CNAM

Projet RCP209 : Prédiction des mouvements journaliers des actions américaines

Jérôme Petit

8 avril 2019

1 Introduction

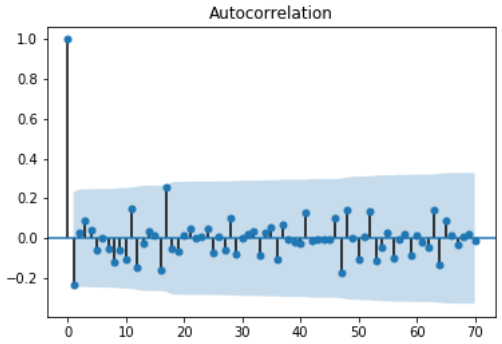
Dans le cadre du cours RCP209 du master TRIED, nous avons pu voir différentes méthodes pour l'analyse et la prédiction de données. L'objectif de ce projet est d'utiliser les différentes notions abordées au cours de ce module dans un cas réel. J'ai décidé de travailler sur le jeu de données issue du challenge CFM : prédiction des mouvements journalier des actions US. Le but est de prédire le signe du prix relatif du stock sur la période 15h30-16H connaissant l'évolution par tranche de 5min sur la période 9h-15H30. Par mouvement, l'organisateur du challenge entend le mouvement relatif à un certain benchmark qui n'est pas communiqué. Ainsi les donnée ne sont pas des prix mais une différence de ratio, ainsi une valeur nulle signifie que l'action a évolué de la même manière que le benchmark. L'utilisation d'un benchmark est une technique courante dans la gestion de fond. Le benchmark peut être le CAC40 ou le SP500 et l'objectif du gérant est de surperformer ce benchmark.
Dans ce rapport, je présenterai dans un premier l'analyse statisque faite sur les données. Puis je présenterai les modèles étudiées pour la prédiction des données : la régression logistique, les SVM linéaire et les réseaux de neurones.

2 Analyse de données

2.1 Volumes et données manquantes

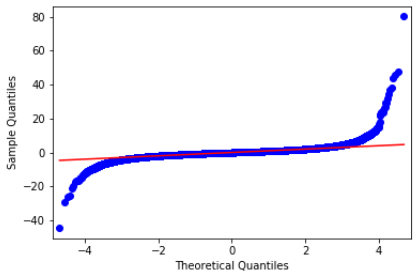
Pour ce projet nous disposons de 52.918.217 données d'entrainement. Parmi ces données d'entrainement il y a 347646 données manquantes soit 0.65% de l'échantillon dis-

ponible. Il y en a tout 745.327 séries temporelles de longueur 71. Le nombre 71 correspond au nombre de tranche de 5 mins compris en 9h et 15h30. Ces séries temporelles sont issues de 680 actions pour 1511 dates différentes. Nous n'avons pas d'information sur les dates et les titres. Les dates s'étalent sur plusieurs années et donc permettent d'avoir des séries temporelles pour différentes situations économiques.
L'absence de données pour un titre à un instant donnée peut être du à une absence de quotation du titre, un problème avec le fournisseur de données ou un problème en interne sur le stockage des données. Pour des titres très liquides, l'absence de quotation est peu probable et il s'agit généralement d'un problème avec la récupération du cours de l'action. Il faut pouvoir déterminer la quotation manquante car cette quotation existe forcement. Il existe différentes approches : remplacement par 0, replacement par la moyenne, interpolation. J'ai utilisé l'interpolation afin de lisser les données et d'éviter les sauts dans les cours des titres. Il existe différents types d'interpolation : linéaire, exponentiel .... Étant donnée qu'il s'agit de données ayant une fréquence de 5 min, les différentes méthodes seront équivalentes du fait du peu d'écart entre les données. C'est pourquoi j'ai choisi d'utiliser la méthode d'interpolation linéaire. Les données inconnues à 9H30 et 16h30 ne peuvent pas être déterminées par l'interpolation. Pour ces données j'ai utilisé une extrapolation constante, c'est à dire que l'on remplace les données inconnues par la dernière donnée connue. Contrairement au méthode remplacement par la moyenne ou 0, cela n'introduit pas de discontinuité dans la série temporelle.
Afin de prédire le cours de l'action en fin de journée, on peut se demander si des modèles stationnaires tels que les modèles ARMA, AR ou MA peuvent être utilisé. Pour cela on peut regarder autocorrelogramme afin de regarder s'il y a de l'auto corrélation pour un certain lag. Les cours journalier ne présentent pas de auto-corrélation, comme le montre l'acf obtenu pour l'ID : 146875523

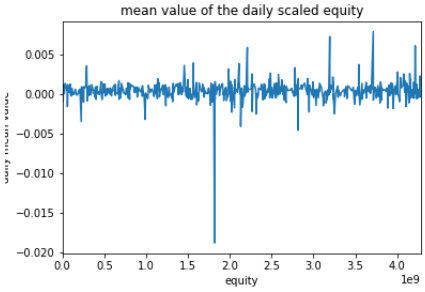


2.2 Analyse globale

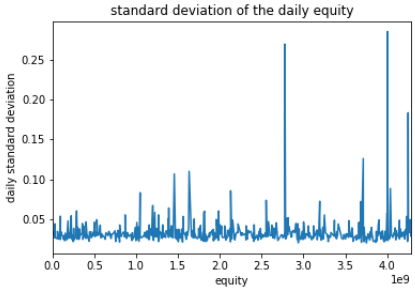
Cela veut dire que les modèles ARMA ne semblent pas appropriés. On pourrait étudier les différence des séries temporelles afin de vérifier si l'on peut l'approcher par des modèle ARIMA, mais je n'ai pas réaliser cette étude afin de me concentrer uniquement sur les méthodes du cours. J'ai pu observé que les données d'apprentissage : données d'entrainement et résultat ont une distribution similaire et ne suivent pas une distribution normale. J'ai observé que les prix de fin de journée ont une kurtosis supérieure à la loi normale (qqplot réalisé avec la librairie sciplot) :



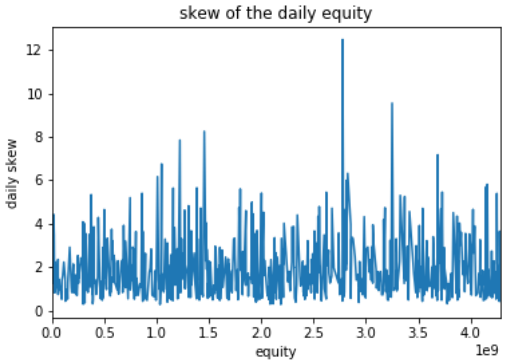
Les données d'entrainement et de fin de journée ont une distribution centrée en zero avec des outliers. En effet pour une tranche horaire donnée l'amplitude des prix est très variable. Par exemple à 9H30 le minimum est de -153.162990, le maximum est de 368.253598 pour un prix moyen de -0.006423, alors qu'à 9H35 le prix moyen est proche -0.003866 mais le maximum est de 135.032397 et le minimum est de -95.909324. On remarque également que les quantiles 25% et 75% sont de l'ordre de ± 1. Comme la distribution n'est pas normale et qu'il y a des outliers j'ai choisi de recentrer les données en utilisant RobustScaler de la librairie sklearn. Cette méthode utilise les quantiles pour recentrer les données. J'ai utilisé les quantiles 1% et 99%. Contrairement à la méthode minMax cette méthode ne rend pas la distribution multi-modale. Après avoir recentré les données avec RobustScaler et en regroupant les données par titre, j'ai pu déterminer l'évolution du prix relatif journalier pour chaque titre. On observe que la série temporelle oscille autour de zero avec peu d'outliers.



On observe également que la volatilité du prix est relativement avec des pics sur certains titres :



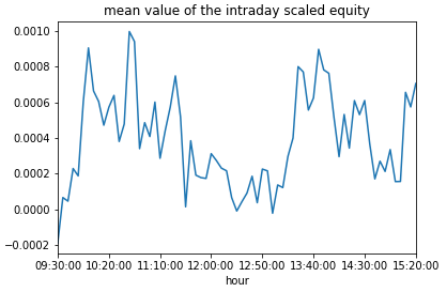
Finalement on observe une asymétrie positive des titres, ce qui confirme que la distribution n'est pas normale :



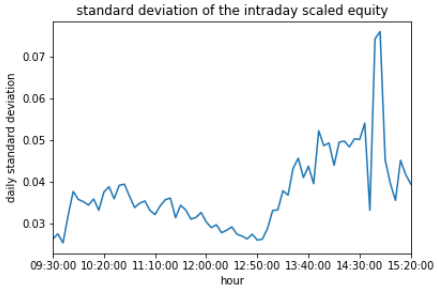
Cela montre que pour tous les titres le prix relatif moyen oscille autour de 0 et que l'amplitude de l'oscillation dépend des titres, ci dessous se trouve les informations concernant la moyenne et l'écart type des titres :

	moyenne des prix journalier	écart type des prix journalier	médiane des prix journalier	asymétrie
moyenne	0.000394	0.032972	-5.840450e	1.861731
écart type	0.001240	0.017569	9.270646e-4	1.401303
min	-0.018845	0.020032	-0.028685	0.271616
max	0.007914	0.285208	5.686465e-03	12.480709

J'ai réalisé la même analyse pour les données intraday. Dans ce cas les données de tous les titres sont regroupées par heure.

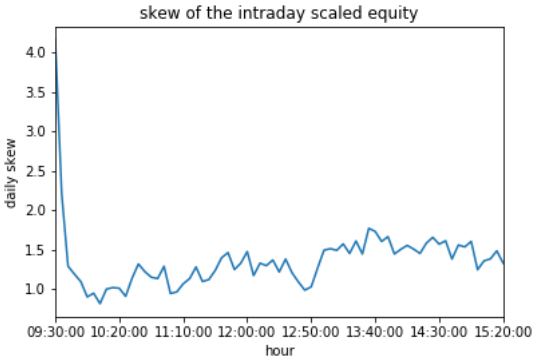


On observe un saut important à l'ouverture, puis une oscillation autour de la moyenne. Par contre la volatilité est surtout observée en fin de journée avant la clôture ce qui est assez courant sur les marchés. Le fait de recentrer les données à diminuer la volatilité à l'ouverture des marchés



Les prix durant la journée présentent également une asymétrie positive. Ce qui confirme

que les cours ne suivent pas une distribution normale durant la journée :



ci dessous se trouve les informations sur les cours intraday :

	moyenne des prix intraday	écart type des prix intraday	médiane des prix intraday	asymétrie intraday
moyenne	0.000394	0.037428	-0.000011	1.367154
écart type	0.000265	0.009753	0.000093	0.419735
min	-0.000186	0.025223	-0.000216	0.816181
max	0.000994	0.075983	0.000229	4.165111

3 LES MODÈLES ÉTUDIÉES

Dans ce challenge, l'organisateur fournit la métrique qui sera utilisée pour classer les candidats :

```
def CFM_metrics(y_true, y_pred):
""" Return the metrics used for the CFM Data Challenge 2019.
    This metrics is simply the accuracy of the prediction of positive  returns
    (see below for details), but based on true *returns* (y_true) and
    probabilities (of returns being positive).

    Args
        y_true: Pandas Dataframe
            target returns. Positive returns will give True, and negative ones Fals

        y_pred: Pandas Dataframe
            predicted probability of positive returns.

    """
    return ((y_true.values > 0) == (y_pred.values > 0.5)).mean()
```

Cette métrique mesure la moyenne des valeurs correctement prédites. Il s'agit d'un problème de classification à deux états : le stock sur (resp. sous) performe le benchmark (*end of day return>0*) resp. (*end of day return<=0*). J'ai comparé alors plusieurs modèles de classification vu dans le cours :

i) régression logistique

ii) SVM linéaire
iii) réseaux de neurones

3.1 Régression logistique

Soit Y une variable à deux états 0 et 1. LA variable Y est la variable à prédire et X les variables explicatives de Y. La régression logistique binaire consiste à déterminer la probabilité $P(Y = 1|X)$ par une méthode de maximum de vraisemblance sur la fonction :

$$\Pi_\omega P\left(Y(\omega) = 1|X\right)^{Y(\omega)} \left(1 - P\left(Y(\omega) = 1|X\right)\right)^{1-Y(\omega)}$$

De plus la régression logistique utilise la fonction sigmoïde pour évaluer les probabilité. La probabilité s'écrit donc :

$$P(Y|X) = \frac{1}{1 + \exp(-w^T X)}$$

La fonction sigmoïde va donc éliminer les outlier. L'inconvénient avec cette méthode est que l'on cherche à prédire Y à l'aide de combinaison linaire de la variable explicative X puis on applique la sigmoïde sur cette combinaison linéaire. Cette méthode est donc intéressante dans les cas de dépendance linéaire. On peut compléter cette méthode par des termes de régularisation (Ridge ou lasso) afin de contrôle la norme de w. Avec la régularisation, la régression logistique devient un problème d'optimisation pour la norme l2 (régularisation Ridge)

$$\min_{w,c} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \log(\exp(-y_i(X_i^T w + c)) + 1)$$

et pour la norme l1 (régularisation lasso)

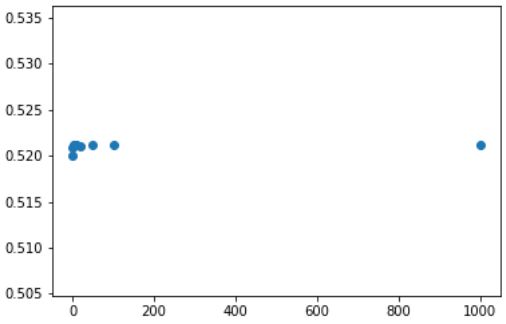
$$\min_{w,c} \|w\|_1 + C \sum_{i=1}^{n} \log(\exp(-y_i(X_i^T w + c)) + 1)$$

3.1.1 Méthodologie

Afin de trouver le meilleur modèle possible parmi les régressions logistique, j'ai décomposé mes données d'apprentissage en deux jeux de données : apprentissage et test via la function train test split de sklearn (70% apprentissage et 30% tests). De plus j'ai converti les résultats des données d'apprentissage en donnée binaire : 0 si la valeur est négative et 1 sinon. J'ai utilisé l'optimiser : SAG (Stochastic Average Gradient Descent) qui est plus performant que les méthodes lbfgs pour des grands jeux de données. SAG utilise la norme l_2. Afin de déterminer le coefficient de régularisation C, j'ai réalisé d'une validation croisée : 5-Fold. La librairie sklearn fournit un modèle LogisticRegressionCV qui réalise dans un premier temps la détermination du paramètre C par validation croisée, puis calibre le modèle sur la donnée d'apprentissage. Je n'ai pas fournit de liste de valeur du paramètre, dans ce cas LogisticRegressionCV choisi aléatoirement un nombre de valeur (donné dans le constructeur) de C dans l'intervalle $[10^{-4}, 10^4]$. J'ai utilisé la métrique score défini dans la classe LogisticRegressionCV. Cette métrique est la même que celle fournit par l'organisateur du challenge. Voici les résultats obtenus :
On remarque que les valeurs obtenues pour les différents facteurs de régularisation

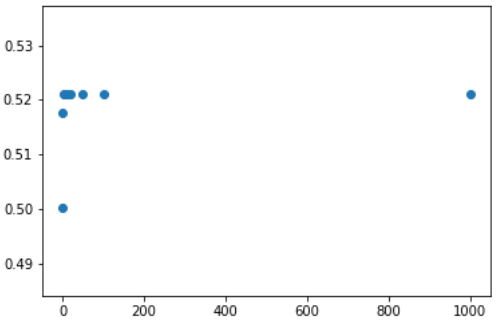
nombre de paramètre	meilleur C	score sur les données d'entrainement	score sur les données de test	valeur testée
20	100	0.5207311089303238	0.520225940187568	
10	21.5443469	0.5211240339793916	0.5195103734810979	
5	100	0.5207311089303238	0.520225940187568	

sont très proches. On en déduit que le facteur de régularisation C dans le cas où la régularisation est l2 à peu d'impact. Afin de confirmer ce comportement j'ai testé le modèle de régression logistique avec le solver SAG (norme l_2) pour les valeurs suivantes : C=[0.001, .01, .1,1,5,10,20,100,1000]



C	score training	score test
0.001	0.5200123435966634	0.5188574188614439
0.01	0.5209611138370952	0.5192912311772414
.1	0.521125950686948	0.5194745951457743
1	0.5211317008096172	0.5194656505619435
5	0.5211336175171737	0.5194477613942817
10	0.5211202005642787	0.5194656505619435
20	0.521125950686948	0.519461178270028
100	0.5211278673945045	0.5194656505619435
1000	0.521125950686948	0.5194522336861972

J'ai effectué la même étude pour le méthode SAGA et en utilisant la norme l_1 qui permet de supporter des poids *sparse*. J'obtiens les résultats suivant :



C	score training	score test
0.001	0.5002165879538764	0.4994923948675978
0.01	0.5175455409715407	0.5159414845325784
.1	0.5211547013002944	0.5194253999347045
1	0.5211853686211972	0.5194790674376898
5	0.5211547013002944	0.5194522336861972
10	0.5211221172718351	0.5194656505619435
20	0.5211317008096172	0.5194432891023663
100	0.5211240339793916	0.5194656505619435
1000	0.5211240339793916	0.5194656505619435

Le fait que le coefficient de régularisation ait peu d'impact dans les résultats doit être du au fait que les prix sont proche de zéros en moyenne ce qui rend le terme de régularisation constant. J'ai testé différente valeur de C sur le jeu de donnée de test. Le meilleur résultat obtenu fut pour C=21.5443469 et un solver SAG avec une régularisation l2 (score de 0.523 qui supérieur au benchmark du challenge). Le benchmark du challenge a été obtenu avec un modèle LightGBM boosted tree.

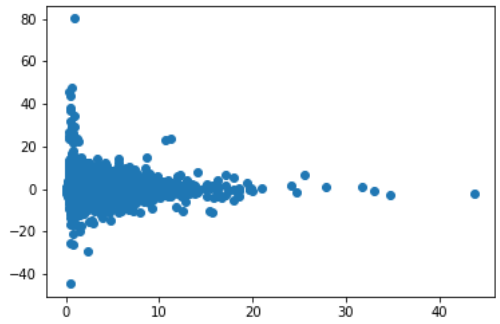
3.2 SVM linéaire

Afin de trouver le meilleur modèle possible parmi les SVM, j'ai décomposé mes données d'apprentissage en deux jeux de données : apprentissage et test via la function train test split de sklearn (70% apprentissage et 30% tests). De plus j'ai converti les résultats des données d'apprentissage en donnée binaire : 0 si la valeur est négative et 1 sinon.
La classification par SVM linéaire réside dans l'estimation d'un hyperplan qui permet de séparer les données en deux catégories. Pour cette approche j'ai utilisé la librairie linearSVC de sklearn. Cette librairie est optimisée pour les données de grande dimension. Pour cette étude j'ai étudié le comportement de modèle pour des paramètres de régularisation. Nous avons vu que pour la régression logistique le paramètre n'influe pas sur les résultats. Dans le cas de SVM linéaire, l'impact de C est le suivant : On re-

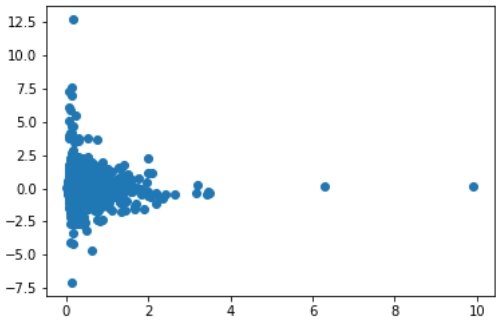
C	score training	score test
0.001	0.5184578230119027	0.5181776304902974
0.01	0.5210549517949897	0.518052406316665
.1	0.5205950226938175	0.5187769176069661
1	0.5205988561089303	0.5187411392716426
5	0.5208250276005888	0.5184728017567163
10	0.521058865922473	0.5193985661832119
20	0.5212773705839058	0.5195864024436603
100	0.5215322926889107	0.5198368507909248
1000	0.521072282875368	0.5200783545543585

trouve des résultats similaires à ceux de la régression logistique sur les données d'apprentissage. Sur le données de test les résultats sont moins bon. Cette observation se confirme sur les données de test du challenge ou le score obtenu par des méthodes linéaires SVC est inférieur à celui de la régression logistique. On est dans un situation de

sur-apprentissage et de la dispersion des points ne permet pas d'avoir une séparation linéaire des points. Ci dessous le graphique des points (moyenne, valeur fin de journée)



Ci dessous le graphique des points (écart type, valeur fin de journée)



3.3 Réseaux de neurones

Pour l'approche par réseau de neurones, j'ai étudié les réseaux de neurones avec ou sans couche cachée et l'impacte des fonction d'activation sur les performances. Les résultats sur l'ensemble d'entrainement était légèrement meilleur que ceux observé pour la régression logistique mais il y avait une forte baisse lorsque l'on passe sur les données de test. Ceci est un sur apprentissage du modèle. Pour éviter ce sur apprentissage j'ai utilisé des Dropout pour chaque couche caché. Le Dropout évite sur le sur apprentissage car les neurones utilisés pour l'apprentissage ne seront pas les même (probabilité de désactivation est la probabilité du dropout). L'utilisation du Dropout à permis d'améliorer l'erreur de généralisation du modèle. Pour l'initialisation du modèle, j'ai choisi l'initialisation glorot normal fourni par la librairie Keras (initialisation de Xavier). Cette initialisation a permis d'améliorer le score sur les données d'apprentissage. Pour déterminer le nombre de couche intermédiaire et de méthode d'activation, j'ai opéré de la manière suivante :

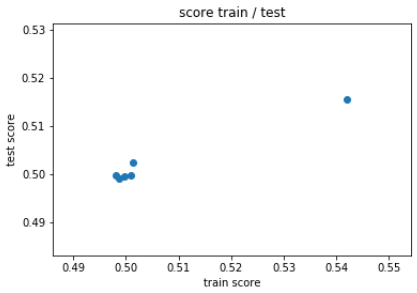
— prépration des données
— détermination du nombre de couche cachée
— détermination des fonctions d'activation

Pour la prépartion des données, j'ai décomposé mes données d'apprentissage en deux jeux de données : apprentissage et test via la function train test split de sklearn (70% apprentissage et 30% tests). De plus j'ai converti les résultats des données d'apprentissage en donnée binaire : 0 si la valeur est négative et 1 sinon.
J'ai décidé de rajouter des couches cachées car sinon on ne profite pas de la capacité des réseaux de neurones à rendre non linéaire le problème. J'ai pu observer qu'au delà de 2 couches cachées, il y avait une dégradation de l'erreur de généralisation. Cela est du au sur apprentissage du réseau. Je me suis limité à l'étude de réseau à 3 couches. J'ai obtenu le meilleur en utilisant :

i) nombre de couches 3
ii) fonctions d'activation : Relu couche 1, tanh couche 2 et softmax couche 3
iii) initialisation : glorot normal
iv) optimizer : adam

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
fc1 (Dense)                  (None, 1000)              72000
_________________________________________________________________
dropout_29 (Dropout)         (None, 1000)              0
_________________________________________________________________
activation_29 (Activation)   (None, 1000)              0
_________________________________________________________________
fc2 (Dense)                  (None, 100)               100100
_________________________________________________________________
dropout_30 (Dropout)         (None, 100)               0
_________________________________________________________________
activation_30 (Activation)   (None, 100)               0
_________________________________________________________________
fc3 (Dense)                  (None, 2)                 202
=================================================================
Total params: 172,302
Trainable params: 172,302
Non-trainable params: 0
_________________________________________________________________
None
```

Sur les données d'apprentissage les réseaux de neurones on fournit de meilleur résultat que la régression logistique mais sur les données de test le réseau a obtenu un score inférieur au benchmark. En diminuant le nombre de couche intérieur alors l'erreur de généralisation diminue mais l'erreur d'apprentissage augmente. Pour les fonctions d'activation le choix de tanh a donné de meilleur résultat que pour la sigmoïde. Cela provient du fait que la sigmoïde se ramène à [0,1] alors que tanh se ramène à [-1,1] et tient compte des valeurs négatives. Cette approche reste moins bonne que l'approche par régression logistique car le sur apprentissage de cette méthode est plus important. J'ai utilisé l'activation relu pour la première pour tenir compte de l'asymétrie positive des données. J'ai réalisé l'étude suivante afin de déterminer les nombre de neurones dans les couches de layer :



Ci dessous se trouve les valeurs obtenues pour le test du nombre de neurones
Afin d'être plus complet une étude sur un choix de valeur plus grand aurait du être fait

neurones layer 1	neurone layer 2	nb de parametre	score train	score test
1000	100	172,302	0.49815996074582924	0.4997428432149956
2000	100	344,302	0.49875222338076547	0.49904069338600454
3000	100	516,302	0.499654992639843	0.4995550069548138
5000	10	410,032	0.5013091112610403	0.5023725508630058
100	10	8,232	0.50087785206084	0.4998143998859092
100	100	410,032	0.5420966480618253	0.5154763661737727

ainsi qu'une optimisation du learning rate.
Afin de compléter cette étude sur les réseaux de neurones, on peut mentionner les réseaux récurrents : LSTM et GRU que je n'ai pas étudié de manière approfondi au cours de ce projet mais qui sont souvent utilisés dans la prédiction de série temporelle. Si des réseaux pré entrainer existe pour la détection de tendance exist j'aurai pu également les utiliser

4 Conclusion

Au cours de se projet, j'ai pu mettre en pratique les techniques étudiées au cours du module RCP209. Ce projet consiste classifier des séries temporelles. Il s'agit de classification binaire et la métrique fournit par l'organisateur du challenge mesure la précision de la classification en utilisation la moyenne des bons résultats. C'est pour cela que je me suis orienté sur les techniques de classification : régression logistique , SVM linéaire, réseaux de neurone. L'étude des séries temporelles m'a permis de mettre en avant l'asymétrie positive des données et le fait qu'elle soit centrée. Grâce à cela j'ai pu utiliser un recentrais des données plus adaptés. Les résultats obtenus sont intéressant pour la régression logistique car il domine le benchmark. Mais ce ne fut pas le cas pour les réseaux de neurones. L'utilisation de DropOut a pu diminuer l'erreur de généralisation mais le résultat sur les données de test reste inférieur au benchmark.
Une amélioration possible serait de tenir compte du temps et de chercher à exhiber des dépendances temporelle dans les données. Cela permettrait d'utiliser des réseaux récurrent du type LSTM ou GRU.
Une autre piste d'amélioration consiste à utiliser des réseaux pré entrainer et à faire du fine tuning sur ce modèle pré entrainé.