



Mémoire de certification

Conduire un projet de sciences de données

CNCP 1527

Modèles de prédiction pour des séries de données temporelles

JetPack Data

ALAOUI MOHAMED

Data Science Starter Program
DSSP 6 – 2017

Remerciements

Je tiens tout d'abords à remercier toutes les personnes qui m'ont accompagné durant ces derniers mois afin de réussir cette entreprise de certification à l'Ecole Polytechniques.

Je voudrais particulièrement rendre hommage à mes proches qui m'ont encouragé tout au long de ce parcours.

Je tiens à remercier Mme Alexandra DUFAU du cabinet ALTEDIA qui m'a permis de rencontrer Mr Shankar Arul président de JetPack Data avec qui l'échange a été de grande qualité et qui m'a permis d'appliquer concrètement mes travaux au sein de sa startup, merci Shankar.

Par ailleurs, je souhaite à mes collègues de la promotion DSSP 6 ainsi qu'à l'équipe pédagogique de l'Ecole Polytechniques Executive Education tout le succès que mérite leurs projets et initiatives.

Enfin, je souhaite manifester ma reconnaissance à mon ancien employeur, BNP PARIBAS qui a permis la réalisation de ce projet, à mon ancien manager, Mme Alice DESHAYS, dont l'aide a été déterminante.



Résumé

Ce rapport contient un compte rendu détaillé des travaux qui ont été effectués suite au volet présentiel du Data Science Starter Programme de l'école Polytechniques Executive Education.

Afin d'appliquer les concepts acquis durant cette première phase de certification, j'ai décidé de réaliser un projet en entreprise pour une mise en œuvre concrète de mon étude.

Ainsi, j'ai eu l'opportunité de collaborer avec la startup JetPack Data afin de définir et implémenter un module de prédiction de séries temporelles pour les utilisateurs de la plateforme.

Table des matières

1	Introduction	5
1.1	Enjeu	5
1.2	Contextualisation	6
1.3	Structure de ce mémoire	6
2	Vue d'ensemble de la plateforme JetPack Data	7
2.1	Présentation de la plateforme	7
2.2	Technologies utilisées et architecture simplifiée de la plateforme	9
2.3	Description des jeux de données utilisés	10
2.3.1	<i>Jeu de données correspondant à un revenu économique</i>	10
2.3.2	<i>Jeu de données correspondant à un volume de vente (lien)</i>	11
2.3.3	<i>Jeu de données correspondant à un stock financier</i>	11
2.3.4	<i>Commentaire sur les jeux de données utilisées dans ce mémoire</i>	11
2.4	Contour du « produit minimum viable » à livrer	12
3	Modèles statistiques pour la prédiction des time series	13
3.1	Introduction à la statistique des séries temporelles et exploration des séries temporelles	13
3.1.1	<i>Notion de stationnarité</i>	13
3.1.2	<i>Décomposition d'une série non-stationnaire</i>	15
3.2	Expérimentation et sélection de modèles de prédiction	16
3.2.1	<i>Expérimentation avec le modèle ARIMA [6] [7]</i>	17
3.2.2	<i>Expérimentation avec le module Prophet de Facebook</i>	20
3.2.3	<i>Description du produit livré</i>	21
4	Machine Learning, Deep Learning et time series.....	24
4.1	Fonctionnement des RNN LSTM	24
4.1.1	<i>Aperçu général d'un RNN</i>	24
4.1.2	<i>Détail d'un neurone LSTM</i>	25
4.2	Réseau de neurones dans un contexte de time series	25
4.3	Expérimentation autour des LSTM et des time series	26
4.3.1	<i>Préparation des données pour de l'apprentissage supervisé</i>	26
4.3.2	<i>Préparation des données pour de l'apprentissage d'un réseau LSTM</i>	27
4.4	Résultat des différents tests effectués	29
4.4.1	<i>Variation du volume d'entraînement pour les ventes de champagne</i>	29
4.4.2	<i>Réseau LSTM sur un stock financier</i>	31
4.4.3	<i>Récapitulatif sur les réseaux LSTM</i>	33
5	Critique des différents résultats obtenus.....	34
5.1	Modèles statistiques	34
5.1.1	<i>Synthèses des résultats</i>	34
5.2	Modèles de Deep Learning	35
5.2.1	<i>Synthèses des résultats</i>	35
5.3	Commentaires	36
6	Conclusion et élargissement.....	37
7	Annexes.....	38

1 Introduction

Dans ce rapport nous abuserons volontiers de certains anglicismes. Nous utiliserons le nom de séries temporelles ou de time series d'une manière interchangeable. Nous userons également du mot forecasting pour désigner une prévision. Aussi, nous évoquerons le mot prédiction pour dénommer la même problématique.

1.1 Enjeu

Les séries de données temporelles ou time series peuvent être considérées comme une séquence indexée par un support temporel ou tout simplement des données numériques qui évoluent dans le temps. Elles appartiennent à la famille de collection de données numériques ordonnées par un ensemble dénombrable. Ainsi les méthodes étudiées dans ce rapport peuvent être généralisées à cette famille de données.

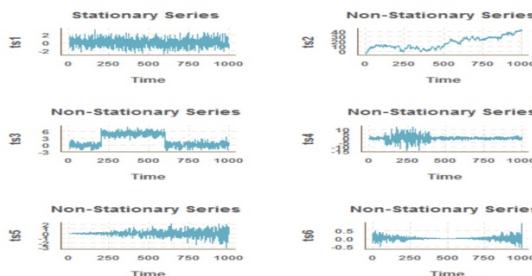


Figure 1.1 – Type de time series



Figure 1.2 – Time Series de type stochastique – Stock financier

L'étude des time series est une branche des sciences de données omniprésente. En effet, tous les systèmes dynamiques variant au cours du temps produisent des séries de données de ce type. L'analyse prédictive de séries temporelles concerne une multitude de domaines : l'étude des tendances économiques et financières, le comportement des utilisateurs d'internet, les

indicateurs issus de réseaux de capteurs, le traitement de différents signaux en biologie, médecine, physique et bien d'autres domaines.

Le forecasting est un exercice délicat car il nécessite une spécialisation poussée ainsi qu'une large expérience. Cependant, il existe des librairies qui rendent accessible cette tâche dans une certaine mesure pour les non experts.

Nous pouvons notamment citer Facebook et sa librairie Prophet écrite en R et Python. Un papier a été publié à ce propos [1]. Nous en aborderons certains aspects par la suite.

1.2 Contextualisation

Le maître mot de la plateforme JetPack Data est la simplicité et l'accessibilité aux non spécialistes de l'exploration des données.

Mon travail consiste à répondre à deux objectifs :

- Identifier, expérimenter puis livrer des modèles simples à utiliser répondant au besoin de la plateforme JetPack Data
- Appliquer les concepts et notions abordés lors du premier volet de la certification DSSP 6

1.3 Structure de ce mémoire

Dans un premier temps, nous présenterons la plateforme JetPack Data et les technologies utilisées. Nous verrons aussi les jeux de données utilisés pendant nos expérimentations puis nous définirons le contour du produit à livrer.

Par la suite, nous examinerons les modèles statistiques permettant d'appréhender les types de séries temporelles existantes au sein de la plateforme JetPack Data. Nous sélectionnerons ensuite ceux qui seront utilisés pour notre livrable.

Enfin, nous ouvrirons notre expérimentation sur des méthodes de Deep Learning pouvant intéresser la plateforme afin d'analyser une catégorie de jeu de données encore plus vastes.

2 Vue d'ensemble de la plateforme JetPack Data

JetPack Data est une startup franco-indienne employant trois salariés. Ses locaux se trouvent à Chennai, London et Paris.

Fondée par Shankar Arul, puis lauréate du French Tech Ticket 2017, JetPack Data a été incubée par NUMA NUMA. Ses locaux se trouvent à la StationF.

La plateforme JetPack Data ambitionne de rendre accessible l'exploration intelligente de la donnée aux non spécialistes. L'outil permet à ces utilisateurs d'explorer des données business avec simplicité en utilisant une interface homme machine performante et ergonomique utilisant des graphiques de transition réactifs [2].

2.1 Présentation de la plateforme

La plateforme est accessible en démonstration via le lien suivant : [JetPack Data](#)

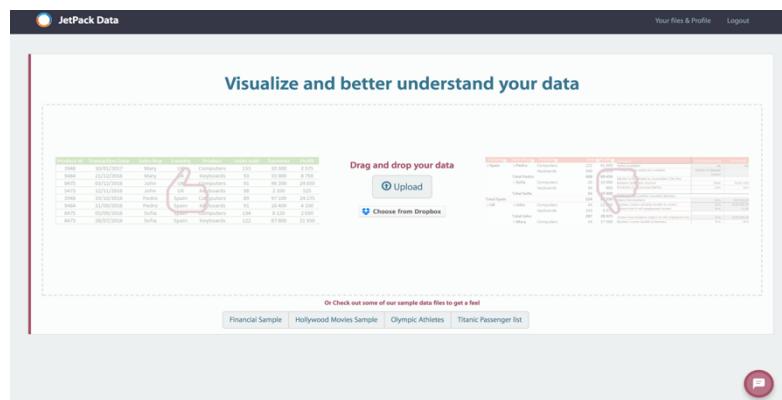


Figure 2.1 – Page de chargement d'un jeu de données

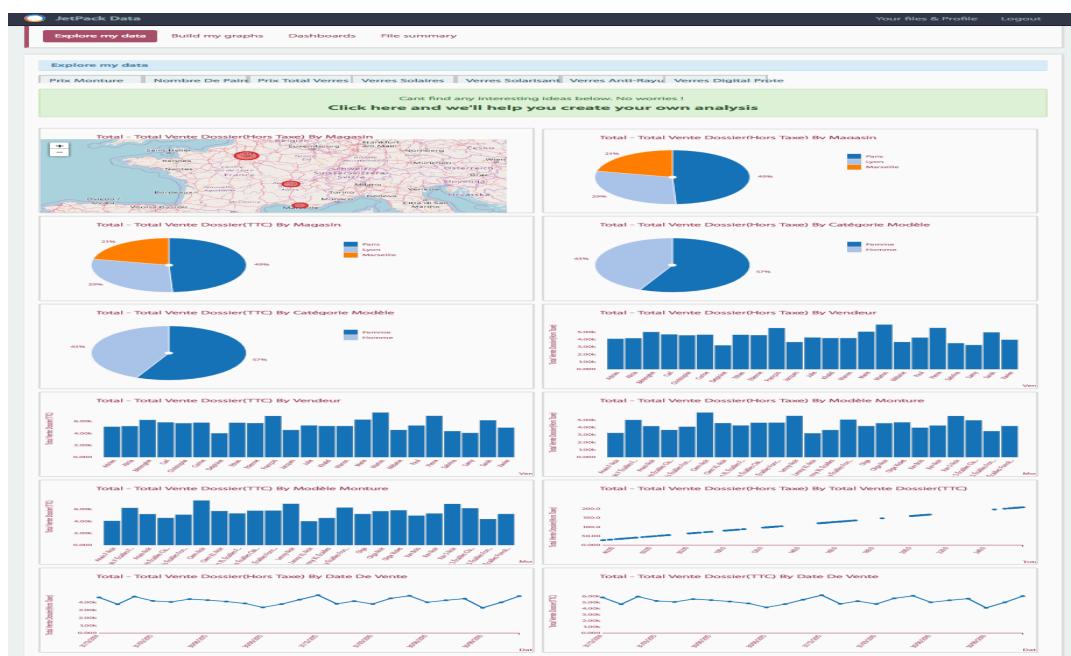


Figure 2.2 – Inférence de graph après chargement des données

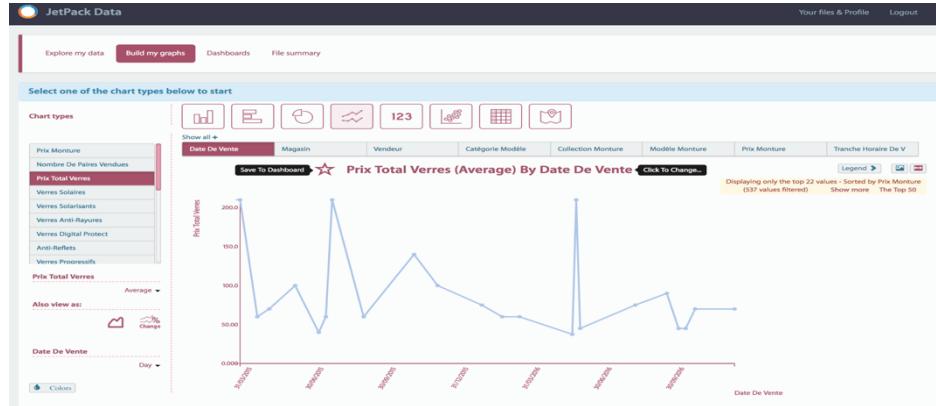


Figure 2.3 – Création de graph personnalisé en deux clics

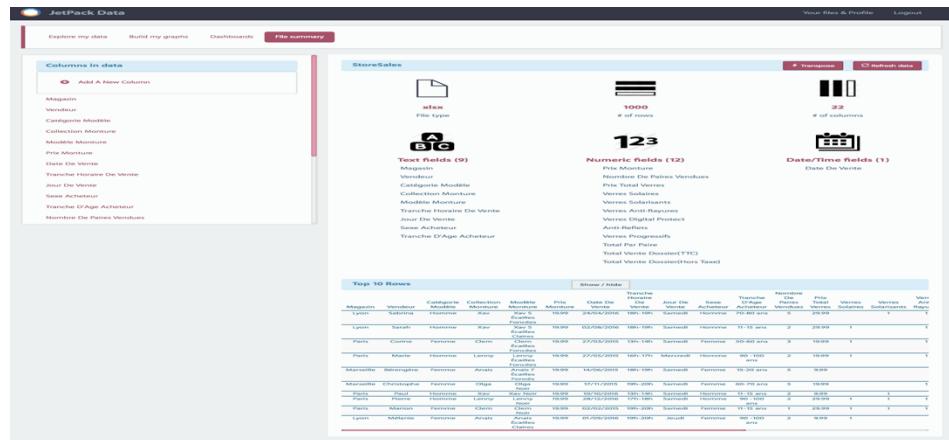


Figure 2.4 – Récapitulatifs des données

2.2 Technologies utilisées et architecture simplifiée de la plateforme

La plateforme est basé sur une architecture n-tiers avec une interface utilisateur graphique en « single page application ».

Coté front-end, les technologies AngularJs , D3 js, SocketIO ont été intégrées :

- AngularJs permet le « two ways » binding pour un affichage réactif au changement de la data
- D3 js permet un affichage ergonomique des graphiques basé sur des animations de transition.
- SocketIO permet une communication synchrone/asynchrone avec le back-end et donne la possibilité de faire du push du back-end vers l'interface graphique.

La partie service back-end est géré par un Framework python Flask permettant de faire du REST et d'orchestrer les services de la plateforme. Ces derniers s'appuient sur un cache memcached afin d'optimiser l'accès à la donnée et utilise un SGBD Mysql pour gérer les droits. Enfin un outil de messaging Celery permet une communication fluide avec la couche de persistance.

Le stockage est sous Amazon Web Services S3 (AWS S3) pour la partie données métiers et sous une base de données relationnelles Mysql pour l'authentification et la juridiction.

Le déploiement est effectué sous un cloud AWS avec une gestion dynamique des instances de type actif/passif.

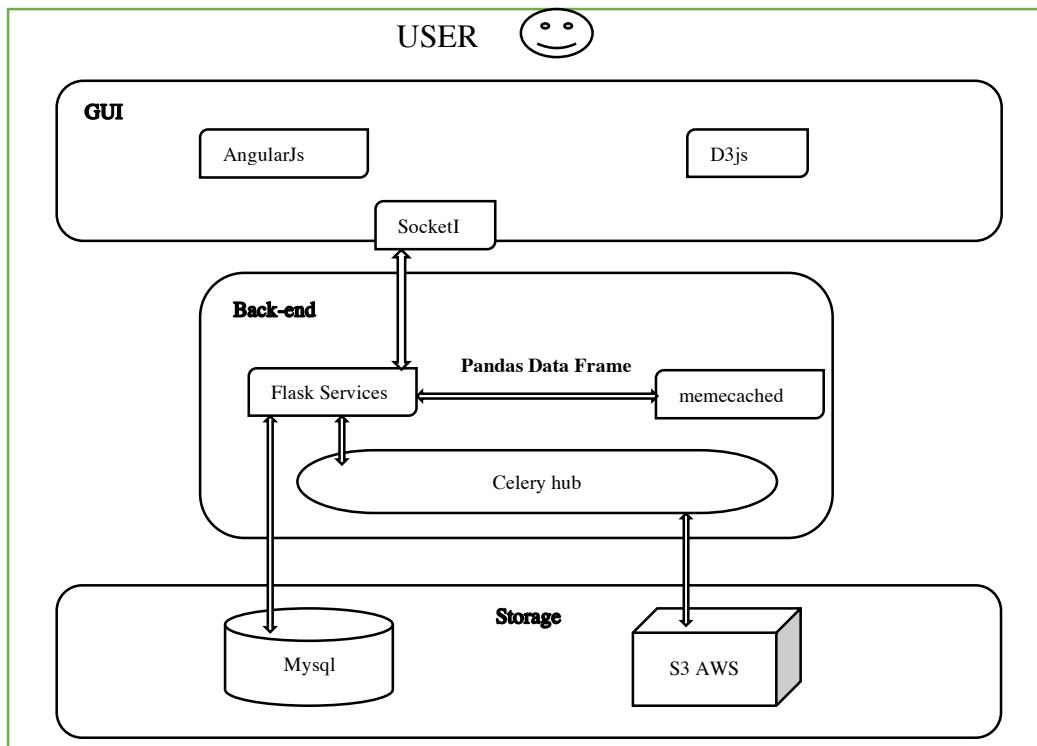


Figure 3 Architecture logicielle

2.3 Description des jeux de données utilisés

Les utilisateurs de la plateforme JetPack Data sont majoritairement des « business analyste » travaillant sur des problématiques économiques et financières. Ainsi, nous avons choisi dans nos expérimentations des jeux de données reflétant ce type de time series. Cependant, les méthodes abordées dans ce mémoire sont utilisables pour d'autres catégories de time series.

Les données temporelles étudiées ont une structure de data frame Pandas avec deux colonnes **ds et y** :

- ds : représentant le temps
- y : représentant la donnée numérique à étudier

2.3.1 Jeu de données correspondant à un revenu économique

Ces données représentent principalement des revenus quotidiens entre janvier 2014 et Janvier 2015 en différentes monnaies. Nous nommerons ce jeu de données DS-revenu, DSrev ou TSrev selon le contexte.

	rownum	country	currency_code	credit	transaction_date	cda_number	contract_number	product_name	vendor	contract_owner	...
0	1	BE	EUR	0	2013-04-10	a0YC000000TRMwLMAX	006C000000kuYhr	Superior room with breakfast for two people	Courtyard by Marriott Brussels	Pim De Schepper	... 4,000€
1	1	BE	EUR	0	2013-10-15	a0YC000000TVK7nMAH	006C000000ININp	Standard Double Room (weekdays)	NH Musica(cbds)	Nils Oldenburg	... 3,000€
2	1	BE	EUR	0	2013-10-15	a0YC000000TVK7oMAH	006C000000ININp	Standard Double Room (weekdays and weekend)	NH Musica(cbds)	Nils Oldenburg	... 3,000€
3	1	BE	EUR	0	2013-10-15	a0YC000000TVK7oMAH	006C000000ININp	Standard Double Room (weekdays and weekend)	NH Musica(cbds)	Nils Oldenburg	... 3,000€
4	1	BE	EUR	0	2013-10-15	a0YC000000TVK7oMAH	006C000000ININp	Standard Double Room (weekdays and weekend)	NH Musica(cbds)	Nils Oldenburg	... 3,000€

Figure 4.1 – Data Set complet

	rownum	credit	external_option_id	booking_nbr	stay_day	number_of_adults	number_of_children	GB_usd_curr	GR_usd_curr	GR_loc_curr
count	86563.0	86563.0	8.656300e+04	8.656300e+04	86563.000000	86563.000000	86563.0	8.656300e+04	8.656300e+04	8.656300e+04
mean	1.0	0.0	3.832531e+07	1.252555e+06	2.191895	4.387256	0.0	2.704284e+09	3.190199e+09	2.517450e+03
std	0.0	0.0	8.109641e+06	5.616675e+05	1.583258	3.197926	0.0	2.542705e+09	2.233206e+09	1.526857e+04
min	1.0	0.0	1.972670e+07	0.000000e+00	0.000000	0.000000	0.0	3.293900e+04	0.000000e+00	0.000000e+00
25%	1.0	0.0	3.162541e+07	7.218560e+05	1.000000	2.000000	0.0	1.110251e+09	1.510366e+09	2.530000e+02
50%	1.0	0.0	3.849803e+07	1.230939e+06	2.000000	4.000000	0.0	1.845881e+09	2.896307e+09	1.175000e+03
75%	1.0	0.0	4.516658e+07	1.773456e+06	2.000000	4.000000	0.0	3.396709e+09	4.420805e+09	3.293000e+03
max	1.0	0.0	5.434296e+07	2.278815e+06	21.000000	56.000000	0.0	9.996275e+09	9.999398e+09	4.358224e+06

Figure 4.2 – Description des données numériques

Les données temporelles qui vont nous intéresser sont dans la colonne « GR_loc_curr » qui correspond au revenu quotidien en monnaie locale.

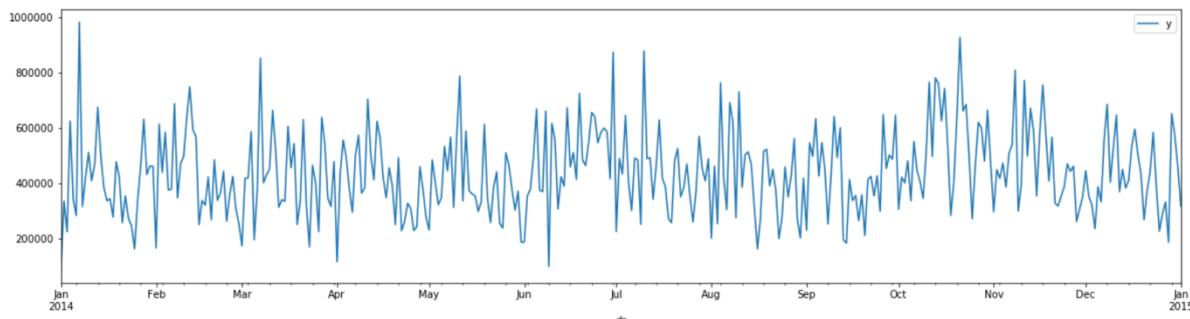


Figure 4.3 – TS-revenu

Les caractéristiques de cette time série seront abordées dans la partie exploration données et statistique.

2.3.2 Jeu de données correspondant à un volume de vente ([lien](#))

Ce jeu représente le volume mensuel de vente de champagne entre 1964 et 1974. Nous le nommerons DS-volume (TS-volume, TSvol) selon le contexte.

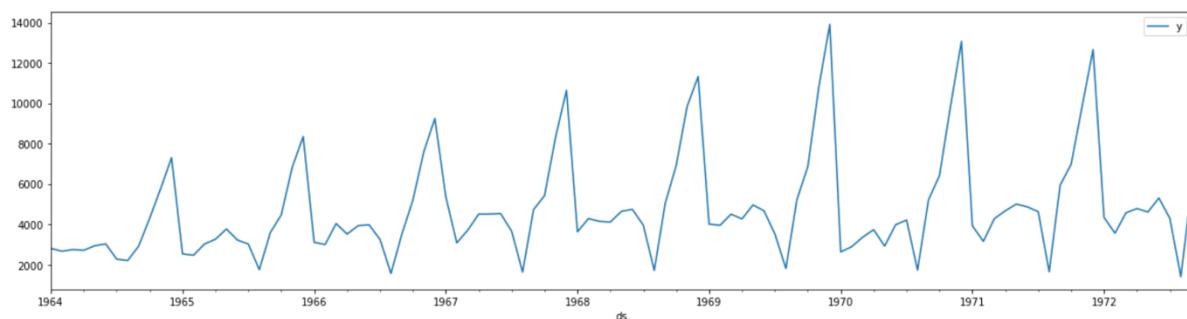


Figure 4.4 – TS-volume

2.3.3 Jeu de données correspondant à un stock financier

Ce jeu de données représente le NASDACQ de fin 2012 jusqu'à nos jours. Nous le nommerons DS-stock, DSstock ou TSstock selon le contexte.

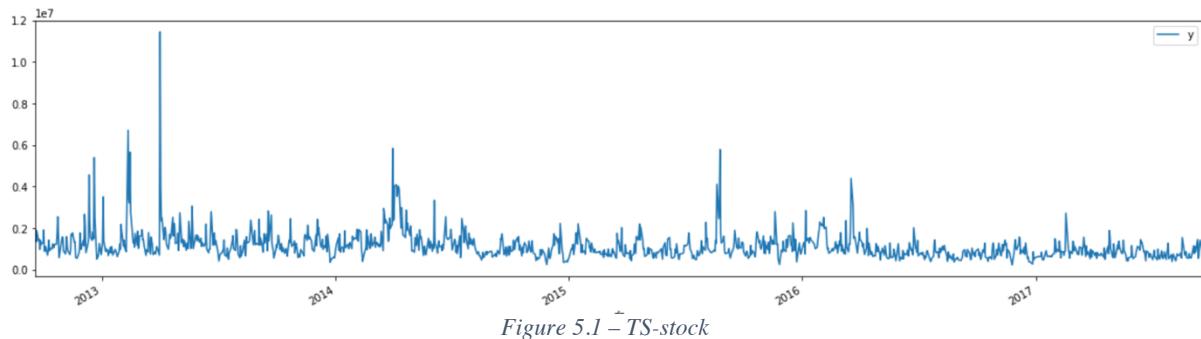


Figure 5.1 – TS-stock

2.3.4 Commentaire sur les jeux de données utilisées dans ce mémoire

Comme nous pouvons le constater, les caractéristiques des trois jeux de données sont assez différentes. Certaines semblent montrer une certaine régularité d'autre moins. Nous pouvons

légitimement penser que ces caractéristiques vont contraindre la démarche d'analyse et de prédiction.

2.4 Contour du « produit minimum viable » à livrer

Les données affichées aux niveaux de la plateforme proviennent essentiellement de data frame au sens de la librairie [Pandas](#). Ainsi les interfaces à livrer fourniront en sortie un data frame Pandas.

Le produit à livrer consistera en un module python appelé : **jpddsforecasting** fournissant une API permettant l'exécution d'un « pipeline » de forecasting.

Ce module devrait être déployé sous [Pypi](#) et installable avec « [pip](#) ». Le code source devrait être facilement consultable via le projet GitHub jetpackdata.

Par ailleurs, toutes les expérimentations faites durant ce projet ont été publiées sous forme de « note book » Jupyter accessible via le GitHub du projet : [lien](#).

3 Modèles statistiques pour la prédition des time series

Dans cette section, nous suivrons la démarche qui a permis de sélectionner les modèles statistiques pour l'implémentation et la livraison d'un produit minimum viable (« MVP »).

Nous aborderons certains détails concernant les modèles étudiés sans pour autant abuser d'un formalisme mathématiques rigoureux afin de rester accessible au plus large public.

Cette partie est librement inspirée de plusieurs blogs dont [3] ainsi que des articles suivants [4] [1]. Elle sera organisée de deux sous parties :

- Une première consacrée à l'exploration des données
- une deuxième présentant des expérimentations sur des modèles statistiques de forecasting.

3.1 *Introduction à la statistique des series temporelle et exploration des séries temporelles*

3.1.1 Notion de stationnarité

L'une des caractéristiques des time series rendant difficile tout exercice d'analyse et de prédition est la **non-stationnarité**.

Prenons l'exemple des ventes mensuelles de champagne :

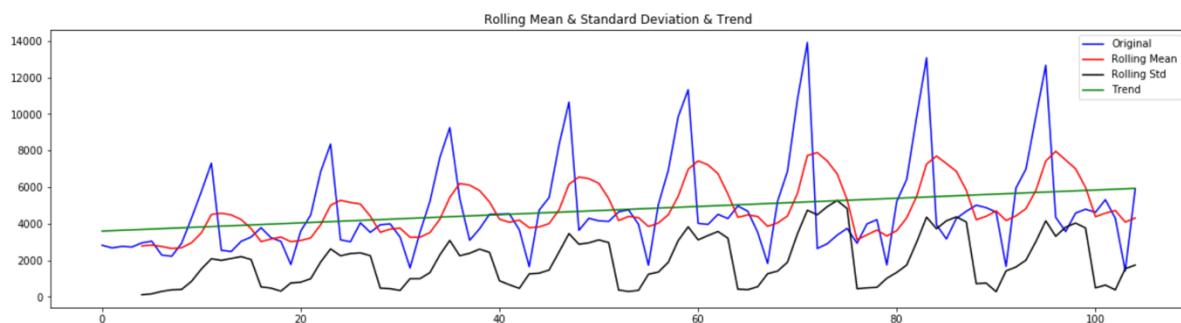


Figure 6.1 – Vente de champagne non stationnaire

La **non-stationnarité** est intimement lié à la dépendance intrinsèque des données de la série. Si la dynamique temporelle influe sur la distribution de la donnée, la série est dite non-stationnaire.

La série est stationnaire si ses sous séries décalées de k périodes suivent la même loi statistique. En d'autres termes, si nous décalons de k périodes (ou pas de temps) la série, le processus aléatoire générant les données n'est pas impacté. Ce qui est en pratique impossible mais approximativement vrai sous certaines hypothèses.

La difficulté liée à ce type de série est essentiellement dû au fait que les algorithmes de régression ou de filtration utilisé dans l'analyse de ces séries ne peuvent plus être calibrés facilement car leurs paramètres ne suivent plus une loi statistique facilement estimable. Il est donc nécessaire de tenter de « stationnariser » ce type de séries.

Il existe une caractérisation plus faible de la stationnarité. En effet, elle se manifeste par une moyenne mobile et un écart type mobile constant non dépendant du temps. Elle se manifeste également par une autocorrélation stable autour de 0.

Rappelons qu'un écart type représente la racine carrée de la moyenne du carré des écarts à la moyenne et permet de mesurer la dispersion par rapport à la moyenne.

Rappelons qu'une autocorrélation au lag k d'une série temporelle correspond à la corrélation de la série avec une version d'elle-même décalée de k périodes ou pas de temps.

Les courbes de « Rolling std » (ecart type modile) et de « Rolling mean » (moyenne mobile) montrent que la série est non stationnaire.

Le graph de l'autocorrélation de notre jeu de données permet d'identifier une corrélation positive au lag 1 en considérant le seuil de corrélation à +/- 0.20.

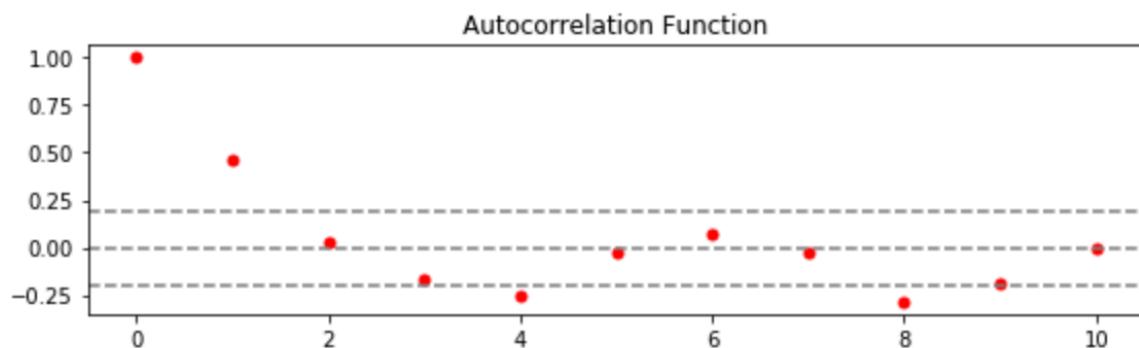


Figure 6.2 - Autocorrélation de vente de champagne

Pour tenter de réduire la non-stationnarité et pour rendre compte également des petites variations, nous allons stabiliser la variance en appliquant une fonction logarithmique à notre jeu de données puis nous réexaminerons de nouveau la fonction d'autocorrélation :

```
TSvol_log = np.log(TSvol+1)  
acf_pacf_plot(TSvol_log, 10)
```

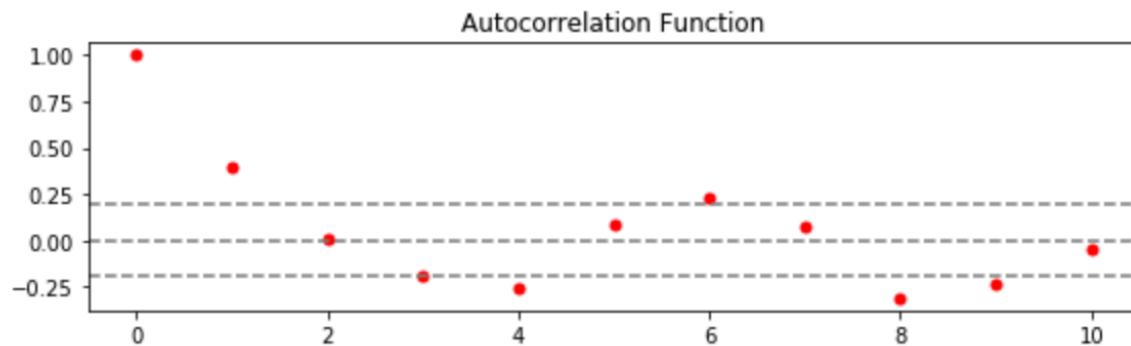


Figure 6.3 - Autocorrélation après transformation logarithmique

Nous confirmons ainsi l'autocorrélation au lags 1 ainsi que la non-stationnarité du logarithme également. Nous effectuerons par la suite notre étude sur le logarithme, de meilleure qualité, en gardant à l'esprit qu'il faudrait effectuer des transformations inverses afin de retrouver la série de départ.

Il existe par ailleurs, un test statistique, dit de Dickey-Fuller, permettant de confirmer la stationnarité d'une série avec une certaine incertitude [5].

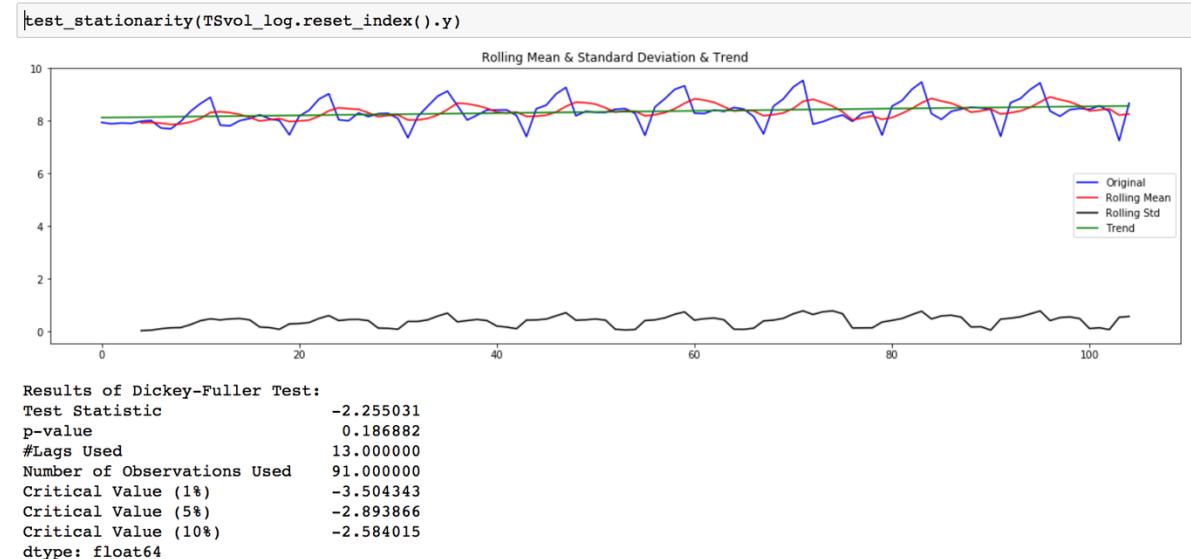


Figure 6.4 – Test de stationnarités

Afin d'avoir une série stationnaire à 99%, il faudrait que la valeur du test statistique soit inférieure au Critical value (1%). Ce qui n'est pas le cas pour notre jeu de test actuel.

3.1.2 Décomposition d'une série non-stationnaire

La courbe de « trend », représentant la tendance, montre une manifestation à la hausse certes atténuée par l'effet du logarithme. Les piques de la courbe « original » (représentant le logarithme de la série de départ) sont séparés d'un intervalle de douze mois ce qui laisse entendre une certaine saisonnalité dans les données.

Ces deux composantes typiquement non-stationnaires peuvent être, l'une ou l'autre, traitées séparément par des modèles ad hoc. Ainsi, en supposant une certaine structure (additive ou multiplicative) sur la série d'origine, nous pouvons effectuer une décomposition de cette série.

Le choix de la décomposition est lié à l'expérimentation. Notons qu'un choix additif sur la série logarithmique induit une hypothèse de structure multiplicative sur la série d'origine.

Supposons une structure multiplicative sur la série d'origine et décomposons la série logarithmique :

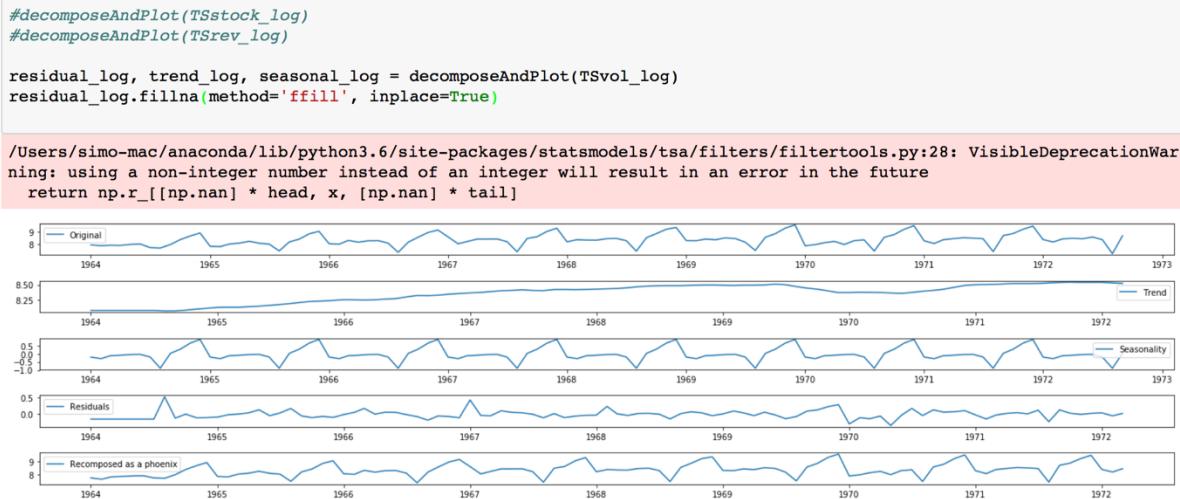


Figure 6.5 – Décomposition d'une time series

Faisons maintenant un test de stationnarité sur le résiduel correspondant à la série logarithmique sans sa composante saisonnière et de tendance.

```
test_stationarity(residual_log.reset_index().y)
```

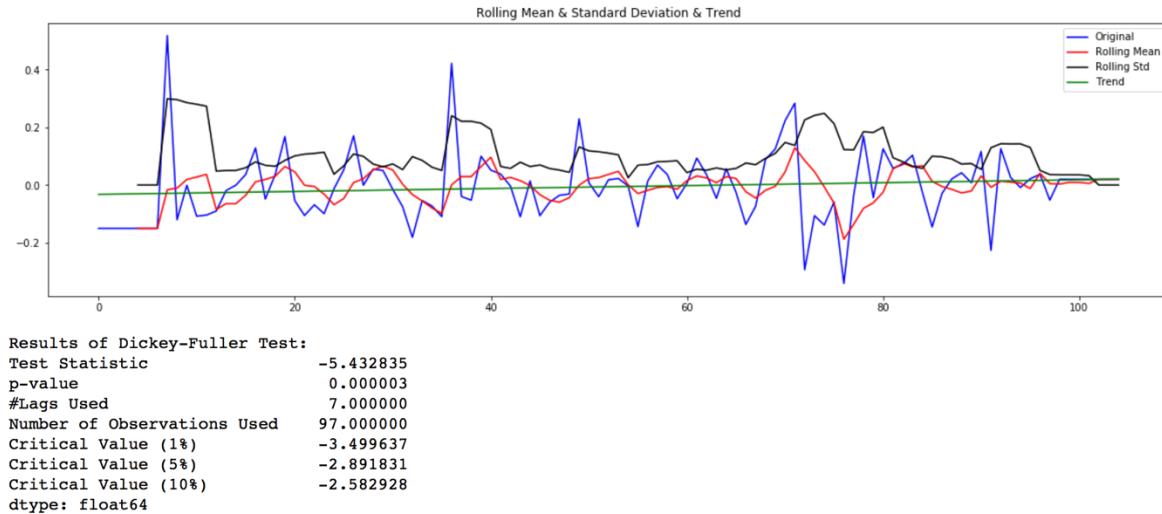


Figure 6.6 – Test de Dikey-Fuller

Nous constatons un test statistique positif à 99% (-5,4 étant inférieur à -3,4).

Nous venons de décrire comment extraire une composante stationnaire de la série temporelle sur laquelle un modèle à régression pourrait agir avec efficacité. Nous avons aussi extrait deux composantes pouvant être traitée par des modèles d'extrapolation en utilisant de l'analyses spectrale ou des projections sur un espace fonctionnel bien choisi.

3.2 Expérimentation et sélection de modèles de prédiction

Dans cette section nous décrirons les expérimentations qui ont abouti à la livraison d'un module prédictif pour la plateforme JetPack Data.

3.2.1 Expérimentation avec le modèle ARIMA [6] [7]

Il existe des méthodes de lissage permettant de détecter des changements de tendances sur un futur à court terme. Ses méthodes peuvent être perçus comme des filtres projectifs permettant de montrer une facette pertinente de la série de données qui peut se prolonger dans le futur proche.

Ces méthodes dérivent d'une méthode plus générale dite ARIMA pour Auto Regressif Integrated Moving Average.

Expliquons les principes de ce modèle puis examinons quelques expérimentations effectuées avec ce dernier :

3.2.1.1 *Explications du modèle ARIMA*

Un modèle ARIMA est étiqueté comme **ARIMA (p, d, q)**, dans lequel **p** est le nombre de termes **autorégressifs**, **d** est le nombre de **différenciations** à effectuer et à **réintégrer**, **q** est le nombre de termes pour la **moyenne mobiles**.

- Les processus autorégressifs supposent qu'une prédition est la somme pondérée d'un historique contigu de taille **p** auquel s'ajoute un aléa. Si l'historique est composé de deux réalisations, nous sommes en présence d'un **ARIMA(2,_,_)**.
- La stationnarité de la série est primordiale pour le bon déroulement de la méthode. Une série non-stationnaire peut être différenciée pour obtenir une série stationnaire. Elle peut également être redifférenciée un certain nombre de fois avant de devenir stationnaire. C'est exactement l'idée derrière le processus d'intégration qui peut être vu comme un cumul de différenciation permettant de reconstruire la série de départ.
- Les processus de moyenne mobile supposent que chaque prédition est fonction des erreurs entachant les prédictions précédentes auxquelles s'ajoute sa propre erreur de prédition. Si on considère trois réalisations, on sera dans le cas d'un **ARIMA(,_,3)**

Il est souvent préférable de soustraire la tendance en amont pour tester la stationnarité d'une série. Ainsi, nous sommes souvent en présence d'un résiduel d'une série qui n'a plus besoin de différenciation. Le paramètre **d** sera souvent zéro.

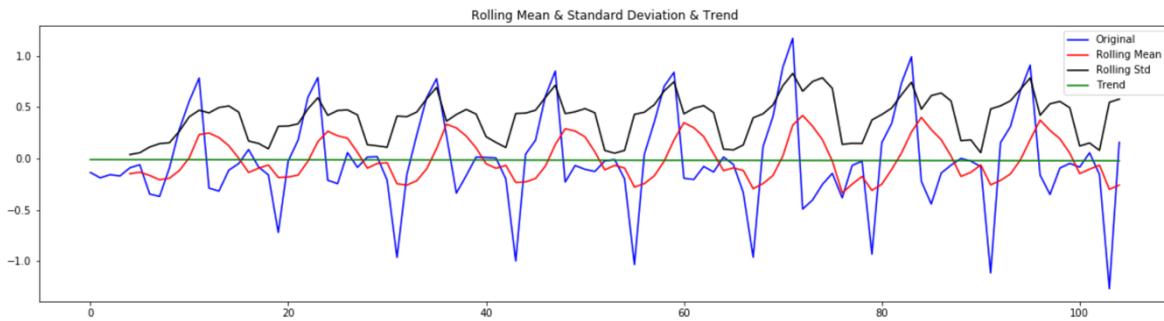
Passons maintenant au calibrage du modèle ARIMA qui consiste à choisir les différents paramètres afin d'avoir un modèle robuste permettant de bonne prédition.

3.2.1.2 *Expérimentation ARIMA sur un exemple concret (Jupyter Notebook)*

Pour cette expérience, nous avons pris le parti de garder la composante saisonnière dans le résiduel car nous avons constaté qu'une soustraction de la tendance suffit pour avoir une certaine stationnarité comme le prouve le test de Dickey-Fuller suivant.

Ce parti pris consiste à considérer, sous condition de stationnarité, qu'un modèle régressif sur les réalisations des valeurs de la série sera plus robuste qu'une analyse spectrale paramétrique.

```
shift_log = TSvol_log - trend_log
test_stationarity(shift_log.reset_index().y)
```



```
Results of Dickey-Fuller Test:
Test Statistic          -4.029747
p-value                 0.001263
#Lags Used             11.000000
Number of Observations Used 93.000000
Critical Value (1%)    -3.502705
Critical Value (5%)    -2.893158
Critical Value (10%)   -2.583637
dtype: float64
```

Figure 7 – Logarithme et test de stationnarité

Le test de Dickey-Fuller confirme la stationnarité à 99% (-4.029 est inférieur à -3.50).

Afin de déterminer les paramètres du modèle, nous allons exécuter une grille ARIMA en combinant le nombre de termes autoregressifs allant de un à quatre avec le nombre de termes pour la moyenne mobile allant aussi de un à quatre.

Nous visualiseront par la suite l'écart type (RMSE : root mean square error) représentant l'erreur entre le résiduel et la prédition sur l'ensemble de la série de données :

```
AR = [1, 2, 3, 4]
IT = [0]
MA = [1, 2, 3, 4]
rmse_scores = []
step = -1

global_start_time = time.time()
for lag_ar in AR:
    for diff_st in IT:
        for lag_ma in MA:
            step +=1
            try:
                rmse_orig = arima_fcast(shift_log, trend_log, lag_ar=lag_ar, diff_st=diff_st,
                                         lag_ma=lag_ma, loga=True, plot=False)
            except:
                pass

            rmse_scores.append(rmse_orig)
            print("Iteration number :" + str(step) + " : Loss :" + str(rmse_orig)
                  + " for : arima_" + str(lag_ar) + "_" + str(diff_st) + "_" + str(lag_ma))

print('TRAINING DURATION (s) : ', time.time() - global_start_time)

Iteration number :0: Loss :2168.6515202667633 for : arima_1_0_1
Iteration number :1: Loss :2012.195218242848 for : arima_1_0_2
Iteration number :2: Loss :2008.4641633771614 for : arima_1_0_3
```

```
plt.figure(1, figsize=(20, 5))
plt.plot(pd.Series(rmse_scores).rolling(window=int(len(AR)*len(MA)/7),center=False).mean(), color='blue', label = 'Loss')
plt.title('Loss')
plt.legend(loc='best')
plt.show(block=False)
```

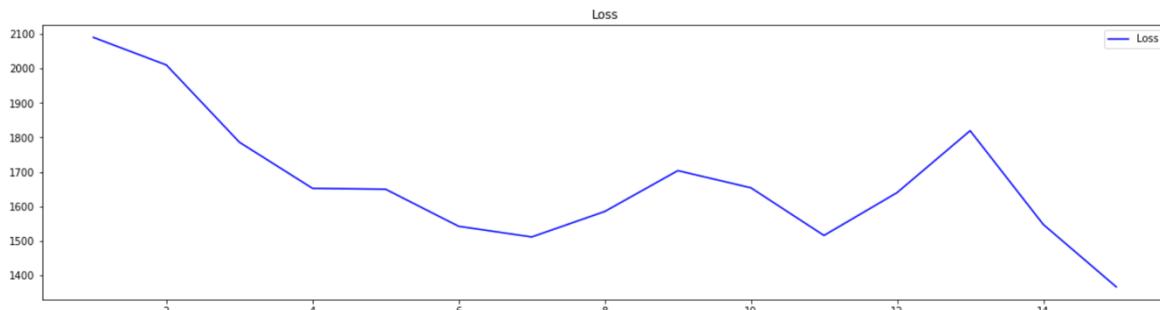


Figure Loss Optimizition

Nous pouvons considérer qu'à l'itération quatorze, nous avons atteint un RMSE minimal.

Iteration number : 14: Loss : 1278.989741401742 for : arima_4_0_3

Ainsi **p=4** et **q=3**.

Il est possible de déterminer **p** et **q** avec les fonctions d'autocorrélation, et d'autocorrélation partielles. [7] donne quelques explications mais cet exercice reste difficile et demande de l'expérience.

```
#acf_pacf_plot(TSrev_log.y, lags=100)
#acf_pacf_plot(TSstock_log.y, 300)
acf_pacf_plot(shift_log, 20)
```

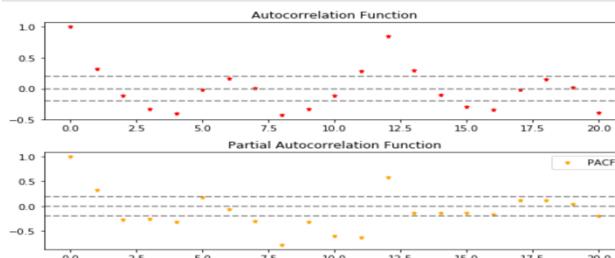


Figure 7.2 - Acp- pAcf

L'autocorrélation partielle correspond à la corrélation intrinsèque de la série avec elle-même décalé de k période en ne tenant pas compte des contributions de corrélations des périodes intermédiaires (les k-1nième).

Examinons maintenant la qualité de prédiction du modèle **arima_4_0_3** : **1278.989741401742**.

```
rmse = arima_fcast(shift_log, trend_log, lag_ar=4, diff_st=0, lag_ma=3)
```

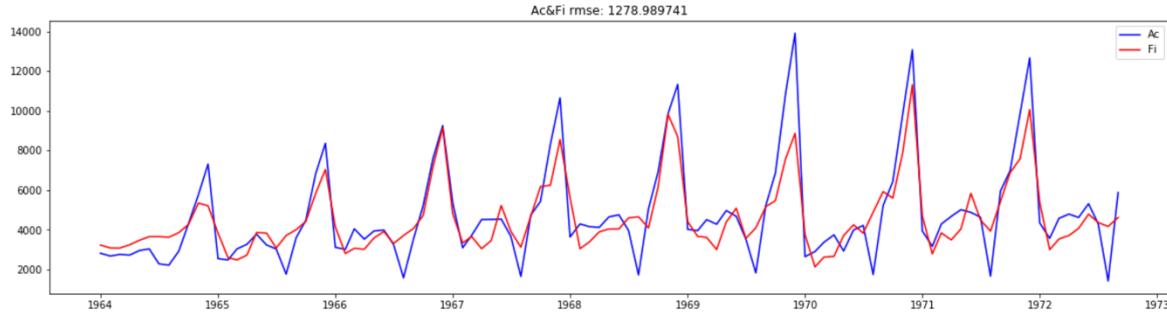


Figure 8 – ARIMA(4,0,3)

Cette dernière figure représente la série finale reconstruite après prédiction. La courbe « Ac » correspond à la série actuelle d'origine et la courbe « Fi » correspond à la courbe ajustée.

3.2.2 Expérimentation avec le module Prophet de Facebook

Les choix qui ont abouti au développement du module [prophet](#) sont documentés dans l'article [1].

Les besoins de Facebook en termes de forecasting sont intrinsèquement liés à l'utilisation des réseaux sociaux et aux données économiques.

Prophet suppose une structure additive de la time series données comme entrée :

Une tendance avec des points de changements et une saturation de tendance

Une saisonnalité extraite en utilisant une analyse [spectral](#).

Les congés modélisés par des fonctions caractéristiques contrignant le calibrage

Un bruit stationnaire

L'ajustement des paramètres du modèle se fait à l'aide d'un algorithme d'approximation basé sur des chaînes markoviennes et des simulations Monte Carlo ([MCMC](#)). La librairie [Stan](#) est utilisée pour implémenter cet algorithme.

Ces choix font de prophet une librairie efficace pour différents types de time series.

Une illustration dans l'article de Facebook mérite le détour :

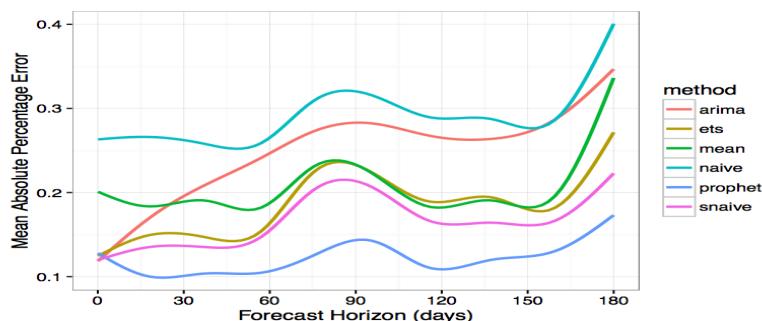


Figure 9 – Comparaisons Modèle

Les courbes arima/prophet présentent un ratio d'environ $0,27/0,17 = 1,8$.

Vu les fonctionnalités de Prophet, nous avons décidé de l'intégrer au produit livré.

Nous avons par ailleurs décidé d'intégrer également les modèles « sma » (Simple Moving Average) et « ewma » (Exponentiel Weighted Moving Average) vu leur notoriété chez les utilisateurs de JetPack Data.

3.2.3 Description du produit livré

Le module a été publié en open source sur GitHub : [Module jpddsforecasting](#). Il est compatible pour l'instant avec la version Python 2.

Il est installable via l'outil pip comme suit :

```
→ ~ pip2 install jpddsforecasting
Requirement already satisfied: jpddsforecasting in /usr/local/lib/python2.7/site-packages
Requirement already satisfied: fbprophet in /usr/local/lib/python2.7/site-packages (from jpddsforecasting)
Requirement already satisfied: sklearn in /usr/local/lib/python2.7/site-packages (from jpddsforecasting)
Requirement already satisfied: scipy in /usr/local/lib/python2.7/site-packages (from jpddsforecasting)
Requirement already satisfied: pystan>=2.14 in /usr/local/lib/python2.7/site-packages (from fbprophet->jpddsforecasting)
Requirement already satisfied: pandas>=0.18.1 in /usr/local/lib/python2.7/site-packages (from fbprophet->jpddsforecasting)
Requirement already satisfied: scikit-learn in /usr/local/lib/python2.7/site-packages (from sklearn->jpddsforecasting)
Requirement already satisfied: numpy>=1.8.2 in /usr/local/lib/python2.7/site-packages (from scipy->jpddsforecasting)
Requirement already satisfied: Cython!=0.25.1,>=0.22 in /usr/local/lib/python2.7/site-packages (from pystan>=2.14->fbprophet->jpddsforecasting)
Requirement already satisfied: python-dateutil in /usr/local/lib/python2.7/site-packages (from pandas>=0.18.1->fbprophet->jpddsforecasting)
Requirement already satisfied: pytz>=2011k in /usr/local/lib/python2.7/site-packages (from pandas>=0.18.1->fbprophet->jpddsforecasting)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python2.7/site-packages (from python-dateutil->pandas>=0.18.1->fbprophet->jpddsforecasting)
→ ~ python2 -m jpddsforecasting
```

Une démonstration avec le jeu de donnée de vente de champagne permet d'avoir une idée sur l'utilisation de ce module :

```
import jpddsforecasting.pipelineforecasting as pplprocess
forecast_data = pplprocess.run_forcast(TSvol, config)
```

Nous avons utilisé une configuration en « Simple Moving Average » avec un historique glissant de six mois pour prédire 24 mois :

```
(La documentation GitHub permet d'expliquer les différents attributs techniques)
config = {
    'id_model': '0001',
    'model': 'sma',
    'date': tb.get_now(),
    'window_size': 6,
    'prediction_conf': {'future_period': 24, 'freq': 'M'},
    'tech_conf': {'clean': True}
}
```

Nous avons obtenu le résultat suivant :

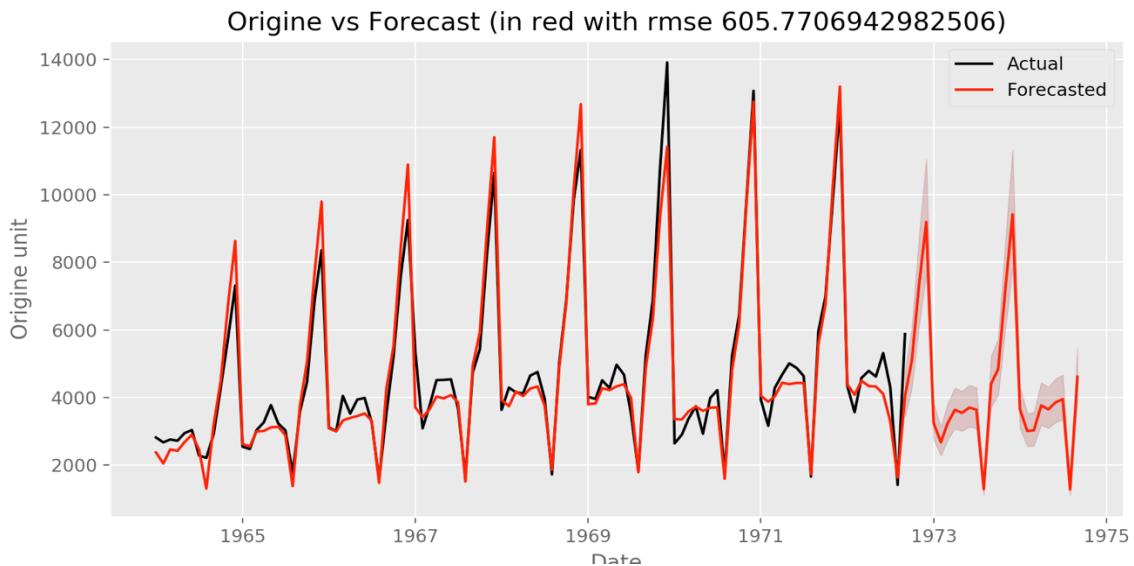


Figure 10.1 – sma - prédition de vente de Champagnes à 24 mois avec un historique de 6 mois

Une moyenne mobile sur la série originale ou logarithmique n'aurait pas donné ce type de résultat. En effet, sa performance se dégraderai très rapidement aux premiers trois, quatre mois. Nous avons donc appliqué la moyenne mobile simple uniquement sur le résiduel. Les composantes de saisonnalité et de tendance ont été traitées et prédites à l'aide du modèle de Facebook. Le même choix a été effectué pour le modèle « ewma ».

Faisons l'expérience avec la librairie ‘prophet’ intégrée dans le pipeline jpddsforecasting :

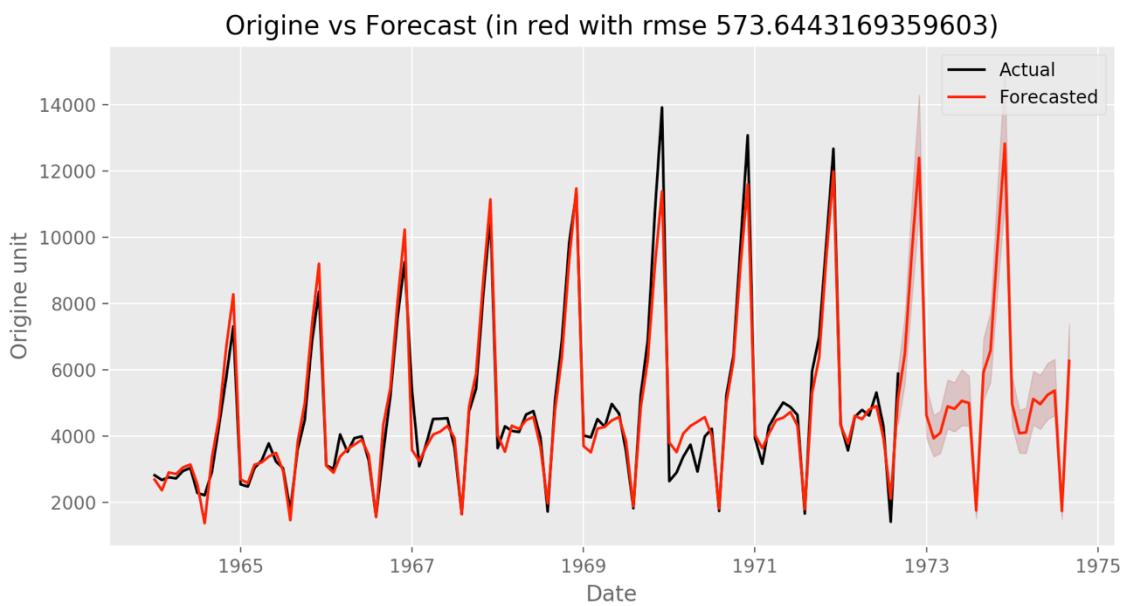


Figure 10.2 – Avec prophet

On remarque que la distribution de l'erreur est clairement meilleure pour ‘prophet’.

Calculons le rapport RMSE arima/prophet : il est environ de $1278/573 = 2,2$ ce qui est en adéquation avec la valeur trouvée dans le graphique de la figure 9 de Facebook : $\sim 0,27/0,17 = 1,8$.

Dans cette partie nous avons présenté les modèles statistiques utilisés pour analyser et prédire des séries temporelles. Nous avons aussi expliqué notre démarche qui a conduit à faire certains choix concernant le produit à livrer. Nous allons étudier maintenant d'autre type de modèles issus du monde du Machine Learning ou plus spécifiquement du Deep Learning.

Ces modèles peuvent être appliqués sur une large panoplie de séries temporelles. Ils peuvent donc intéresser la plateforme JetPack Data dans un futur proche.
Par ailleurs, leur étude constitue une très bonne application des connaissances acquises lors de la première partie du programme de certification.

4 Machine Learning, Deep Learning et time series

Dans cette partie nous parlerons des méthodes d'apprentissage profond ou du Deep Learning. La littérature foisonne de papiers autour de ces sujets. Le succès des réseaux de neurones depuis quelques années est sans équivoque et il est principalement dû à plusieurs facteurs concomitants dont principalement la disponibilité de données sous format numérique et la puissance calculatoire. Les réseaux de neurones deviennent incontournables dans l'intelligence artificielle vu leur efficacité.

Dès lors, il n'est pas question de détailler le fonctionnement des réseaux de neurones mais de principalement se focaliser sur les aspects pratiques nous permettant une application rapide aux time series.

L'ouvrage [8] permet une introduction efficace et pratique aux réseaux de neurones.

Pour une application sur des données temporelles, nous aurons besoin de réseaux à mémoire permettant la restitution d'un contexte.

Les RNN (Recurrent Neuronal Network) sont naturellement des candidats. Plus précisément, les LSTM (Long Short Term Memory) sont de bons candidats. Cet excellent blog [9] permet d'avoir une idée précise sur leur fonctionnement.

Nous allons dans cette partie commencer par un bref aperçu sur les RNN LSTM pour ensuite contextualiser leur utilisation et présenter quelques expérimentations et résultats.

4.1 Fonctionnement des RNN LSTM

4.1.1 Aperçu général d'un RNN

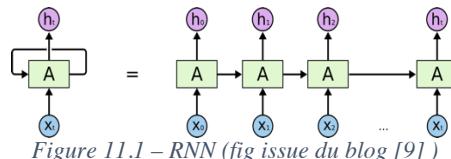


Figure 11.1 – RNN (fig issue du blog [9])

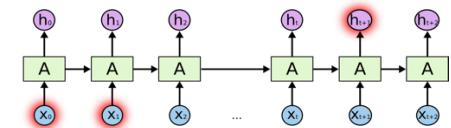


Figure 11.2 – Long Term Memory (fig issue du blog [9])

La Figure 1 est une représentation d'un RNN de base. Les opérations effectuées par un neurone à l'étape t sont affecté par le résultat immédiatement précédent. Ce procédé ne tient pas compte des résultats historiques long et moyen termes mais seulement du voisinage immédiat.

Ainsi par exemple, imaginons que nous voudrions prédire le mot d'une phrase simple : « La couleur du ciel est <> », un RNN bien entraîné avec le bon champ lexical permettrait de trouver la bonne couleur.

Cependant pour la phrase suivante: « Il n'arrête pas de pleuvoir depuis ce matin, la couleur du ciel est <> », un RNN simple risque d'être moins performant.

Il est donc nécessaire d'envisager d'autre type de RNN dont le LSTM.

4.1.2 Détail d'un neurone LSTM

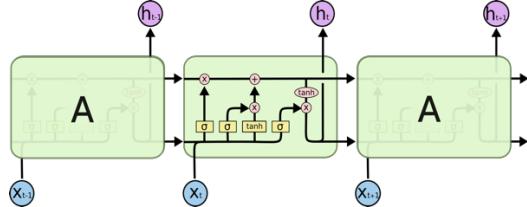


Figure 11.3 – LSTM (fig issue du blog [9])

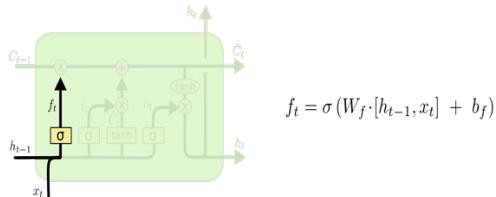


Figure 11.4 – LSTM focus

Dans les Figure 3, nous avons un ensemble de filtres interne à un neurone. On peut les classifier en trois sortes :

- Un filtre sur l'information long terme (filtre multiplicatif sigma entre 0 et 1).
- Un filtre sur l'information courante (filtre sigmoïde tanh entre -1 et 1).
- Un filtre pour la genèse de la sortie?

Il y a d'autres variantes d'un réseau LSTM plus ou moins complexes permettant de meilleures performances selon le problème étudié.

Pour approfondir le sujet, la thèse [10] étudie en détail les réseaux LSTM.

4.2 Réseau de neurones dans un contexte de time series

Dans notre contexte, nous utiliserons comme jeu de données le volume de vente de champagne (DS-volume). Nous étudierons par la suite un jeu de test plus stochastique correspondant à un stock financier (DS-stock).

Nous allons par ailleurs utiliser une méthode dite de « multi-step » forecasting permettant de prédire p prochaines étapes à l'aide de n étapes passées :

Prédir $h_t, h_{t+1}, \dots, h_{t+p}$ avec $x_{t-1}, x_{t-2}, \dots, x_{t-n}$

La validation du modèle sera effectuée à l'aide d'un « Rolling-Forecast » scenario (ou « Walk-Forward » validation model). Il s'agit de rendre disponible une valeur prédictive au modèle de prédiction agissant au pas de temps suivant. Cette procédure permet de simuler le cas réel classique permettant d'enrichir notre historique dès la réalisation et le relevé des ventes du mois achevé.

Nous évaluerons la NRMSE afin de mesurer l'apprentissage sur les jeux de données de validation.

Nous utiliserons la bibliothèque [Keras](#) qui est une surcouche de [TensorFlow](#).

4.3 Expérimentation autour des LSTM et des time series

Ces expérimentations sont accessibles via GitHub en format [note book Jupiter](#).

Nous devons tout d'abord transformer la time serie en des données exploitables par des algorithmes d'apprentissage puis définir le réseau LSTM à utiliser afin de commencer l'apprentissage.

4.3.1 Préparation des données pour de l'apprentissage supervisé

L'idée est de construire un nuage de point de dimension m en utilisant une fenêtre glissante de taille m. L'algorithme de construction consiste à faire glisser la fenêtre à chaque pas de temps et relevé les coordonnées du point.

Voici le code permettant ce type de traitement. La dimension est notée dim. Dans certain cas elle sera notée **n_seq** ou **window_size** car elle correspond aussi à la taille d'une séquence représentant un point ou la taille de la fenêtre glissante.

```
#ts to supervised learning
def prepareWithoutLabel(serie, dim):
    sequence_length = dim
    size = len(serie)
    result = []
    for i in range(size-sequence_length):
        result.append(serie[i:i+sequence_length])

    result = np.array(result)

    number_of_sequence = result.shape[0]
    train_set_size = round(0.8 * number_of_sequence)

    train_set = result[:int(train_set_size), :]
    x_train = train_set[:, :]
    x_test = result[int(train_set_size):, :]

    return [x_train, x_test]
```

Il est à noter que l'on a saisi l'occasion pour définir le jeu de test pour les besoins de validation. J'ai opté pour 20% de points de validation d'où le coefficient 0.8 dans le code. La part du training set utilisé pour la cross validation, sera définie lors du fit du modèle.

Afin de préparer les données au réseau LSTM, certaines opérations de normalisation et de différentiation sont nécessaires pour permettre au réseau de fonctionner correctement.

En effet le calibrage des filtres neuronaux demande des valeurs entre -1 et 1 pour pouvoir appliquer une sigmoïde d'une manière homogène sur les données d'entraînement.

Par ailleurs, la différentiation permet de s'affranchir de la tendance lors de l'apprentissage car il est difficile d'apprendre une tendance. Le blog suivant permet de clarifier [11] ce point. La complexité provient principalement de la structure des filtres qui projettent les données dans l'intervalle [-1,1] ce qui rend l'apprentissage de la tendance problématique si nous ne connaissons pas à l'avance les bornes des données futures.

La normalisation et la différenciation sont réalisées par les routines suivantes :

- **MinMaxScaler** de scikit-learn pour la normalisation
- **Shiftter** pour la différenciation

La fonction **prepare_for_lstm** permet d'avoir un jeu d'entraînement et de test prêt pour l'apprentissage. Il faudra par la suite effectuer toutes les transformations inverses pour l'évaluation du modèle.

```
# Diff & Normalisation for series. Return train and test sets ready for supervised learning
def prepare_for_lstm(series, n_seq):
    # extract points
    points = series.values

    # transform data to be stationary
    diff_series = shiftter(points)
    diff_values = diff_series.values
    diff_values = diff_values.reshape(len(diff_values), 1)

    # rescale values to -1, 1
    scaler = MinMaxScaler(feature_range=(-1, 1))
    scaled_values = scaler.fit_transform(diff_values)
    scaled_values = scaled_values.reshape(len(scaled_values), 1)

    # transform into supervised learning problem X, y
    X_train, X_test = prepareWithoutLabel(scaled_values,n_seq)
    return scaler, X_train, X_test
```

4.3.2 Préparation des données pour de l'apprentissage d'un réseau LSTM

L'apprentissage se fait via la fonction **fit_lstm**. Les paramètres de modèles peuvent être classifiés en deux catégories.

La première catégorie est liée à l'exercice même du forecasting comme la taille de la fenêtre (**n_seq** ou **window_size**), la taille **n_lag** du vecteur d'entraînement qui dé facto définit la période future à prédire. Prenons l'exemple des données représentant la vente mensuelle de champagne DS-volume:

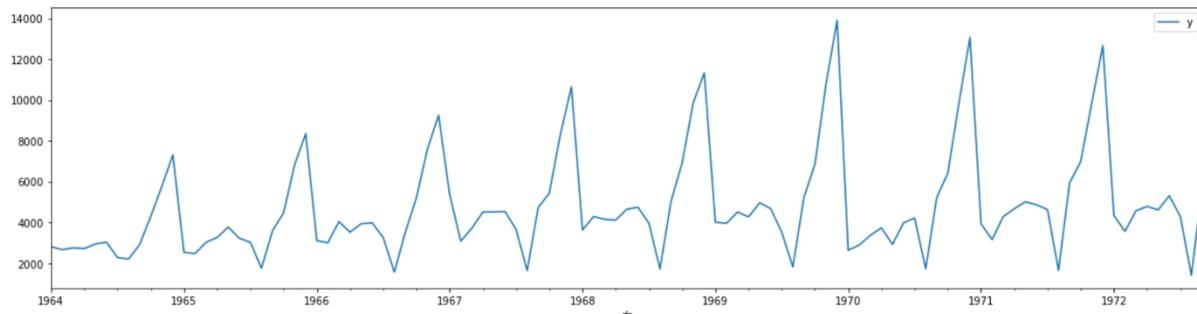


Figure 12.1

Si nous voulons prédire la vente des trois prochains mois, il faudrait choisir **window_size** et **n_lag** tel que leur différence soit égale à trois. On pourra par exemple faire une expérience dans un premier temps avec : **window_size = 4, n_lag = 1**.

Ce qui veut dire, que nous allons construire un nuage de points à quatre dimensions et que nous tenterons d'apprendre les liens existant entre la première dimension et les trois autres. Il serait intéressant par la suite d'agir sur ce type de paramètre pour voir l'impact sur l'apprentissage.

La seconde catégorie correspond aux paramètres d'apprentissage et à la topologie du réseau. Nous utiliserons un réseau à une couche cachée en paramétrant au besoin le nombre de neurones. Nous commencerons par : **n_neurons = 10**.

Comme décrit précédemment, nous utiliserons du multi-steps forecasting avec une réinjection des valeurs observées pour les prédictions suivantes. Ce procédé est assimilable à de l'apprentissage en ligne (Online Learning) et non en batch ou mini-batch. Ainsi à chaque itération les poids du modèle seront ajustés selon un algorithme d'optimisation de type « descente du gradient » ([12] pour l'« optimizer » réellement utilisé et [8] pour la descente du gradient).

Nous utiliserons le paramètre **n_batch** à 1 qui permet de répondre à cette contrainte : **n_batch = 1**.

L'apprentissage du réseau dépendra aussi du nombre d'itérations effectuées sur l'ensemble du training set. Ce paramètre se nomme **nb_epoch**, il sera initialisé à 50 itérations : **nb_epoch = 50**

L'implémentation LSTM de Keras a besoin d'une structure de données spécifiques. En effet, le jeu d'entraînement doit être un tableau tridimensionnel (trainingSet, timeSteps, featuresSize) avec la sémantique suivante :

- **trainingSet** : l'ensemble des points du jeu d'entraînement, dans notre cas précis on est à 80% des points construit via la time series
- **timeSteps**: le pas de temps permettant une itération d'apprentissage, dans notre cas le pas est de 1 pour n'importe quel type de time series daily, monthly ou yearly.
- **featuresSize** : correspond à la dimension des points du training set, dans notre cas initial, ceci correspond à **n_lag = 1**

Afin d'accélérer les routines d'optimisation interne du réseau de neurone, on gardera l'état de chaque batch (chaque epoch dans notre cas vu que **n_batch = 1**). Ceci permettra à l'algorithme d'optimisation de partir de coefficients de meilleures qualités au niveau des neurones. Ainsi Nous utiliserons l'attribut **statefull=True** au niveau de la définition de la couche centrale.

```
# fit an LSTM network to training data
def fit_lstm(train, n_lag, n_batch, nb_epoch, n_neurons):
    # reshape training into [samples, timesteps, features]
    X, y = train[:, 0:n_lag], train[:, n_lag:]
    X = X.reshape(X.shape[0], 1, X.shape[1])
    y = y.reshape(y.shape[0], y.shape[1])

    # design network
    model = Sequential()
    model.add(LSTM(n_neurons, batch_input_shape=(n_batch, X.shape[1], X.shape[2]), stateful=True))
    model.add(Dense(y.shape[1]))
    model.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])
    # fit network
    print("Fitting.....")
    tbCallBack = keras.callbacks.TensorBoard(log_dir='./Graph', histogram_freq=0, write_graph=True, write_images=True)
    evaluator = model.fit(X, y, epochs=nb_epoch, batch_size=n_batch, verbose=1, shuffle=False, validation_split=0.1,
                          callbacks=[tbCallBack])
    print("LSTM Fitted")
    return model, evaluator
```

L'apprentissage du réseau est calibré en minimisant la fonction loss (la variance de l'erreur dans notre cas entre le vecteur à 3 dimensions attendues et le vecteur à 3 dimensions issues des filtres d'un neurone). La recherche du minimum, se fait à l'aide d'un algorithme dérivé de la descente du gradient stochastique [12] et [3].

4.4 Résultat des différents tests effectués

Nous allons commencer par faire varier dans un premier temps le volume d'entraînement pour ensuite étudier un autre type de série temporelle.

4.4.1 Variation du volume d'entraînement pour les ventes de champagne

Voici les résultats du premier essai effectué sur DS-volume avec les paramètres suivants :

window_size = 4

n_lag=1

n_batch=1

nb_epoch=50

n_neurons=10

```
n_test = X_test.shape[0]

# make forecasts
global_start_time = time.time()
forecasts = make_forecasts(model, n_batch, X_test, n_lag)
print('Forcast duration (s) : ', time.time() - global_start_time)
# inverse transform forecasts and test
forecasts = inverse_transform(TSvol.y, forecasts, scaler, n_test+window_size-n_lag)

actual = [row[n_lag:] for row in X_test]
actual = inverse_transform(TSvol.y, actual, scaler, n_test+window_size-n_lag)

# evaluate forecasts
evaluate_forecasts(actual, forecasts, window_size=n_lag)
# plot forecasts
plot_forecasts(TSvol.y, forecasts, n_test+window_size-n_lag)

Forcast duration (s) :  3.270462989807129
NRMSE average : 0.241613
```

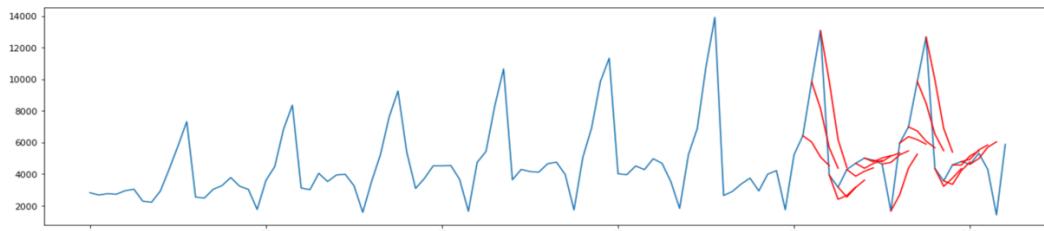


Figure 12.2 – prédiction à 3 mois des ventes de champagne

```
# summarize history for accuracy
plt.figure(figsize=(20,5))
plt.plot(evaluator.history['acc'])
plt.plot(evaluator.history['loss'])
plt.plot(evaluator.history['val_acc'])
plt.plot(evaluator.history['val_loss'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['Accuracy', 'Loss', 'Validation Accuracy', 'Validation Loss'], loc='upper left')
plt.show()
```

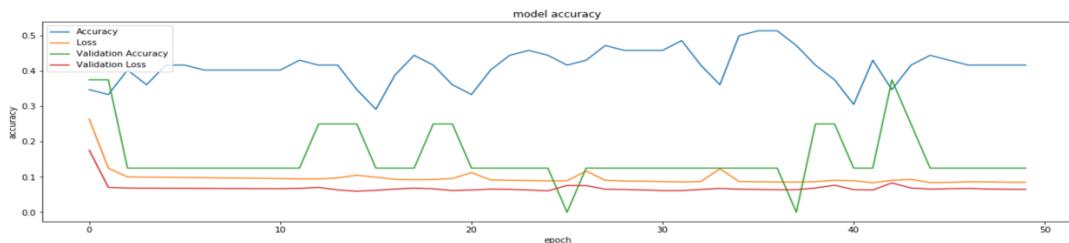


Figure 12.3 – Comportement du réseau à 10 neurones avec 50 itérations

Le réseau semble apprendre vue la décroissance des fonctions de coût. Son pouvoir prédictif reste à confirmer car le taux d'apprentissage n'est pas très explicite vu la structure des courbes Accuracy (précision des filtres neuronaux) et Validation Accuracy (taux d'apprentissage sur les jeux de données utilisés en cross validation).

Les tests finaux donnent une NRMSE en moyenne de 0.2416 qui est pour l'instant notre valeur de référence.

Notons par ailleurs que le temps d'apprentissage a été de 40.97 s.

Faisons maintenant un test avec 2000 itérations et 30 neurones ([Live TensorBoard](#)) :

```
# summarize history for accuracy
plt.figure(figsize=(20,5))

plt.plot(pd.Series(evaluator.history['acc']).rolling(window=100,center=False).mean())
plt.plot(pd.Series(evaluator.history['loss']).rolling(window=100,center=False).mean())
plt.plot(pd.Series(evaluator.history['val_acc']).rolling(window=100,center=False).mean())
plt.plot(pd.Series(evaluator.history['val_loss']).rolling(window=100,center=False).mean())
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['Accuracy', 'Loss', 'Validation Accuracy', 'Validation Loss'], loc='upper left')
plt.show()
```

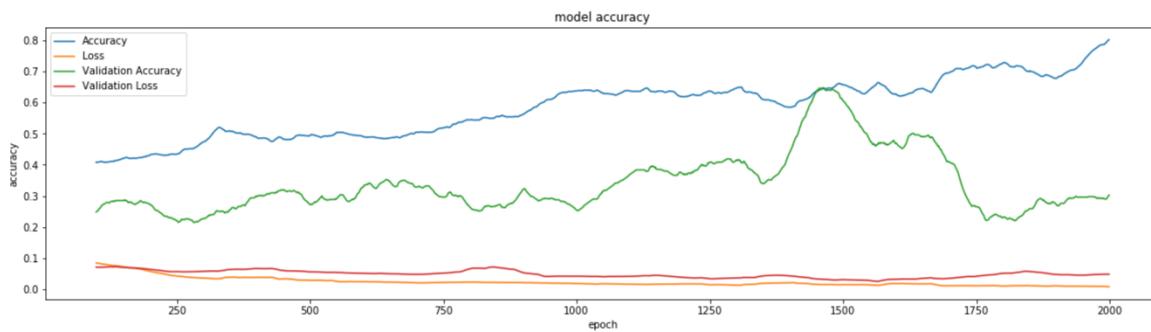


Figure 12.4

On observe clairement un taux d'apprentissage continu au fur et à mesure des itérations. Par ailleurs, la validation accuracy dépasse les 50% vers la 1500 ième iteration. Ce meilleur résultat est confirmé par les tests finaux avec une NRMSE en moyenne de 0.1163.

```

n_test = X_test.shape[0]

# make forecasts
global_start_time = time.time()
forecasts = make_forecasts(model, n_batch, X_test, n_lag)
print('Forecast duration (s) : ', time.time() - global_start_time)
# inverse transform forecasts and test
forecasts = inverse_transform(TSvol.y, forecasts, scaler, n_test+window_size-n_lag)

actual = [row[n_lag:] for row in X_test]
actual = inverse_transform(TSvol.y, actual, scaler, n_test+window_size-n_lag)

# evaluate forecasts
evaluate_forecasts(actual, forecasts, window_size=n_lag)
# plot forecasts
plot_forecasts(TSvol.y, forecasts, n_test+window_size-n_lag)

Forecast duration (s) : 4.946046829223633
NRMSE average : 0.116388

```

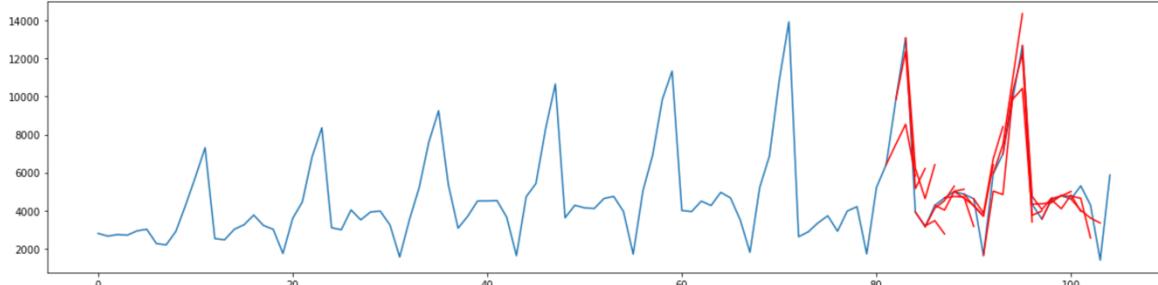


Figure 13

En examinant la courbe de cross validation, on remarque qu'à partir de 1500 itérations les performances se dégradent sur le jeu de validation. Un test avec 1500 itérations et toujours avec le même nombre de neurones confirmera cette hypothèse.

Différentes expériences ont été menées afin d'étudier le comportement du réseau.

Dans un premier temps, les dimensions et la taille des points d'entrainement ont été changés sur le même jeu de données. Nous remarquons une amélioration sensible en augmentant la taille des points d'entrainement ce qui est compréhensible vue la structure de l'autocorrélation de cette time series.

4.4.2 Réseau LSTM sur un stock financier

Nous allons maintenant expérimenter une time series de type stochastique (DS-Stock) :

```

# fit model
n_lag=5
n_batch=1
nb_epoch=100
n_neurons=10

global_start_time = time.time()
model, evaluator = fit_lstm(X_train, n_lag, n_batch, nb_epoch, n_neurons)
print('Training duration (s) : ', time.time() - global_start_time)

0.0013 - val_acc: 0.2100
Epoch 95/100
898/898 [=====] - 6s - loss: 0.0071 - acc: 0.2416 - val_loss: 0
0.0013 - val_acc: 0.2100
Epoch 96/100
898/898 [=====] - 6s - loss: 0.0071 - acc: 0.2416 - val_loss: 0
0.0013 - val_acc: 0.2100
Epoch 97/100
898/898 [=====] - 6s - loss: 0.0071 - acc: 0.2416 - val_loss: 0
0.0013 - val_acc: 0.2200
Epoch 98/100
898/898 [=====] - 6s - loss: 0.0071 - acc: 0.2428 - val_loss: 0
0.0013 - val_acc: 0.2100
Epoch 99/100
898/898 [=====] - 7s - loss: 0.0071 - acc: 0.2416 - val_loss: 0
0.0013 - val_acc: 0.2100
Epoch 100/100
898/898 [=====] - 6s - loss: 0.0071 - acc: 0.2416 - val_loss: 0
0.0013 - val_acc: 0.2100
LSTM Fitted
Training duration (s) : 613.3668029308319

```

Figure 14.1 – Calibrage du modèle pour une time serie de type stock – Processus stochastique



Figure 14.2 – Evaluation du modèle et cross validation

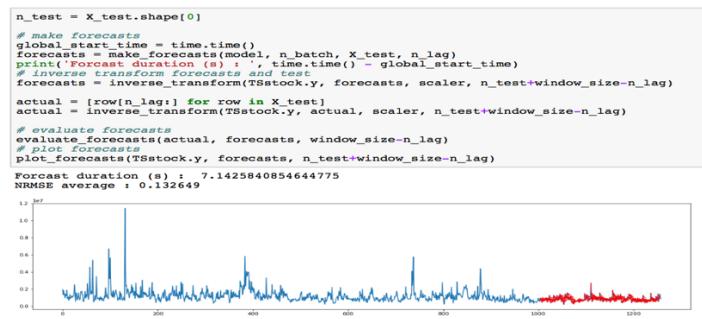


Figure 14.1 – Résultat sur le jeu de tests

Le résultat sur le jeu de test semble avoir une bonne performance. Cependant, nous sommes en présence d'un cas flagrant de sur-apprentissage. En effet, les données de cross-validation et notamment les fonctions de coût sont proches de zéro et ne varient pas trop. Par ailleurs, la validation accuracy n'est pas prometteuse. Dans ce cas, il est nécessaire de faire d'autres expériences.

Ajouter des neurones ou des couches cachées risque de renforcer le sur-apprentissage car nous augmentons le biais lié à la complexité du modèle.

Dans un premier temps, nous pourrons améliorer le jeu de données en effectuant une régularisation. Ceci permet de mettre sous contrainte les poids des filtres afin de mieux approcher la valeur cible.

Nous ne tiendrons pas compte également d'un certain nombre de neurone lors de l'apprentissage mais nous les utiliserons lors de la cross validation et du test. Ces deux optimisations sont possibles grâce à l'utilisation des attributs `kernel_regularizer`, `activity_regularizer` et `dropout` :

```

# fit an LSTM network to training data
def fit_lstm(train, n_lag, n_batch, nb_epoch, n_neurons):
    # reshape training into [samples, timesteps, features]
    x, y = train[:, 0:n_lag], train[:, n_lag:]
    x = X.reshape(X.shape[0], 1, X.shape[1])
    y = y.reshape(y.shape[0], y.shape[1])

    # design network
    model = Sequential()
    model.add(LSTM(n_neurons, batch_input_shape=(n_batch, X.shape[1], X.shape[2]), stateful=True,
                  dropout=0.4, kernel_regularizer=regularizers.l2(0.01),
                  activity_regularizer=regularizers.l1(0.01)))
    model.add(Dense(y.shape[1]))
    model.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])
    # fit network
    print("Fitting.....")
    tbCallBack = keras.callbacks.TensorBoard(log_dir='./Graph', histogram_freq=0, write_graph=True,
                                              write_images=True)
    evaluator = model.fit(x, y, epochs=nb_epoch, batch_size=n_batch, verbose=1, shuffle=False,
                           validation_split=0.1,
                           callbacks=[tbCallBack])
    print("LSTM Fitted")
    return model, evaluator

```

Voici les résultats obtenus :

```

print(evaluator.history.keys())
# summarize history for accuracy
plt.figure(figsize=(20,5))

plt.plot(pd.Series(evaluator.history['acc']).rolling(window=5,center=False).mean())
plt.plot(pd.Series(evaluator.history['loss']).rolling(window=5,center=False).mean())
plt.plot(pd.Series(evaluator.history['val_acc']).rolling(window=5,center=False).mean())
plt.plot(pd.Series(evaluator.history['val_loss']).rolling(window=5,center=False).mean())
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['Accuracy', 'Loss', 'Validation Accuracy', 'Validation Loss'], loc='upper left')
plt.show()

dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])

```

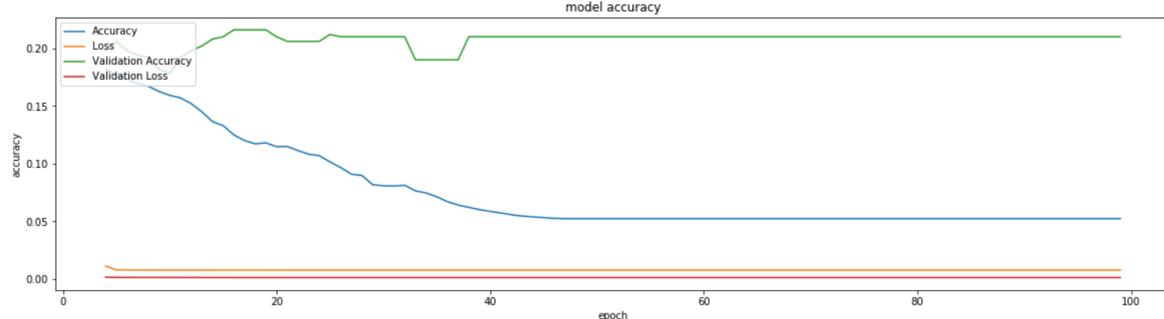


Figure 15 Validation croisée avec dropout et régularisation

L'utilisation des RNN LSTM pour la prédiction des stocks financiers à des fins d'investissement est à appréhender avec précaution. En effet, il existe des risques avérés de sur-apprentissage.

L'utilisation de modèles de volatilité historique et/ou implicite peuvent éventuellement aider à contraindre davantage les poids des filtres internes des neurones afin d'avoir un calibrage plus précis.

4.4.3 Récapitulatif sur les réseaux LSTM

Ainsi, nous avons expérimenter l'application des réseaux de neurones aux time séries sans pour autant couvrir la totalité des possibilités et des optimisations possibles.

L'intégration les RNN LSTM au pipeline utilisé par JetPack Data est envisageable. Cependant, une attention particulière doit être prise lors de l'étape d'apprentissage du modèle. Il

conviendrait d'exécuter cette tâche au premier chargement des données dans la plateforme et mettre le maximum d'information en cache afin de rendre utilisable la fonctionnalité de forecasting.

Il faudrait par ailleurs connaître en amont, les time series intéressantes pour le client ou inférer toutes les time series pertinentes.

Une extension pourrait consister en un apprentissage multiple de différentes time series du jeu de données chargé dans la plateforme afin de préparer plusieurs modèles pour une utilisation ultérieure.

5 Critique des différents résultats obtenus

La mesure de performance choisie est la « Normalized Root Mean Squared Error ». Elle permet une normalisation de l'erreur pour comparer les modèles par types de séries temporelles.

5.1 Modèles statistiques

Afin de pouvoir comparer les modèles, nous allons utiliser une profondeur d'historique en nous basant sur les paramètres optimaux du modèle ARIMA pour la série TS-volume.

Nous allons considérer une profondeur d'historique de 4 pour toutes les séries afin de prédire la 5ième valeur.

5.1.1 Synthèses des résultats

NRMSE DataSet/Model	ARIMA ARIMA (4,0,3)	Prophet	SMA (avec tendance et saisonnalité de Prophet) Windows Size : 4	EWMA (avec tendance et saisonnalité de Prophet) Windows Size : 4
TS-volume (tendance et saisonnalité évidente)	0.1295	0.0458	0.0881	0.1303
TS-revenu (tendance et saisonnalité non évidente)	0.1459	0.1606	0.1203	0.0981
TS-Stock (stochastique)	0.0901	0.0694	0.0423	0.0369

Pour les séries à saisonnalité et tendance évidentes nous remarquons que la librairie Prophet de Facebook possède une performance 2 ou 3 fois meilleure que les autres méthodes. En effet, cette dernière a été conçue pour être robuste dans ces cas précis vu les besoins en analyse de séries temporelles de Facebook.

Par ailleurs, pour les séries de types « revenu économique » ou « stock financier » dont les processus sont plus complexes et aléatoires, les modèles régressifs sont de meilleure performance. En effet, l'historique immédiat joue un rôle important sur ce type de séries quand leur résiduel (hors tendance et saisonnalité) est localement stationnaire, ce qui est le cas de TS-revenu et TS-Stock.

5.2 Modèles de Deep Learning

Afin de pouvoir comparer les modèles pour chacune des séries, nous allons utiliser des vecteurs de taille 5 et nous prédirons les 2 dernières valeurs à l'aide des 3 premières.

5.2.1 Synthèses des résultats

NRMSE DataSet/Model	LSTM Cas 1	LSTM avec Dropout et Régularisation Cas 2
TS-volume (tendance et saisonnalité évidente) – 71/8 1700 itérations LSTM 30 nodes hidden layer	0.1424 	0.2550
TS-revenu (tendance et saisonnalité non évidente) – 259/29 500 itérations LSTM 10 nodes hidden layer	0.2178 	0.1901
TS-Stock (stochastique) – 901/101 100 itérations LSTM 10 nodes hidden layer	0.1311 	0.1498

Dans le tableau ci-dessus nous avons consolidé les NRMSE des différents jeux de données avec leurs courbes d'apprentissage.

Nous allons considérer dans un premier temps un cas de référence, cas 1, des réseaux de neurones LSTM sans optimisation.

Dans un deuxième temps, cas 2, nous allons contraindre le modèle en utilisant une régularisation puis réduire aléatoirement à chaque itération le nombre de neurones utilisées pour entraîner le modèle. Ceci permettra d'atténuer les effets de sur-apprentissage.

Pour ce faire nous utiliseront un « dropout » à 0.4.

- La courbe bleue correspond à l'évolution de la précision du modèle à chaque itération
- La courbe verte correspond à l'évolution de la précision de la cross-validation
- Les courbes rouges et oranges étant les fonctions de coût à minimiser

Le comportement du modèle sur la série TS-volume (avec saisonnalité et tendance évidente) révèle un taux d'apprentissage ascendant tout au long des 1700 itérations pour les cas 1 et 2.

Le réseau LSTM se comporte aussi bien qu'un modèle statistique si nous augmentons le nombre d'itérations et de neurones. Cependant, l'investissement en ressources de calculs serait plus conséquent, et pour ce type de série, la librairie Prophet fournit de très bons résultats avec moins de ressource de calcul.

Les résultats pour les séries à saisonnalité et tendance cachées ou stochastiques sont plus mitigés.

Les fonctions de coût restent constantes et proches de zéro. Nous sommes face à un cas de sur-apprentissage, probablement dû à la complexité du modèle et surtout à la taille des séries.

Nous constatons par ailleurs une constance de la précision tout au long des itérations dans le cas 1. Dans le cas 2, nous remarquons l'apparition d'une tendance ascendante suite à l'ajout du « dropout » et la « régularisation ». Les optimisations utilisées dans le cas 2 améliorent le modèle, cependant il est nécessaire d'avoir des données plus volumineuses afin de constater clairement cette amélioration et pouvoir profiter de toute la puissance des RNN LSTM.

5.3 Commentaires

L'investissement dans un RNN LSTM est intéressant si nous possédons un jeu d'entraînement conséquent et assez de ressources pour permettre une optimisation des hyper-paramètres et de la structure du réseau.

Pour des jeux de données plus réduits, les modèles statistiques sont plus légers et donnent de bons résultats. La librairie Prophet est robuste pour des séries temporelles avec tendance et saisonnalité ayant un historique d'un an ou plus.

Pour les séries de petite taille de types stochastiques et localement non-stationnaires, les modèles régressifs donnent de bons résultats pour des prédictions à court terme.

6 Conclusion et élargissement

Nous avons présenté dans ce mémoire la Startup JetPack Data ainsi que ses besoins en termes de prédictions sur des données de type time series économiques. Afin de répondre au besoin, nous avons livré un module packagé sous [pypi](#) dont le code est disponible en open source sur la plateforme [github](#). L'intégration du module dans la plateforme JetPack Data est en cours.

Plusieurs pistes d'évolution existent pour enrichir ce module de prédiction. Nous avons cité précédemment l'intégration de ARIMA mais il faudrait étudier en détail la pertinence de ce choix vu que prophet répond au besoin efficacement.

Aussi, il serait judicieux d'ouvrir certains paramètres de prophet aux utilisateurs afin de pouvoir itérer et améliorer les modèles de prédiction d'une manière continue sans nuire à l'ergonomie de la plateforme.

Si le développement de la plateforme le permet, certaines données complexes nécessitant une estimation paramétrique volumineuse, peuvent tirer profit d'un RNN LSTM qu'il serait envisageable d'intégrer au pipeline aussi.

Pour élargir notre étude, nous nous sommes penchés sur les méthodes inspirées par la topologie algébrique [13] [14] [15] [16] pour étudier la structure de certaines time series [17]. Ces méthodes sont très prometteuses pour les systèmes dynamiques à variation temporelle.

7 Annexes

Bibliographie

- [1] S. J. T. a. B. Letham, «Forecasting at Scale,» 13 Janvier 2017.
- [2] G. G. R. Jeffrey Heer, «Animated Transitions in Statistical Data Graphics».
- [3] J. Brownlee. [En ligne]. Available: <https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/>.
- [4] B. G. a. G. Bandyopadhyay, «Gold Price Forecasting Using ARIMA Model».
- [5] wikipedia. [En ligne]. Available: https://en.wikipedia.org/wiki/Dickey–Fuller_test.
- [6] D. DESBOIS, «Une introduction à la méthodologie de Box et Jenkins : l'utilisation de modèles ARIMA avec SPSS».
- [7] D. Delignières, «Séries temporelles – Modèles ARIMA.».
- [8] A. Geron, Hands-On Machine Learning with Scikit-Learn & TensorFlow, O'Reilly.
- [9] C. Olah, «Understanding-LSTMs,» [En ligne]. Available: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>.
- [10] F. Gers, «Long Short-Term Memory in Recurrent Neural Networks».
- [11] N. Kourentzes, «can-neural-networks-predict-trended-time-series,» [En ligne]. Available: <http://kourentzes.com/forecasting/2016/12/28/can-neural-networks-predict-trended-time-series/>.
- [12] J. L. B. Diederik P. Kingma, «ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION».
- [13] C. M. Pereira et Rodrigo F. de Mello, «Persistent homology for time series and spatial data clustering Persistent homology for time series and spatial data clustering,» *Expert Systems with Applications*, p. 2, 01 septembre 2015.
- [14] «Introduction to the R package TDA».
- [15] L. M. Seversky, Shelby Davis et Matthew Berger, «On Time-series Topological Data Analysis: New Data and Opportunities». *Computer Vison Foundation*.
- [16] E. Munch, «Applications of Persistent Homology to Time Varying Systems».
- [17] Y. K. Marian Gidea, «Topological Data Analysis of Financial Time Series: Landscapes of Crashes,» *arXiv*, 2017 Avril 2017.
- [18] S. Price, «Mining the past to determine the future: Comments,» *International Journal of Forecasting*, 2009.
- [19] G. Bontempi, «Machine Learning Strategies for Time Series Prediction,» 2013.

- [20] P. Gómez-Gil, «On the use of ‘Long-Short Term Memory’ neural networks for time series prediction,» 2014.
- [21] B. R. O. & A. Trabelsi, «Optimizing Multilayer Perceptrons for Time Series Prediction,» 13 Fevrier 2003.