.23
## - normnucl    1   119.76 133.76
## - mitoses     1   122.15 136.15
## - mgadhesion  1   122.69 136.69
## ...

#modèle final
print(sel)

## Coefficients:
## (Intercept)        clump   ucellshape   mgadhesion      bnuclei   bchromatin
##     -9.6063       0.5339       0.3538       0.2526       0.4040       0.4197
##    normnucl      mitoses
##      0.1554       0.5539
##
## Degrees of Freedom: 698 Total (i.e. Null);  691 Residual
## Null Deviance:       900.5
## Residual Deviance: 117.2     AIC: 133.2
```

Exemple sous R

Noter la décroissance de l'AIC et la règle d'arrêt de l'algorithme.

Recursive Feature Elimination (RFE)

Principe

Backward. Basé sur l'importance des variables (valeur absolue des coefficients pour la régression logistique), élimination de la moins importante à chaque étape.

Exemple : Scikit-learn

class `sklearn.feature_selection.` **RFE**(*estimator, *, n_features_to_select=None, step=1, verbose=0, importance_getter='auto'*)

Given an external estimator that assigns weights to features (e.g., the coefficients of a linear model), the goal of recursive feature elimination (RFE) is to select features by recursively considering smaller and smaller sets of features. First, the estimator is trained on the initial set of features and the importance of each feature is obtained either through any specific attribute or callable. Then, the least important features are pruned from current set of features. That procedure is recursively repeated on the pruned set until the desired number of features to select is eventually reached.

n_features_to_select : *int or float, default=None*

The number of features to select. If `None`, half of the features are selected. If integer, the parameter is the absolute number of features to select. If float between 0 and 1, it is the fraction of features to select.

Problèmes

- Les variables doivent être absolument exprimées sur la même échelle
- Comment fixer l'hyperparamètre « n_features_to_select » ?

REFCV (CV pour cross-validation)

Mieux que le RFE simple, mais plus de calculs aussi

(1) Estimation des performances (accuracy ou taux d'erreur) en validation croisée à chaque étape du RFE

(2) Choisir la configuration qui optimise la performance

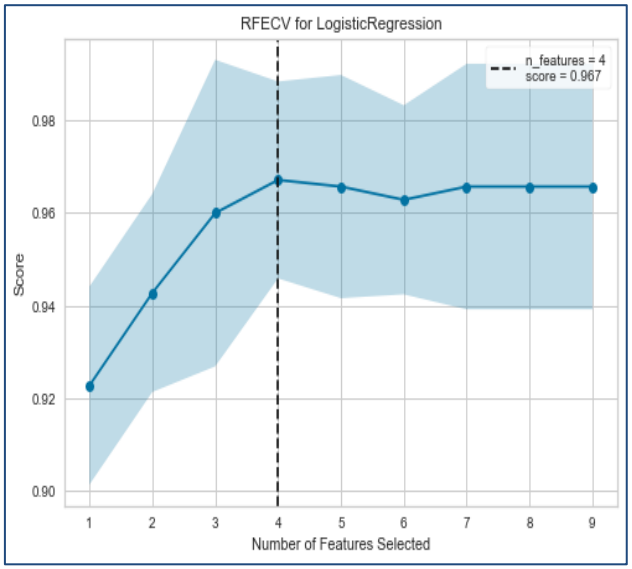
➔ « n_features_to_select » n'est plus un problème.

```
#avec la librairie yellowbrick
#travailler avec var. centrées et réduites
#pour être sûr que les coef. sont comparables
from yellowbrick.model_selection import RFECV
selRfe = RFECV(LogisticRegression(),cv=10,scoring='accuracy')
selRfe.fit(Z,y)
selRfe.show()
```

```
#liste des 4 variables sélectionnées
X.columns[selRfe.support_]

Index(['clump', 'ucellshape', 'bnuclei', 'bchromatin'], dtype='object')
```

Sélection basée sur l'accuracy (taux de reconnaissance)



4 variables sélectionnées

Solutions pour les fortes dimensionnalités (Ridge, Lasso, Elasticnet)

RÉGULARISATION

Problème des grandes dimensions

Problème

Malédiction de la dimensionnalité. Risque de sur-apprentissage (les données collent trop aux données) dans plusieurs situations (qui peuvent se cumuler) :

- p >> n (grande dimension, fréquent dans certains domaines ex. text mining)
- Beaucoup de variables non-pertinentes et/ou redondantes (colinéaires)

Conséquences

Les coefficients estimés sont erratiques, instables c.-à-d. si on rajoute ou retire quelques individus de la base d'apprentissage, les coefficients obtenus sont très différents.

Le modèle se généralise mal dans la population, effondrement des performances.

Solution

Régression pénalisée. Diriger (réguler) plus fermement la modélisation en imposant des contraintes sur les valeurs prises par les coefficients estimés de la régression (on parle aussi de « shrinkage », rétrécissement des valeurs pour les stabiliser).

Il faut apprendre des données, mais pas trop pour ne pas en être sur-dépendant !

Régression Ridge

A la sauce Scikit-learn (pas présentation usuelle, mais correspond à Ridge en pratique)

Pénalité sur la norme L2 des coefficients

(β_j – les coefficients doivent être directement comparables)

Pénalité sur la norme L2 des coefficients (hors constante)

Fait office de fonction de coût (au même titre que binary cross entropy, etc.). Attention $y_i \in \{+1, -1\}$ dans cette formulation.

Critère à optimiser pour la modélisation

$$\min_{\beta_0,\beta_1,\ldots,\beta_p} \frac{1}{2}\sum_{j=1}^{p}\beta_j^2 + C\sum_{i=1}^{n}\log(\exp(-y_i \times LOGIT_i) + 1)$$

« C » paramètre permettant d'arbitrer entre la pénalité et la « vraisemblance » dans le guidage de la modélisation

- C → ∞ , régression usuelle, non-pénalisée (aucun lissage)
- C → 0, la pénalité prend le pas sur l'apprentissage, les coefficients tendent tous vers 0 (trop lissée)

Exemple

```
#régression ridge 'l2', fortement régularisée - C = 0.05
regRidge = LogisticRegression(penalty='l2',C=0.05)
regRidge.fit(Z,y)

#coefs. Ridge
print(pandas.DataFrame(regRidge.coef_[0],index=X.columns,columns=(['ridge'])))

                ridge
clump        0.727676
ucellsize    0.451427
ucellshape   0.552636
mgadhesion   0.381987
sepics       0.294111
bnuclei      0.823869
bchromatin   0.551660
normnucl     0.388729
mitoses      0.319485

#à comparer avec param. par défaut
#qui étaient penalty='l2' et 'C = 1.0'
print(beta)
                beta
clump        1.336242
ucellsize    0.222793
ucellshape   0.837304
mgadhesion   0.598962
sepics       0.187225
bnuclei      1.341467
bchromatin   0.904485
normnucl     0.432049
mitoses      0.754527
```

(1) Les coefficients ont bien été « rétrécis »(sous pression) lorsqu'on régularise plus fortement.

(2) Forcément les constantes sont différentes également : « -1.029 » pour Ridge, « -1.197 » pour les param. par défaut (qui correspondait déjà un Ridge avec C = 1.0 en réalité).

(3) Est-ce que pour autant le modèle plus régularisé est plus performant ? Seul l'échantillon test peut nous le dire !!!

Régression Lasso

Least Absolute Shrinkage and Selection Operation

Inconvénient de Ridge : Les coefficients (β_j) tendent vers 0 avec la régularisation, mais ne sont jamais nuls ➔ pas de sélection de variables. Lasso permet d'y remédier avec une pénalité sur la norme L1 des coefficients.

Critère à optimiser pour la modélisation

$$\min_{\beta_0,\beta_1,\ldots,\beta_p} \frac{1}{2}\sum_{j=1}^{p}|\beta_j| + C\sum_{i=1}^{n}\log(\exp(-y_i \times LOGIT_i) + 1)$$

« C » paramètre de régularisation toujours, avec

- C → ∞ , régression usuelle, non-pénalisée
- C → 0, la pénalité prend le pas sur l'apprentissage, les coefficients tendent tous vers 0 ; parfois elles prennent la valeur 0 → sélection de variables

Exemple

```
#Lasso - nous oblige à changer d'algo d'optimisation
regLasso = LogisticRegression(penalty='l1',C=0.01,solver='saga')
regLasso.fit(Z,y)

#coefs. Lasso -- on observe l'effet "sélection de variables"
print(pandas.DataFrame(regLasso.coef_[0],index=X.columns,columns=(['lasso'])))
```

	lasso
clump	0.151262
ucellsize	0.387277
ucellshape	0.299874
mgadhesion	0.000000
sepics	0.000000
bnuclei	0.575756
bchromatin	0.079255
normnucl	0.000000
mitoses	0.000000

(1) L'algorithme par défaut « lbfgs » ne sait pas gérer la norme L1 (LASSO)

(2) On observe la sélection de variables avec les coefficients nuls. Ces variables n'interviennent pas dans le modèle à déployer.

(3) La constante est également différente ici (-0.7839)

(4) Est-ce que pour autant le modèle est plus performant ? Seul l'échantillon test peut nous le dire !!!

Inconvénients de Lasso : **(1)** Dans les grandes dimensions (p >> n), LASSO sélectionne au plus « n » variables mécaniquement, même si des variables additionnelles sont pertinentes. **(2)** Pour deux explicatives pertinentes (liées avec la cible) mais corrélées, LASSO en choisit une arbitrairement, au lieu de partager leurs influences (comme le ferait RIDGE).

Combiner les avantages de RIDGE et LASSO

$$\min_{\beta_0,\beta_1,\ldots,\beta_p} \frac{1-\rho}{2}\sum_{j=1}^{p}\beta_j^2 + \rho\sum_{j=1}^{p}|\beta_j| + C\sum_{i=1}^{n}\log(\exp(-y_i \times LOGIT_i)+1)$$

« ρ » un nouveau hyperparamètre (L1_RATIO) qui permet d'arbitrer entre RIDGE et LASSO :

- ρ = 1 ➔ LASSO
- ρ = 0 ➔ RIDGE

Exemple

```
#elasticnet -- choix de l'algo contraint également
#nouveau param. avec l1_ratio
regElastic = LogisticRegression(penalty='elasticnet',l1_ratio=0.5,
C=0.01,solver='saga')
regElastic.fit(Z,y)

#coefs. elasticnet -- on observe moins l'effet "sélection de variables"
print(pandas.DataFrame(regElastic.coef_[0],index=X.columns,columns=(['elastic'])))

             elastic
clump       0.324843
ucellsize   0.303607
ucellshape  0.319152
mgadhesion  0.132241
sepics      0.103166
bnuclei     0.480993
bchromatin  0.251785
normnucl    0.182070
mitoses     0.000000

#constante
regElastic.intercept_

array([-0.85358447])
```

(1) Nouveau hyperparamètre avec « L1_RATIO » pour faire la part entre RIDGE et LASSO

(2) Seule « mistoses » est évacuée cette fois-ci.

(3) La constante est également différente ici (-0.8536)

(4) Est-ce que pour autant le modèle est plus performant ? Seul l'échantillon test peut nous le dire !!!

Comment fixer les « bonnes » valeurs de tous ces hyperparamètres ?

Solution la plus simple et la plus usitée : essayer différentes valeurs et regarder les performances prédictives : soit avec un échantillon dédié (autre que l'échantillon test, on parle de « validation set »), soit en validation croisée (ex. GridSearchCV de Scikit-Learn).

```
#grille de recherche, ex. pour Ridge
#jouer sur le coefficient de régularisation
parametres = [{'C':[0.001,0.01,0.05,0.1,0.5,1.0,10.0,100.0]}]

#évaluation en validation croisée
#avec le critère accuracy
from sklearn.model_selection import GridSearchCV
grid = GridSearchCV(estimator=regRidge,param_grid=parametres,scoring='accuracy',cv=10)
grid.fit(Z,y)

#affichage
print(pandas.DataFrame.from_dict(grid.cv_results_).loc[:,['params','mean_test_score']])
          params  mean_test_score
0  {'C': 0.001}         0.898489
1   {'C': 0.01}         0.961408
2   {'C': 0.05}         0.964244
3    {'C': 0.1}         0.965673
4    {'C': 0.5}         0.965673
5    {'C': 1.0}         0.965673
6   {'C': 10.0}         0.965673
7  {'C': 100.0}         0.965673
```

Trop régularisé, le dispositif n'apprend pas assez des données.

Faible régularisation semble préférable sur cet exemple. Ce n'est pas étonnant au regard des caractéristiques de la base (n = 699 >> p = 9), peu de risque de surapprentissage (calculs numériquement stables du vecteur gradient durant le processus d'apprentissage).

Régression pour une variable cible à K modalités (classes) (K > 2)

RÉGRESSION MULTICLASSE

Sol1. – Régression multinomiale

Généralisation de la fonction de coût.

Multinomial Cross Entropy Loss

$$J = -\sum_{i=1}^{n}\sum_{k=1}^{K} y_{ik} \times \log(\pi_{ik})$$

Où y_{ik} prend la valeur 1 si y_i=k, 0 sinon

π_{ik} est obtenu à partir des fonctions des décision Z_k dont nous estimons les coefficients ($a_{k,j}$)

Fonctions de décision (1 fonction par classe)

$$Z_k = a_{k,0} + a_{k,1}X_1 + \cdots + a_{k,p}X_p \text{ , } k = 1, \ldots, K$$

Transformation softmax pour estimer les probabilités d'appartenance aux classes

$$\pi_{ik} = \frac{e^{z_{ik}}}{\sum_{k=1}^{K} e^{z_{ik}}}$$

$$\begin{cases} 0 \leq \pi_{ik} \leq 1 \\ \sum_{k=1}^{K} \pi_{ik} = 1 \end{cases}$$

Exemple – Iris Dataset

```
#importation des données
import pandas
D = pandas.read_excel("iris.xlsx")

#info – type ∈ 3 classes {setosa, versicolor, virginica}
D.info()

RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   sep_length   150 non-null    float64
 1   sep_width    150 non-null    float64
 2   pet_length   150 non-null    float64
 3   pet_width    150 non-null    float64
 4   type         150 non-null    object

#préparation - matrice X
X = D[D.columns[:-1]]

#vecteur y
y = D.type

#régression logistique multinomiale
#sans préparation des données
from sklearn.linear_model import LogisticRegression
reg1 = LogisticRegression(multi_class='multinomial')
reg1.fit(X,y)

#affichage des coefficients
import numpy
print(pandas.DataFrame(numpy.transpose(reg1.coef_),index=D.columns[:-1],columns=reg1.classes_))

              setosa  versicolor  virginica
sep_length -0.423397    0.534141  -0.110744
sep_width   0.961705   -0.317964  -0.643741
pet_length -2.519504   -0.205376   2.724880
pet_width  -1.085960   -0.939655   2.025615

#constantes
print(reg1.intercept_)
[  9.88124959   2.21932611 -12.1005757 ]
```

La base « IRIS » est archiconnue (archi courue) en machine learning.

Matrice avec 3 équations (3 classes = 3 fonctions de décision) à 4 variables explicatives.

Et 3 constantes aussi bien sûr.

Sol.2 -- Passer par les régressions binaires

Parce que parfois, « multinomial » implique un vecteur gradient et (éventuellement) une matrice hessienne à très forte dimensionnalité [ex. $K \times (p+1)$ pour le vecteur]

One vs. rest

Modéliser une classe (1) vs. les autres (0), K modèles à entraîner.

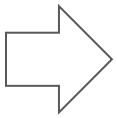
En classement, prédire la classe qui présente la probabilité d'appartenance la plus élevée. Attention, création artificielle de déséquilibre entre les classes.

One vs. one

Opposer les classes par paire, $\frac{K(K-1)}{2}$ modèles à entraîner.

En classement, on choisit la classe qui a remporté le plus de victoires.

Attention, chaque classifieur est construit avec moins d'observations.



Warning. Ces différentes approches (multinomial, one vs. rest, one vs. one) peuvent aboutir à des prédictions différentes sur les mêmes données.

Exemple – Iris Dataset

```
#régression logistique avec one vs. rest
#sans préparation des données
from sklearn.linear_model import LogisticRegression
reg2 = LogisticRegression(multi_class='ovr')
reg2.fit(X,y)

#affichage des coefficients
import numpy
print(pandas.DataFrame(numpy.transpose(reg2.coef_),index=D.columns[:-1],columns=reg2.classes_))
```

```
              setosa  versicolor  virginica
sep_length -0.445246   -0.185871  -0.394446
sep_width   0.894694   -2.114894  -0.513280
pet_length -2.325428    0.697706   2.930825
pet_width  -0.978692   -1.251396   2.417106
```

```
#constantes
print(reg2.intercept_)
```

```
[  6.72046653   5.54269649 -14.43117395]
```

(1) « setosa » contre les autres, « versicolor » contre les autres, etc.

(2) Les coefficients sont différents de la version « multinomiale ».

(3) Pour savoir si les 2 modèles (« multinomial » et « ovr ») classent de manière identique sur un jeu de données, il faut tester.

(4) Avec « one vs. one », le modèle ne s'exprime pas aussi simplement sous la forme de K fonctions de décision.

CONCLUSION

Régression logistique - Conclusion

- La régression logistique est une technique de machine learning à la fois basique (c'est la méthode que l'on apprend toujours dans un premier temps, cf. Coursera)
- Mais très populaire, notamment grâce aux perfectionnements « récents » (algorithmes d'apprentissage, stratégies pour la régularisation, …)
- Et l'émergence de librairies de calcul très performantes (sous Python notamment)

- Le traitement des explicatives qualitatives se fait simplement via le codage disjonctif des variables (« one hot encoding »).
- La généralisation à la régression multiclasse (variable cible à K > 2 modalités) peut se faire de différentes manières.

RÉFÉRENCES

Références

- Livre « Pratique de la régression logistique », septembre 2009.
- Diapos « Descente de gradient », avril 2018.
- Diapos « Ridge – Lasso – Elasticnet », mai 2018.
- Tutoriel « Régression Lasso sous Python », mai 2018.
- Tutoriel « Régression Ridge et Elasticnet sous R », mai 2018.
- Vidéo « One Hot Encoding pour la régression logistique », février 2021.