Professeur : Matthieu Durut

Projet de programmation en C++

Prévisions de séries temporelles

Ali Mechawrab – David Billod de Filiquier ENSAE 2A 02/2015

Table des matières

Introduction

Comment prévoir le PIB français en 2015? Comment prévoir la température qu'il fera demain? Ou encore comment savoir combien coûtera ma baguette de pain dans 1 mois? Ce sont là autant de question auxquelles nous allons essayer de répondre.

En effet, nous nous proposons de "prédire" les valeurs futures d'une série temporelle à partir de ses valeurs passées et de celles d'autres séries que nous choisirons. On comprend donc qu'il faut déjà avoir une idée en tête sur les séries temporelles susceptible de rendre compte de l'évolution de la série à expliquer : PIB/Consommation ou Prix baguette/ Prix farine sont par exemples susceptibles de former de bons couples variable à expliquer/variable explicative. On peut aussi considérer plusieurs séries : PIB/Consommation-Investissement par exemple. A l'inverse, prendre comme variable à expliquer le PIB français et prendre comme variable explicative le nombre de communes de l'Azerbaïdjan a relativement peu de chances de fournir une bonne estimation. On préfère donc prendre des variables explicatives que l'on suppose corrélées avec la variable à expliquer car ainsi elles sont plus susceptibles de présenter les mêmes variations.

Toutefois, nous parlons bien ici de tendances ressemblantes et pas seulement de corrélation: ainsi si une variable explicative est une transformation linéaire décroissante de la variable à expliquer, le pouvoir de prédiction de cette variable explicative sera nul, quand bien même elle n'est pas indépendante de la variable à expliquer.

I. Présentation du principe du projet

Ce projet va donc prendre un fichier contenant des séries temporelles et afficher dans la console la ou les valeurs prédites. A cet égard, le fichier contenant les séries doit avoir un format particulier.

En effet ce fichier initial doit tout d'abord être un .txt. Ensuite il doit se présenter sous la forme d'une suite de nombres, dans l'ordre chronologique entrecoupés par le nom des séries correspondantes sous la forme x1,x2… pour les séries explicatives et y pour la série à expliquer. Ainsi on pourrait avoir :

y
1
2
3
X1
4
5
6
X2
6
52.5
0.1
N.B : ici 1 serait la première valeur de y, 2 la seconde,…

Il est important de noter que les nombres à virgule devront ici être présentés avec un point à la place, selon la notation anglo-saxonne traditionnelle. Enfin, et c'est très important les séries doivent être de même taille, c'est-à-dire qu'elles doivent avoir le même nombre de valeurs.

Maintenant que l'on a structuré la façon de fournir des données au programme, expliquons le principe général du processus.

Premièrement, le programme va normaliser les séries en les divisant par leur écart-type, ceci afin de les rendre numériquement comparables.

Ensuite le programme doit trouver k « séquences » de longueur h, c'est-à-dire k suites de h termes consécutifs parmi les séries explicatives, telles que ces séries soient les plus proches, au sens d'un distance entre vecteurs, de la séquence correspondante aux h derniers termes de y. Pour ce projet, nous nous limiterons à regarder 3 types de distances, issues des normes vectorielles suivantes : norme 1, norme 2 et norme infinie. Une fois ces séquences trouvées, on va moyenner sur les valeurs suivantes pour avoir notre prédiction. Si on ne connaît pas la valeur suivante de l'une des séquences les plus proches, on va alors considérer la dernière valeur de la séquence. En remultipliant ensuite par l'écart-type de la série à expliquer, y, on obtient alors la prévision souhaitée.

Toutefois, il vient alors la question de choisir comment trouver de « bons » k et h. Pour cela nous allons faire de la cross-validation. Supposons que nous ayons des échantillons de T dates. Alors pour chaque couple (k,h), variant dans un intervalle choisi arbitrairement, on va prévoir chaque valeur de y en t, avant T. Nous faisons donc alors comme si nous n'avions pas d'information après la date t. Cela nous permet de trouver une erreur d'estimation, pour le couple (k,h) en ce t. En moyennant

ensuite sur chaque t, on obtient une erreur de cross-validation pour un couple (k,h).Ensuite, le programme va choisir les k et h tels qu'ils minimisent cette erreur de cross-validation.

Après avoir trouvé ces paramètres optimaux, on va cette fois prévoir la valeur de y en T+1, qui est inconnue, contrairement aux valeurs que l'on prévoyait dans la cross-validation. On procède selon la même façon que pour obtenir les prévisions de cross-validation.

Ensuite le programme va donc afficher les prévisions issues de chaque distance, et montrent également ce que l'on va appeler le degré d'utilité des séries explicatives. Ce degré d'utilité est ici défini comme la part de plus proches voisins issues de la série en question sur le nombre total de plus proches voisins, soit le k issu de la cross-validation.

On notera que le programme demande à l'utilisateur combien de valeurs le dit utilisateur veut prédire à l'avance. Dans ce cas, si l'utilisateur répond plus que 1, alors le programme va faire ce qui est décrit au-dessus. Mais quand il aura fini de prédire le prochain terme de y, il updatera les données. C'est-à-dire qu'il modifiera y en supprimant la première valeur et en rajoutant la valeur prédite, et ce pour garder toujours la même taille entre tous les tableaux. Toutefois, le passé des séries explicatives restant inchangé, il est intuitif qu'il faut rester prudent vis-à-vis des résultats donnés pour une prédiction lointaine dans l'avenir.

II. Explication des fonctions et des fichiers utilisés

Nous allons désormais nous pencher plus en détail sur la structure du code et son fonctionnement. Nous allons donc commencer par les 2 fichiers Fonc.cpp et Fonc.h son header.

A. Fonc.cpp et Fonc.cpp

Ces deux fichiers vont être constitués des fonctions qui vont nous aider à réaliser notre prévision. En premier lieu, il nous a fallu coder quelques fonctions simples, parmi lesquelles on trouve les fonctions suivantes:

- min_tab : elle prend en argument un tableau dynamique et retourne son minimum
- max_tab : elle prend en argument un tableau dynamique et retourne son maximum
- distance2 : prend en argument deux tableaux dynamiques et retourne la racine carrée de la somme des carrés des différences des termes des tableaux, soit en d'autres termes, la norme 2, dite euclidienne, de la différence entre les deux tableaux
- distance1 : prend en argument deux tableaux dynamiques et renvoie la somme des valeurs absolues de la différence entre les tableaux pris en arguments. C'est donc la norme 1 de la différence entre les deux tableaux

- distanceinf : prend à nouveau en argument deux tableaux dynamiques et renvoie le maximum de la valeur absolue de la différence entre les termes des tableaux pris en argument. C'est donc la norme infinie de la différence entre les deux tableaux
- choix_dist : prend deux tableaux dynamiques en arguments et un entier. l'entier, qui doit être compris entre 0 et 2 détermine si la fonction va retourner distance1, distance2 ou distanceinf. Ici l'entier 0 conduit au choix de la distanceinf et l'entier 1 à celui de la distance1.
- argmin : prend en argument un tableau dynamique et retourne l'indice où se trouve le minimum du tableau
- moyenne : prend un tableau dynamique en argument et retourne la moyenne de ce tableau
- ecart-type : prend un tableau dynamique en argument et renvoie l'écart-type du tableau

Ces fonctions vont constituer le socle mathématique sur lequel repose notre code.

Afin ensuite de pouvoir alléger l'écriture du fichier main.cpp, qui "fait" le travail, nous avons voulu coder plusieurs fonctions qui y intervenaient.

- extract : prend en argument un tableau dynamique, et deux entiers. Dans notre projet il faut pouvoir avoir l'ensemble des séquences évoquées plus haut. Donc en prenant une série, ici un tableau dynamique, cette fonction va renvoyer la séquence issue du tableau en argument, entre l'entier 1 et l'entier 2.
- matrice_sous_seq : prend en argument un tableau dynamique et un entier. Cette fonction s'appuie sur extract pour retourner un tableau de tableau contenant l'ensemble des sous-séquences de taille h, h étant l'entier pris en argument.
- argminmultiple : prend en argument un tableau dynamique et un entier. La fonction va retourner un tableau dynamique qui va contenir les indices des k plus petites valeurs du tableau pris en argument, k étant l'entier pris en argument. Cela nous permettra facilement de trouver les k plus proches voisins, i.e les k séquences qui présentent une distance minimale par rapport à la séquence à prédire.
- renorm : prend en argument un tableau dynamique. Cette fonction renvoie ce même tableau, en ayant divisé tous les éléments par l'écart-type du tableau.
- Construire_matrice_data : prend en argument un tableau dynamique et deux entiers. Dans notre code, le tableau passé en argument est en fait le tableau qui contient toutes les valeurs des séries. tabdata2 va réorganiser cela en renvoyant un tableau de tableaux où le premier élément est le tableau de la première série, etc. Les deux entiers vont donner les dimensions du tableau de sortie
- coupertab : prend un tableau de tableau en argument et deux entiers. Afin de faire notre cross-validation, nous avons besoin de considérer les données s'arrêtant à un certain t, un des entiers pris en argument. La fonction va alors retourner le tableau de tableau des données comme si on ne connaissait les données que jusqu'à t.

- construire_matrice_distance : prend en argument quatre entiers, un tableau de matrices et un tableau. Cette fonction va retourner un tableau de tableau ou une matrice où le i-ème élément est le tableau des distances de chaque sous-séquence de la série xi à la séquence à prédire, passée en argument. Les entiers en arguments servent au choix de la distance et à faire attention aux problèmes de dimension des tableaux manipulés.
- Simplifietab : prend en argument 3 entiers, et un tableau de tableaux ou matrice. Cette fonction va juste transformer notre tableau de tableaux en tableau simple. Les entiers pris en arguments servent simplement à contrôler la taille des objets manipulés.

B. main.cpp

C'est le coeur du programme. C'est lui qui va réaliser les prévisions demandées à l'aide des fonctions codées dans les fichiers évoqués au-dessus.

En premier lieu, l'utilisateur doit indiquer le chemin où se trouve le fichier .txt qui comporte les donées selon le format explicité plus haut. On rappelle que ce chemin ne doit pas comporter d'accents.

Ensuite il doit choisir le nombre de valeurs qu'il souhaite prévoir, i.e le nombre de coup, nbcoup dans le main.

On entre dans une première boucle si l'ouverture du fichier .txt a réussi. Ensuite, nous comptons le nombre de séries, le nombre de valeurs par séries (commun à toutes les séries) et nous stockons ces informations dans un tableau simple, TAB.

On constitue ensuite la matrice des données (tableau de tableaux) où le i-ème élément est un tableau avec les valeurs de la i-ème série.

On passe ensuite à la cross-validation pour trouver les meilleurs k et h, comme expliqué plus haut.

Pour chaque distance, on utilise ensuite un tableau de tableaux ou matrice contenant les données de la série explicative updatées pour la valeur prédite issue de cette distance.

Comme expliqué plus haut, nous créons alors une copie tronquée de nos données pour faire comme si nous n'avions pas toute l'information jusqu'à une date t

Nous normalisons ensuite ces données en divisant chaque série tronquée par son écart-type. Afin de centraliser l'information, nous allons stocker dans un tableau de matrice, Mat_totale, l'ensemble des sous-séquences de taille h de chaque série.

Ensuite nous construisons le tableau des distances de chaque sous-séquence d'une série explicative à la séquence à prédire, les h derniers termes de y.

On va ensuite regarder quelles sont les k séquences les plus proches, puis on stocke dans Val_suiv les valeurs suivantes de ces segments. Dans le cas où une des k plus proches séquences serait les h derniers termes d'une série, nous retiendrons la dernière valeur de cette valeur, faute de pouvoir

considérer la prochaine valeur. On prend ensuite la moyenne de ces valeurs suivantes pour obtenir notre prédiction.

En comparant cette prévision avec la vraie valeur, on obtient une erreur de prédiction. On stocke, à k et h fixés, ces erreurs pour tous les t considérés et la moyenne de ce tableau nous donne l'erreur moyenne de cross-validation pour les k et h considérés. En minimisant cette erreur, on trouve donc les k et h optimaux.

Maintenant que nous savons quels sont les meilleurs paramètres k et h, nous allons pouvoir prédire la prochaine valeur de y, inconnue.

En procédant de manière similaire qu'avec les données tronquées de cross-validation, nous allons ainsi aboutir à une prédiction pour y, pour chaque distance utilisée.

Le programme affiche dans la console un résumé des résultats obtenus en fonction de la distance. On y trouve l'utilité de chaque série ainsi que la ou les prochaines valeurs prédites

Conclusion

Ce programme permet de prédire la ou les valeurs suivantes d'une série temporelle à l'aide d'autres séries temporelles, dites explicatives. Néanmoins, nous sommes conscients des limites des résultats car la qualité de la prévision s'estompe bien entendu avec le nombre de valeurs à prédire et reste très dépendante de la qualité prédictive des séries explicatives.

Nous notons, à la vue de plusieurs résultats obtenus avec diverses séries que, le choix de la distance n'a pas beaucoup d'influence quant aux prévisions, bien qu'elle en ait parfois pour les paramètres optimaux issus de la cross-validation.

On peut aussi s'interroger sur la pertinence des distances prises dans le cadre du projet. Celles choisies nous viennent directement de l'algèbre. Mais on pourrait dire que ce ne sont pas forcément les meilleures distances car elles ne prennent pas suffisamment en compte la tendance des séries, notamment la forme graphique de la série.