Algorithmes d'échantillonnage

Extraction d'échantillons à partir de fichiers à accès séquentiel ne tenant pas en mémoire centrale – Stratégie pour la modélisation prédictive

Ricco Rakotomalala

Echantillonnage dans le contexte big data

- Contexte big data : la taille des bases à traiter devient un enjeu essentiel
  - ➔ Chargement en mémoire de la totalité des données n'est plus possible parfois
  - ➔ Temps de traitements prohibitifs (lorsqu'ils sont possibles)
  - ➔ Développement des technologies big data (informatique distribuée essentiellement)

- Une stratégie alternative existe : travailler sur des échantillons.
  - ➔ Licite parce qu'il y a une forme de redondance (plus ou moins forte) dans les données
  - ➔ Traiter une fraction permet de généraliser (inférer) sur le reste de la base

- Deux enjeux importants :
  - ➔ Technique : échantillonner efficacement dans une base qu'on ne peut pas charger en mémoire
  - ➔ Stratégique : S'assurer une qualité de modélisation équivalente à celle du modèle construit sur la base complète (dans le cadre de l'apprentissage supervisé)

Configuration technique

Base initiale

- Fichier texte CSV (comma separated value)
- Très gros volume, ne tient pas en mémoire centrale
- Accès séquentiel
- Individus non pondérés

Objectif – Extraction d'un échantillon de taille « n »

- Les individus doivent avoir une probabilité identique d'intégrer l'échantillon
- Traitement en une seule passe sur les données
- Production de l'échantillon directement dans un fichier à part ou conservation de l'échantillon en mémoire
- Facilement extensible pour la partition d'une base en ensembles d'apprentissage et de test pour la modélisation prédictive

Plan

1. Echantillonnage lorsque la taille N est connue
2. Reservoir sampling (N inconnu)
3. Stratégies pour l'apprentissage supervisé
4. Conclusion

Technique d'échantillonnage pour « N » connu

La taille « N » de la base de sondage est connue. Soit fournie a priori, soit parce qu'il a été possible d'effectuer une passe préalable (pas trop coûteuse en ressources) sur les données.

Utilisation d'un index

```
sample(1:N,n,replace=F) # par exemple sous R
numpy.random.randint(1,N,n) #par exemple sous Python
```

- Générer un vecteur d'index IDX de taille « n », extrait entre 1 et N sans répétition
- Ouvrir le fichier source en lecture # pour une lecture ligne par ligne (séquentielle)
- i := 0 #numéro de ligne
- TANT QUE pas fin de fichier
  - Lire une ligne
  - i := i + 1
  - SI i %in% IDX
    - ALORS Charger ligne en mémoire ou l'écrire dans un fichier de sortie
  - FIN SI
- FIN TANT QUE

Attention à « i », selon que la première correspond aux noms de variables ou non.

L'opérateur %in% peut être coûteux. En triant les indices de manière croissante, on peut imaginer une gestion plus efficace, et même s'arrêter avant d'atteindre la fin du fichier.

Création d'un index

Comment générer les index si l'on ne dispose pas d'une fonction dédiée ?

- Générer N valeurs aléatoires
- Récupérer les indices des valeurs triées
- Récupérer les n premiers indices

Ce tri de N valeurs peut être coûteux en temps de calcul.

```r
#valeurs aléatoires
N <- 1000
v <- runif(N)

#argument du tri
argument <- order(v)

#prendre les n premiers indices
n <- 10
idx <- argument[1:n]
print(idx)
```

Code R

```python
#valeurs aléatoires
N = 1000
v = numpy.random.random(N)

#argument de tri
argument = numpy.argsort(v)

#prendre les n premiers indices
n = 10
idx = argument[:n]
print(idx)
```

Code Python

Extraction sans utilisation d'un index – Méthode de sélection rejet

On peut s'affranchir du vecteur d'index avec une gestion fine de la probabilité d'inclusion et un générateur de nombres aléatoires.

- Entrée : N, n et fichier source
- Ouvrir le fichier source # lecture séquentielle
- TANT QUE n > 0
  - Lire une ligne
  - SI N * ALEA() <= n
    - ALORS
      - Ecrire ligne dans sortie
      - n := n - 1
  - FIN SI
  - N := N – 1 # péréquation de la probabilité d'inclusion
- FIN TANT QUE

ALEA() générateur de valeurs aléatoires U(0,1)

Le dispositif fonctionne même si N = n

Pour cette approche, les lignes dans l'échantillon sont forcément dans le même ordre que dans la source initiale.

Extraction sans utilisation d'un index (Python)

```
#vecteur de valeurs - représente le fichier source
N = 1000
source = numpy.arange(N)

#n
n = 10

#boucler tant qu'il y a des valeurs à extraire
i = -1
while (n > 0):
    #simule la lecture de la ligne
    i = i + 1
    #test d'inclusion
    if (N * numpy.random.random() <= n):
        #valeur récupérée
        print(source[i])
        #un élément en moins à extraire
        n = n - 1
    #
    N = N - 1
```

Pour les besoins de l'illustration, la source ici est indicée. Mais la transposition à un fichier à accès séquentiel est facile (« i » est incrémentée seulement d'une valeur à chaque passage dans la boucle dans notre exemple)

Pour une lecture ligne par ligne dans un fichier texte, voir readline() [Les fichiers sous Python, page 7]

Extraction sans utilisation d'un index (R)

Pour une lecture ligne par ligne dans un fichier texte sous R, voir readLines() [Read Text Lines from a Connection]

```
#vecteur de valeurs
N <- 1000
source <- 1:N

#n
n <- 10
#échantillon initialisation
echantillon <- c()

#boucler
i = 0
while (n > 0){
  #id pour lecture
  i <- i + 1
  #test d'inclusion
  if (N * runif(1) <= n){
    echantillon <- c(echantillon, source[i])
    #un de moins à extraire
    n <- n - 1
  }
  #un élément de moins de la source
  #à traiter
  N <- N - 1
}
#affichage
print(echantillon)
```

Reservoir sampling

La taille « N » de la base de sondage est inconnue ou trop coûteuse à acquérir

Réservoir sampling (Algorithm R – Vitter, 1985)

Principe :

1. charger les « n » premiers individus dans une collection
2. mettre à jour cette collection au fur et à mesure de la lecture des individus restants

Contrainte :

- Maintenir la collection en mémoire
- Ou tout du moins dans une structure où un accès indicé est possible.

```
• Entrée : n et fichier source
• Sortie : C un collection avec n lignes
• Pour i:= 0 à n-1 ; C[i]:= Lire une ligne
• t := n
• TANT QUE pas fin de fichier
    • Lire une ligne
    • t := t + 1
    • k := TRUNC(t * ALEA()) # 0 ≤ k ≤ t-1
    • SI k < n
        • ALORS C[k]:= ligne
    • FIN SI
• FIN TANT QUE
```

Initialisation : remplissage du « réservoir » avec les n premières lignes du fichier source.

Permet une péréquation de la probabilité d'être dans le réservoir au fil des intégrations / retraits.

Mise à jour du « réservoir » à l'indice n°k.

Chaque item a une probabilité d'inclusion $\frac{n}{card(source)}$ (Wikipédia, Reservoir Sampling)

Réservoir sampling (Python)

Ici aussi, pour les besoins de l'illustration, la source est indicée. Mais la transposition à un fichier à accès séquentiel est facile (« i » est incrémentée seulement d'une valeur à chaque passage dans la boucle dans notre exemple)

L'ordre des lignes n'est pas préservée dans cette approche.

```
import math, numpy

#vecteur de valeurs - représente le fichier source
N = 1000
source = numpy.arange(N)

#collection à remplir
n = 10
collection = numpy.zeros(n)

#remplissage du réservoir
for i in range(n):
    collection[i] = source[i]

#initialisation
t = n

#tant que pas fin de source
for i in range(n,N):
    t = t + 1
    k = math.floor(t * numpy.random.random())
    if (k < n):
        collection[k] = source[i]
    #
#
print(collection)
```

Réservoir sampling (R)

La similitude avec le code Python n'a rien de fortuit.

Dans ce code R, la boucle **for** sur la source n'est pas très efficace. Mais elle est inévitable lorsque l'on devra la transposer en une lecture ligne par ligne dans un fichier à accès séquentiel.

```
#vecteur de valeurs
N <- 1000
source <- 1:N

#n
n <- 10

#réservoir initialisation
collection <- source[1:n]

#initialisation
t = n + 1

#tant que pas fin de source
for (i in (n+1):N){
  t <- t + 1
  k <- floor(t * runif(1))
  if (k <= n){
    collection[k] <- source[i]
  }
}

#affichage
print(collection)
```

Stratégie d'échantillonnage pour l'apprentissage supervisé

Déterminer la taille suffisante d'échantillon (n*) pour obtenir un modèle aussi performant que celui qui aurait été élaboré sur la totalité de la base

Stratégie d'échantillonnage pour l'apprentissage supervisé

Objectif

- Construire le modèle sur un échantillon
- Avec une qualité prédictive équivalente au modèle qui aurait été élaboré sur la totalité des données

➔ Gain de temps, plus de possibilités d'optimisation des paramètres.

➔ Parfois seule solution qui rend les calculs possibles.

Démarche

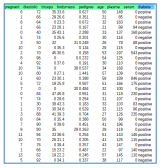
- Choisir un critère d'évaluation du modèle
- Démarrer avec une taille d'échantillon initiale, modéliser, évaluer
- Augmenter graduellement la taille de l'échantillon jusqu'à ce que le critère stagne. C'est un signe que l'on épuisé l'information « utile » des données.
- Paramètres : taille initiale, nombre d'observations additionnelles à chaque étape.

Solution 1 – Random sampling – Augmentation graduelle

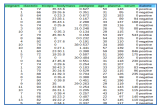
Augmentation de la taille de l'échantillon ($n = n + n_a$)

(pourrait être un échantillon déjà)

Dataset pour la modélisation

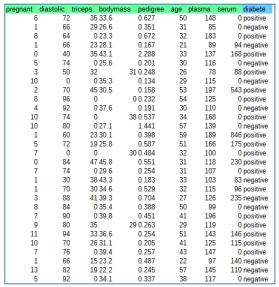


Echantillon, ensemble d'apprentissage (n)



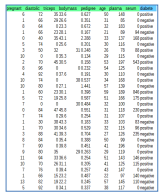
Construction de la fonction f(.) à partir de l'échantillon

$Y = f(X1, X2, \ldots)$



Dataset

Application du modèle (prédiction) sur l'ensemble de test



Dataset pour l'évaluation

(on pourrait choisir de ne travailler que sur un échantillon pour le test)

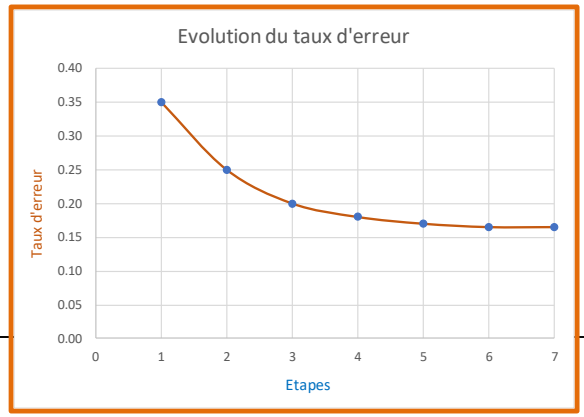
$(Y, \hat{Y})$

Y : valeurs observées

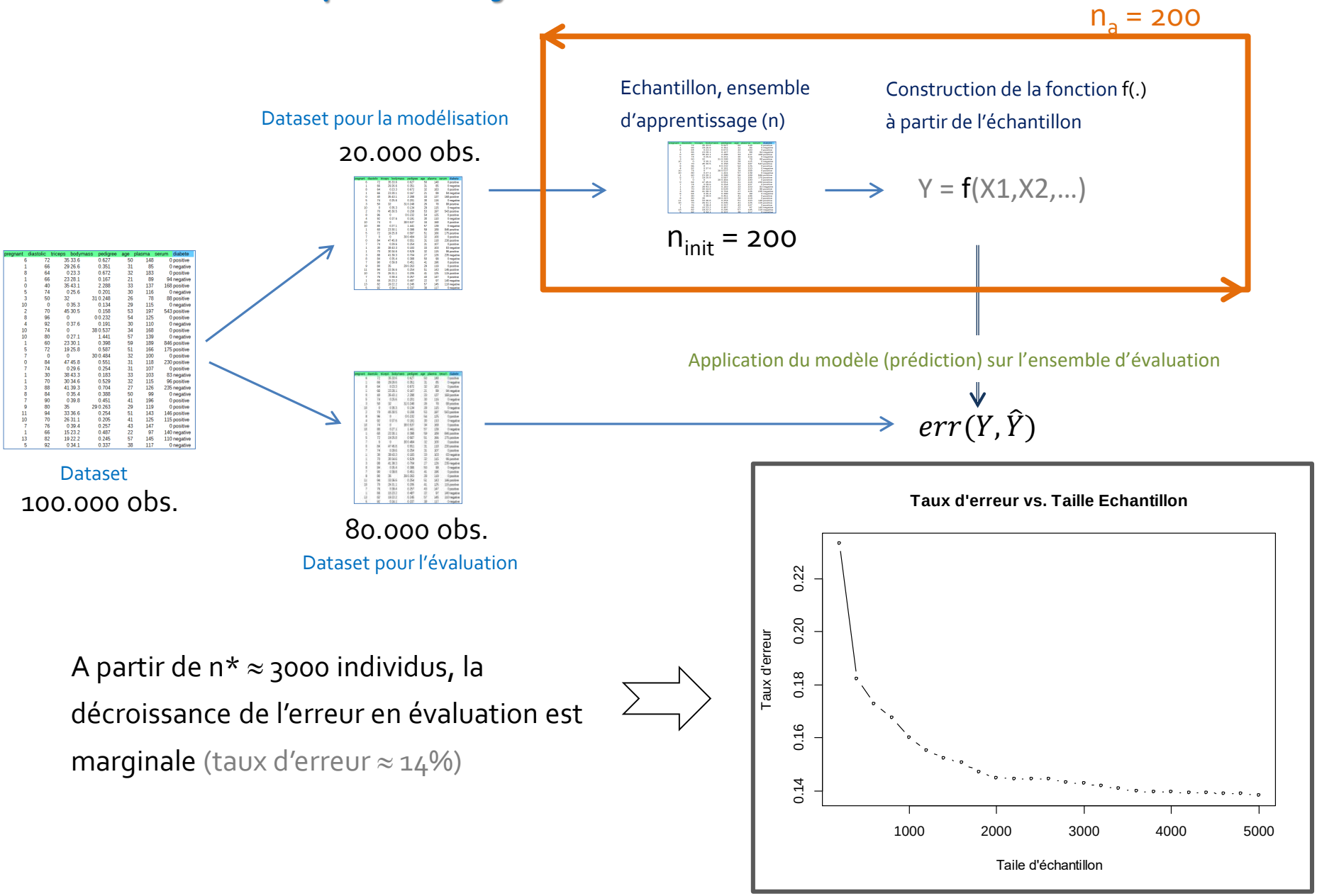
Y^ : valeurs prédites par f(.)

Mesures de **performances** par confrontation entre Y et Y^ (ex. taux d'erreur)

Courbe d'évolution du critère d'évaluation

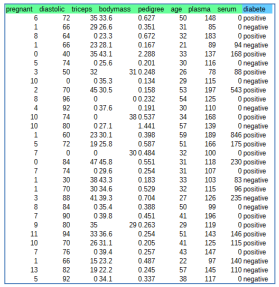


Solution 1 – Exemple : Analyse discriminante + Waveform

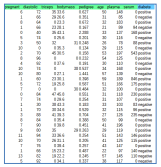


A partir de n* ≈ 3000 individus, la décroissance de l'erreur en évaluation est marginale (taux d'erreur ≈ 14%)

Solution 2 – Windowing

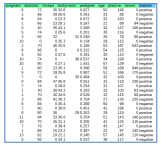


Dataset



Dataset pour la modélisation
(soit la totalité, soit un échantillon)

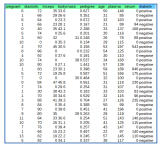
Ensemble d'apprentissage



Construction de la fonction f(.) à partir de l'échantillon

$Y = f(X1, X2, \ldots)$

Application du modèle (prédiction) sur l'ensemble de test



Ensemble de test

$(Y, \hat{Y})$

Identifier les individus mal classés

Les rajouter à l'ensemble d'apprentissage

Les individus bien classés à l'étape courante deviennent l'ensemble de test

Arrêt du processus, lorsque

- Tous les individus de l'ensemble de test sont bien classés
- L'erreur ne décroît plus sur un ensemble d'évaluation à part

Solution 2 – Windowing - Exemple

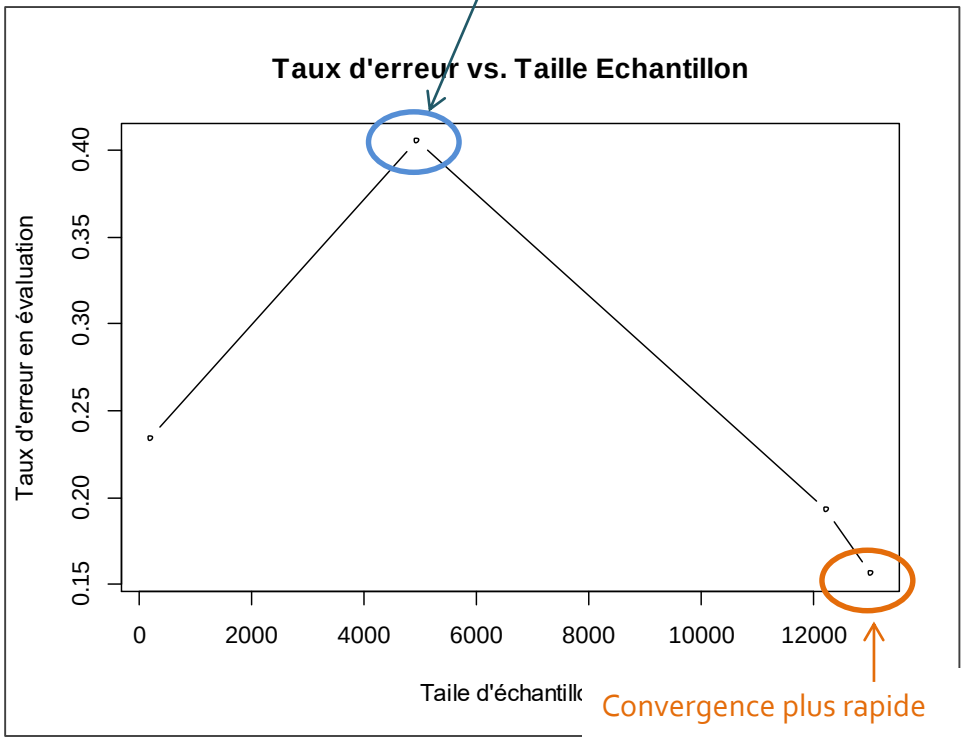
Dégradation à la première itération puisqu'on a introduit toutes les observations à problème (au détriment des données « utiles »).

Base de travail = 20000

Taille initiale échantillon = 200

Taille base évaluation = 80000

Non requis habituellement pour le windowing, mais nous l'utilisons pour suivre de taux d'erreur en généralisation.



Convergence plus rapide (moins d'itérations). Mais taille d'échantillon plus grande.

Constaté habituellement

- Convergence plus rapide (ici arrêt parce que tous les individus sont bien classés sur le reste de la base de travail). Taille de la base finale n* = 12570
- Mais le modèle final est moins bon parfois (à taille égale par rapport à la solution 1) lorsque la base est bruitée (la méthode peut mettre excessivement l'accent sur le bruit et/ou sur les outliers)
- Bon comportement en revanche sur les bases « propres » (cf. Hoeksma)

Conclusion

Algorithmes d'échantillonnage rapides

- Des variantes plus rapides existent pourvu que l'on puisse « sauter » plusieurs enregistrements vers l'avant lors de la sélection des lignes (soit parce que la source est en accès indexé, soit parce que le saut est moins coûteux que la lecture ligne par ligne)
- Des variantes existent pour l'appréhension des pondérations des individus (tirage avec des probabilités inégales)
- Des algorithmes distribués existent pour tirer parti des clusters de machines

Stratégies d'échantillonnage pour la modélisation prédictive

- L'expérimentation menée ici n'a pas valeur de preuve (une base, un algo de machine learning)
- Quoiqu'il en soit, travailler sur des échantillon peut être bénéfique
- La taille optimale n* de l'échantillon dépend de la base traitée et de la méthode de machine learning utilisée : on doit passer par l'expérimentation.

Références

Références

- Deville J.C., Grosbras J.M., « Chapitre 10 - Algorithmes de tirage », in « Les sondages », Droesbeke J.J., Fichet B., Tassi P., editeurs, Economica, 1988.
- Vitter J.S., « Random Sampling with a reservoir », in ACM Transactions on Mathematical Software, 11(1), pp. 37-57, 1985.
- Wikipedia, « Reservoir Sampling ».
- Hoeksma S., « Machine Learning and Data: Exploring the Potential of Windowing ».