Projet EDTS - BRITS

Article presenté : BRITS: Bidirectional Recurrent Imputation for Time Series.

Par : Wei Cao, Dong Wang, Jian Li, Hao Zhou, Yitan Li et Lei Li

Presenté par : Hugo Lerogeron et Salim Talout Zitan

Code source du projet : salimtalout/BRITS (Fork de ce projet)

Introduction

Le but de ce papier est d'apporter une méthode de prédiction de données sur des séries de données temporelles (Multivariate time series, ou MTS). Les MTS sont largement utilisées dans de nombreux secteurs tels que la médecine, la finance, ou encore la météorologie. Il est commun que des données soient manquantes dans ces MTS, et cela peut être dû à plusieurs facteurs, comme un disfonctionnement des capteurs, ou encore une erreur de communication. Cela peut être problématique et il est nécessaire de pouvoir compléter ces données manquantes. Pour ce faire, plusieurs méthodes sont proposées.

Partie théorique

Multivariate Time Series

Avant d'aborder les méthodes utilisées pour résoudre la problématique, définissons les MTS.

Une MTS est une séquence de T observations telle que $X = \{x_1, x_2, ..., x_T\}$, avec $x_t \in \mathbb{R}^D$ comportant D features $\{x_t^1, x_t^2, ..., x_t^D\}$.

Puisque x_t peut avoir des valeurs manquantes, on introduit un vecteur m_t tel que :

$$m_t^d = \begin{cases} 0 \\ 1 \end{cases}$$

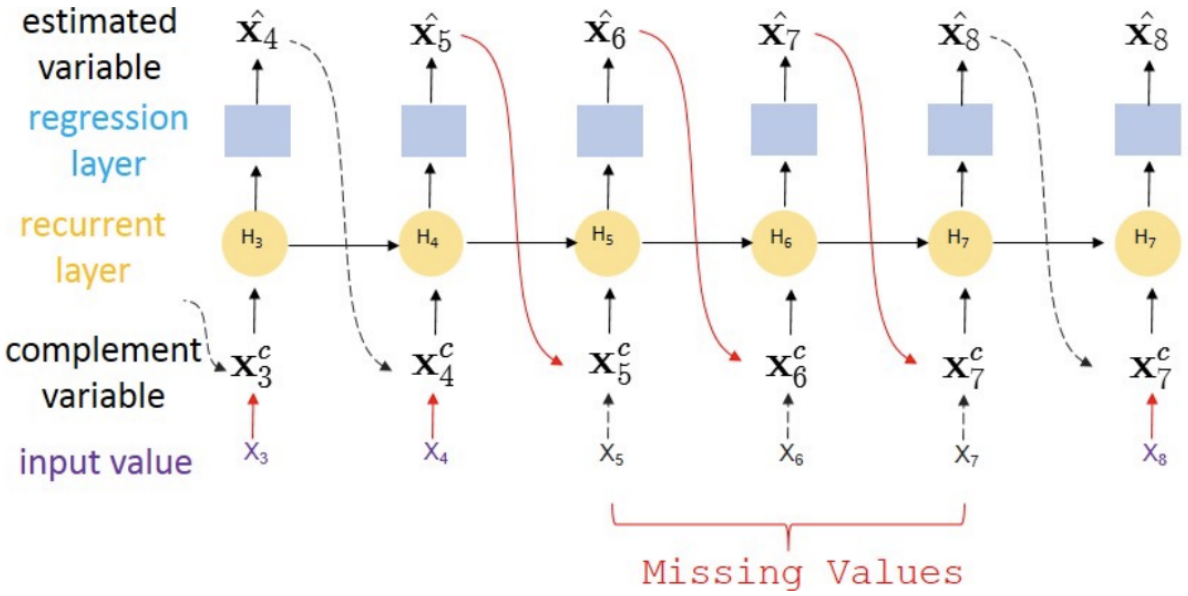
1 si on a une observation, 0 sinon.

RITS-I

Afin de surpasser les modèles RNN déjà existants, on utilise une méthode qui s'appelle RITS-I, qui se base sur des réseaux de neurones récurrents.

Pour cette méthode, supposons qu'à l'étape t, x_t^i et x_t^j sont pas corrélés pour $i \neq j$.

À l'étape t, si x_t est observé, on l'utilise pour valider l'imputation, et donc mettre à jour les poids du réseau. Dans le cas contraire, étant donné que les futures observations sont corrélées avec la valeur actuelle, on remplace x_t par l'imputation obtenue, et on la valide grâce aux observations suivantes.



Algorithme

$$
\begin{aligned}
\hat{\mathbf{x}}_t &= \mathbf{W}_x \mathbf{h}_{t-1} + \mathbf{b}_x, \\
\mathbf{x}_t^c &= \mathbf{m}_t \odot \mathbf{x}_t + (1 - \mathbf{m}_t) \odot \hat{\mathbf{x}}_t, \\
\gamma_t &= \exp\{-\max(0, \mathbf{W}_\gamma \delta_t + \mathbf{b}_\gamma)\}, \\
\mathbf{h}_t &= \sigma(\mathbf{W}_h [\mathbf{h}_{t-1} \odot \gamma_t] + \mathbf{U}_h [\mathbf{x}_t^c \circ \mathbf{m}_t] + \mathbf{b}_h), \\
\ell_t &= \langle \mathbf{m}_t, \mathcal{L}_e(\mathbf{x}_t, \hat{\mathbf{x}}_t) \rangle \, .
\end{aligned}
$$

Tout d'abord, on calcule l'estimation x_t. Ensuite, on calcule x_t^c, qui prend x_t dans le cas où l'on a la mesure, ou $\hat{x}_t$ sinon. Étant donné que les valeurs sont manquantes de manière irrégulière, on introduit γ_t, qui indique les valeurs manquantes. On prédit l'état t en fonction des étapes précédentes. Ensuite, on calcule l'erreur estimée à partir de l'erreur absolue moyenne.

$$\hat{\mathbf{y}} = f_{out}(\sum_{i=1}^{T} \alpha_i \mathbf{h}_i)$$

Enfin, on obtient la prédiction $\hat{y}$.

$$\frac{1}{T} \sum_{t=1}^{T} \ell_t + \mathcal{L}_{out}(\mathbf{y}, \hat{\mathbf{y}})$$

On fait ensuite la mise à jour en minimisant la loss accumulée.

Problèmes engendrés

Les principaux problèmes de cette méthode sont que les erreurs sur les valeurs manquantes ne sont obtenues jusqu'à l'observation suivante. Donc cela induit une convergence moins rapide, mais aussi des performances diminuées pendant l'entraînement.

La solution serait donc de ne pas seulement passer par l'observation suivante, mais aussi par l'observation précédente. C'est ce qu'ils ont appelé BRITS-I

BRITS-I

Donc le but de cette méthode est simplement de faire un RITS-I, mais dans les deux sens.

En forward, on estime une séquence et on a la loss pour cette séquence là (pareillement en backward).

- En forward :
  - Estimation : $\{\hat{x}_1, \hat{x}_2, ..., \hat{x}_T\}$
  - Loss : $\{l_1, l_2, ..., l_T\}$
- En backward :
  - Estimation : $\{\hat{x'}_1, \hat{x'}_2, ..., \hat{x'}_T\}$
  - Loss : $\{l'_1, l'_2, ..., l'_T\}$

On introduit un *consistency loss*, qui est en fait la différence entre l'estimation $x_t \in \mathbb{R}^D$ et l'estimation x' à une étape t : $l_t^{const} = Discrepancy(\hat{x}_t, \hat{x'}_t)$.

Ensuite, le loss final sera l'accumulation et loss en backward, forward et aussi la consistency loss.

Et enfin, l'estimation finale sera la moyenne entre l'estimation backward et forward.

RITS et BRITS

On a donc pû voir que l'on peut estimer les valeurs manquantes en regardant l'historique, non seulement dans un sens, mais aussi dans l'autre. Mais dans ces deux cas, on a considéré le fait que deux mesures à un temps t ne sont pas corrélées.

Or, nombreux exemples nous montrent le contraire. La météo entre Rouen est Paris est assez proche à un instant t, et cela est dû à la proximité spatiale. Sauf que les méthodes RITS-I et BRITS-I ne prennent pas cela en compte. L'idée, pour les prochaines méthodes, RITS et BRITS, est d'estimer grâce à l'historique des mesures, mais aussi grâce aux mesures voisines.

Répercutions sur l'algorithme

Donc dans l'algorithme, cela se traduit par l'ajout d'une estimation qui dépend des autres features (mesures voisines).

$$\hat{\mathbf{z}}_t = \mathbf{W}_z \mathbf{x}_t^c + \mathbf{b}_z$$

Ainsi, il faut ajouter une estimation qui combine les deux autres : l'estimation qui dépend de l'historique, mais aussi celle qui dépend des mesures voisines.

$$\hat{\mathbf{c}}_t \quad = \quad \beta_{\mathbf{t}} \odot \hat{\mathbf{z}}_t + (1 - \beta_{\mathbf{t}}) \odot \hat{\mathbf{x}}_t$$

Enfin, il faut modifier l'erreur estimée, pour qu'elle prenne en compte ces deux nouvelles estimations.

$$\ell_t = \mathcal{L}_e(\mathbf{x}_t, \hat{\mathbf{x}}_t) + \mathcal{L}_e(\mathbf{x}_t, \hat{\mathbf{z}}_t) + \mathcal{L}_e(\mathbf{x}_t, \hat{\mathbf{c}}_t)$$

Partie expérimentale

Le papier étant assez techniquement avancé, et puisque l'équipe de chercheurs a lié son papier au github implémentant les techniques développées dans l'algorithme, nous avons utilisé leur git pour nos tests.

L'utilisation du git est assez simple. Pour entraîner un des modèles type RNN comme le RITS-I, on entre simplement la commande suivante:

```
python main.py --model rits_i --epochs 1000 --batch_size 64 --impute_weight 0.3
--label_weight 1.0 --hid_size 108
```

Le modèle s'entraîne ensuite sur les différents jeux de données énoncés dans le papier avant de donner les résultats en test selon les tâches choisies. Selon les modèles, ces tâches sont de remplir les données manquantes et de faire de la classification ou non.

Le code

L'import des données, la définition de l'architecture et l'entraînement des réseaux de neurones et le calcul des métriques de résultat sont faites grâce à l'excellente bibilothèque *PyTorch*.

L'import des données

Cette partie est effectuée dans le module *data_loader* . Via la module *DataLoader* de *PyTorch*, on transforme les données stockées sous format *json* vers un *Loader*. Celui ci permet de ne charger en mémoire que les données que l'on va utiliser par la suite, et fonctionne donc très bien avec des optimiseurs utilisant le principe de la *Stochastic Graident Descent*. Les *DataLoader* sont des objets contenant les *Tensors*, type de *PyTorch* semblables au *nparrays* de *Numpy*, mais permettant d'être mis à jour via descente de gradient.

Les classificateurs

On compare d'abord les résultats obtenus via BRITS à ceux obtenus via techniques de *Machine Learning* plus classiques. Celles ci sont implémentées dans le module *baseline* via la librairie *fancyimpute* et contiennent une simple moyenne, moyenne via *KNN*, *MICE*, *ImputTS* et *STMVL*.

Les modèles basés sur des LSTMs, c'est à dire m_rnn, gru_d, rits_i, rits, brits_i et brits sont définis dans /models. Via *PyTorch* on définit leur architecture en définissant une par une les couches du modèle, avant de définir comment les poids sont mis à jour dans les méthodes forward et backward, mais aussi comment les estimations sont mises à jour dans la méthode reverse.

L'entraînement

Les modèles de *Machine_Learning* classique sont entraînés via la librairie *fancyimpute* directement. Pour les modèles type *rnn*, on définit dans le modèle la manière dont il s'entraîne. Le reste est définit dans le main: l'optimiseur choisi est *Adam*, ainsi que le *batch_size* et les fonctions affichant le déroulement de l'entraînement.

La mesure de performance

La mesure de performance est effectuée dans le *main*. Elle se base principalement sur deux métriques: la *Mean Average Error MAE* $MAE = \frac{\sum_i |pred_i - label_i|}{N}$ et la *Mean Relative Error MRE*

$$MRE = \frac{\sum_i |pred_i - label_i|}{\sum_i |label_i|}$$.

Jeu de données

Les modèles sont évalués sur 3 datasets.

- Air Quality: Mesures de la qualité de l'air faite à Pékin. 13.3% des données sont manquantes.
- Healthcare data: Données à 4000 variables contenant les informations biologiques de patient en salle d'opérations. 78% des valeurs sont manquantes, les données sont donc extrêmement creuses. La tache de prédiction consiste à déterminer si le patient survit ou non à l'opération.
- Human Activity: Mesures de capteurs sur des humains effectuant des tâches quotidiennes. La tâche de prédiction consiste à déterminer l'action en cours.

Résultats

Voici les résultats obtenus par les différents algorithmes sur les jeux de données décrits au dessus:

Method		Air Quality	Health-care	Human Activity
Non-RNN	Mean	55.51 (77.97%)	0.461 (65.61%)	0.767 (96.43%)
	KNN	29.79 (41.85%)	0.367 (52.15%)	0.479 (58.54%)
	MF	27.94 (39.25%)	0.468 (67.97%)	0.879 (110.44%)
	MICE	27.42 (38.52%)	0.510 (72.5%)	0.477 (57.94%)
	ImputeTS	19.58 (27.51%)	0.390 (54.2%)	0.363 (45.65%)
	STMVL	12.12 (17.40%)	/	/
RNN	GRU-D	/	0.559 (77.58%)	0.558 (70.05%)
	M-RNN	14.05 (20.16%)	0.445 (61.87%)	0.248 (31.19%)
Ours	RITS-I	12.45 (17.93%)	0.385 (53.41%)	0.240 (30.10%)
	BRITS-I	11.58 (16.66%)	0.361 (50.01%)	0.220 (27.61%)
	RITS	12.19 (17.54%)	0.292 (40.82%)	0.248 (31.21%)
	BRITS	**11.56 (16.65%)**	**0.278 (38.72%)**	**0.219 (27.59%)**

Ici, la tâche est simplement la recréation des jeux de données en remplissant les trous dans les jeux de données. Les métriques sont *MAE* et *MRE* en pourcentage.

On remarque que, sans surprises, remplacer en faisant simplement la moyenne est très mauvais. C'est un peu mieux si on choisit les K plus proches voisins et qu'on fait la moyenne avec. De plus, BRITS est le meilleur modèle sur tous les jeux de données, battant l'ancien état de l'art *STMVL* sur le jeu de données "Air Quality" et écrasant les solutions *baseline* sur les autres jeu de données. Enfin BRITS dépasse BRITS-I sur tous les jeux, mais est parfois très proche comme sur le jeu "Air Quality" pouvant illustrer que les données ne sont pas spatialement corréllées pour ce jeu de données.

Enfin, même pour les tâches de classification, BRITS est le meilleur algortihme, validant la méthode.

Method	Health-care (AUC)	Human Activity (Accuracy)
GRU-D	0.834 ± 0.002	0.940 ± 0.010
M-RNN	0.817 ± 0.003	0.938 ± 0.010
RITS-I	0.821 ± 0.007	0.934 ± 0.008
BRITS-I	0.831 ± 0.003	0.940 ± 0.012
RITS	0.840 ± 0.004	0.968 ± 0.010
BRITS	$\mathbf{0.850 \pm 0.002}$	$\mathbf{0.969 \pm 0.008}$

Conclusion

Les tests expérimentaux valident l'intêret de la méthode BRITS: celle ci obtient en effet de meilleures performances que toutes les autres méthodes testées sur les 3 jeux de données. Il conviendra cependant de faire valider cette méthode sur d'autres jeux de données.