IMT Nord Europe
École Mines-Télécom
IMT-Université de Lille

Rapport UV Projet

Développement d'un module d'optimisation de portefeuille et de prédiction de la rentabilité des actions en bourses

Élèves : Romain MRAD - Yoan HAKOUN

Tuteurs : Radhouane KAMMOUN - Christelle GARNIER - Vincent ITIER

Remerciements

Avant de réaliser le développement de ce rapport, il est nécessaire de remercier les personnes qui nous ont permis de réaliser ce projet et qui en ont fait une belle expérience.

Nous tenons donc d'abord à remercier madame Christelle Garnier et monsieur Vincent Itier pour leur suivi et leur aide quant-à la partie du projet en lien avec la Data Science.

Nous tenons ensuite à remercier monsieur Radhouane Kammoun pour son suivi et son aide quant-à la partie du projet en lien avec la finance de marché.

Enfin, il se doit de remercier l'IMT Nord Europe pour nous avoir permis de réaliser cet UV Projet en 2e année de cycle ingénieur, dans le cadre de la cinquième période d'études.

Table des matières

Glossaire

Portfolio : “portefeuille” en anglais.

Bullish : traduit une tendance haussière.

Bearish : traduit une tendance baissière.

Diagramme UML : Diagramme “Unified Modelling Language” est un diagramme qui permet de visualiser la structure d’une classe, ou la séquence d’un programme.

LSTM : “Long Short-Term Memory” (cellule de longue mémoire à court terme) est un RNN particulier qui permet de pallier aux problèmes liés à la disparition/explosion du gradient et à celui de la mémoire courte des RNNs.

Cours : le cours boursier est le prix auquel s’échange un titre.

Titre : un titre financier correspond à une partie du capital d'une société (action), ou à une partie de sa dette (obligation).

Cours de clôture ajusté : le cours de clôture ajusté est le cours de clôture après ajustement de tous les fractionnements et distributions de dividendes applicables

Introduction

Dans le cadre de la P5, nous avons décidé de réaliser un UV Projet liant la finance de marché et le machine learning. Au cours de ces 4 semaines, nous avons réussi à développer un module de prédiction du marché et d'optimisation de portefeuille sous Python.

Il s'agit donc, tout au long de ce rapport, de définir le projet, de présenter le travail réalisé, et de mettre en avant les résultats obtenus.

Définition du projet

Le projet consiste à développer un module Python qui permet d'abord d'initialiser un portefeuille par rapport à un capital donné et d'optimiser sa répartition en évaluant un paramètre ou en prédisant les cours des actions en bourse.

Ensuite, il sera nécessaire d'automatiser l'entraînement d'un modèle de prédiction pour chaque action afin de prédire leur cours et l'évolution générale du portefeuille.
Par ailleurs, le module devra donc suggérer un changement dans le portefeuille s'il est prédit à la baisse, et agir sur ce changement en demandant à l'utilisateur.

Enfin, il faudra aussi automatiser la visualisation des données et des modèles utilisés et tester le code.

Présentation du travail effectué

Avant de se lancer dans la programmation, il était nécessaire de définir nos hypothèses et notre stratégie et de planifier le code.

Hypothèses

Le marché considéré étant Euronext Paris, nous avons récupéré toutes les actions qui y étaient échangeables. Cependant, il était nécessaire de filtrer certains titres. Nous avons donc retiré toutes les actions qui étaient :

- Récentes : pas assez de données pour l'entraînement correct d'un réseau de neurones.
- Peu chères : les actions ayant un cours inférieur à 5€ (seuil arbitraire) sont considérées comme vouées à la disparition.

Par ailleurs, nous considérons uniquement le cours de clôture ajusté pour effectuer tous nos calculs. Enfin, nous fixons les frais de transaction à 2€.

Stratégie d'investissement

La stratégie mise en œuvre repose sur un investissement à long terme. En se basant uniquement sur les cours de clôture ajustés, le programme doit être activé quotidiennement après la clôture du marché parisien. Par la suite, des ajustements ne sont suggérés que si le programme prévoit une tendance à la baisse dans le portefeuille.

D'un côté, si le cours de clôture ajusté du jour $N + 1$ (prédit) est inférieur à celui du jour N (connu), il est opportun de vendre cette action à l'ouverture du marché le jour $N + 1$. D'un autre côté, si le cours de clôture ajusté du jour $N + 1$ (prédit) est supérieur à celui du jour N (connu), il est préférable de maintenir cette action dans le portefeuille, car elle est susceptible de prendre de la valeur.

Cependant, en cas d'erreur de prévision (prévision **Bullish** au jour $N + 1$ alors que la tendance est réellement **Bearish**), cela n'a pas d'impact, car la prédiction du jour $N + 2$ est souvent correcte (**Bearish**). On ne vendra pas l'action avant qu'elle ne commence à décliner, mais lors de la vente, on le fera à un cours plus élevé que celui de l'achat, assurant ainsi un bénéfice constant.

Planification du code - Objets et Scripts

Pour établir un plan d'action complet, nous avons eu recours aux diagrammes UML qui permettent de bien définir les tâches à exécuter.
Nous avons donc créé trois classes : **Stock**, **Portfolio** et **Market** (diagrammes UML disponibles dans `docs/class/`).

Class Stock

Cette class permet de définir un objet représentant une action en bourse, de manipuler les données en rapport avec cette action et d'entraîner un modèle prédictif.
Voir Diagramme UML Stock en annexe.

Class Portfolio

Cette class permet de définir un objet représentant un portefeuille contenant des actions (instances d'objet **Stock**), de manipuler le portefeuille, de le visualiser, de prédire son évolution et de communiquer avec la base de données locale du projet.
Voir Diagramme UML Portfolio en annexe.

Class Market

Cette class permet de définir un objet représentant le marché de paris comportant toutes les actions (instances d'objet **Stock**), de manipuler ses données et d'automatiser l'entraînement des modèles prédictifs pour chaque action.
Voir Diagramme UML Market en annexe.

Pour manipuler ces trois objets, nous avons codé plusieurs scripts (dans le répertoire `scripts/`) afin de séparer les exécutions et donner à l'utilisateur la liberté d'action. Ces derniers permettent d'entraîner les modèles prédictifs, initialiser le portefeuille, évaluer sa valeur actuelle, prédire son évolution et modifier son contenu (diagrammes UML disponibles dans `docs/sequence/`)

Script portfolio_initialisation.py

Ce script initialise d'abord une instance des objets **Portfolio** et **Market** et recherche les meilleures N actions dans le marché actuel. Ensuite, il ajoute ces N actions au portefeuille en initialisant des objets **Stock**. Enfin, il résout un problème d'optimisation avec contrainte pour calculer la répartition optimale du portefeuille, visualise la répartition avec un diagramme circulaire et sauvegarde le portefeuille dans la base de données locale.
Voir diagramme UML en annexe.

Script portfolio_evalutaion.py

Ce script permet d'abord de lire le portefeuille sauvegardé, télécharger le cours actuel des actions du portefeuille et calculer la valeur du portefeuille. Ensuite, le programme met à jour la sauvegarde du portefeuille. Enfin, il retrace l'évolution du portefeuille en ajoutant la nouvelle valeur.
Voir diagramme UML en annexe.

Script portfolio_prediction.py

Ce script permet d'abord de lire le portefeuille sauvegardé et de prédire le cours des actions du portefeuille. Ensuite, deux cas se présentent :

- Si toutes les actions sont **Bullish** (prédites à la hausse), le programme s'arrête
- S'il y a N actions du portefeuille qui sont **Bearish** (prédites à la baisse), une prédiction sur toutes les autres actions en bourse est lancée. Le programme récupère ensuite les N actions les plus **Bullish**, les ajoute au portefeuille et relance la fonction d'optimisation qui permet de générer une suggestion de portefeuille optimale avec les actions prédites à la plus grande hausse.

Voir diagramme UML en annexe.

Script portfolio_modification.py

Ce script permet de récupérer la suggestion faite par le script `portfolio_prediction.py` et mettre à jour le portefeuille actuel.

Script train_models.py

Ce script entraîne des réseaux de neurones à N couches **LSTM** en fonction de la configuration et les évalue chacun par rapport à une métrique configurée. Ensuite, le programme sauvegarde le meilleur modèle dans le répertoire `models/sequential/` et génère un graphe des couches du modèle dans `graphs/sequential_models/` (voir exemple en annexe). Enfin, il sauvegarde aussi le scaler entraîné dans `models/scaler/`.
Voir diagramme UML en annexe.

Configuration du modules

Dans le dossier `config/` se trouve trois fichiers de configuration permettant de paramétrer le programme : `portfolio_config.json`, `prediction_config.json` et `market_config.json`

Configuration du portefeuille

Paramètre	Clé JSON	Valeurs
Nombre d'actions dans le portefeuille	`numberOfAssets`	Nombre entier
Capital initial pour l'investissement	`capital`	Nombre décimal
Facteur d'optimisation de portefeuille	`optimisationFactor`	'sharpeRatio', 'profitability', 'risk'
Proportion minimale par action pour l'optimisation	`minimalProportion`	Nombre décimal dans l'interval $]0; 1]$
Choix des actions initiales	`initialisationMethod`	'prediction', 'historical'

Configuration de la prédiction

Paramètre	Clé JSON	Valeurs
Métrique à évaluer lors du test des différents modèles	`predictionMetric`	'r2_score', 'direction', 'mean_squared_error'
Nombre maximal de couches LSTM à tester pour les modèles	`maxLayers`	Nombre entier
Nombre de jours passés à considérer pour la prédiction	`lookback`	Nombre entier

Configuration du marché

Paramètre	Clé JSON	Valeurs
Période à considérer pour télécharger les données historiques d'une action	`period`	'1d', '5d', '1mo', '3mo', '6mo', '1y', '2y', '5y', '10y', 'ytd', 'max'
Taux sans risque	`riskFreeRate`	Nombre décimal dans l'interval $]0; 1]$
Frais de transaction	`tradingFee`	Nombre entier ou décimal
Symbols boursiers de toutes les actions à considérer	`stockTickerSymbols`	Liste de string

Base de données locale

Le répertoire `data/` contient trois fichiers JSON nécessaires au fonctionnement du module.

D'abord, le fichier `current_portfolio.json` (voir annexe) représente le portefeuille actuel. On y retrouve les caractéristiques du portefeuille ainsi que sa répartition sur les différents titres.

Ensuite, le fichier `suggested_portfolio.json` (voir annexe) contient la suggestion générée par lors de la prédiction du portefeuille. Si l'utilisateur décide de mettre en place cette suggestion, le programme copie les contenus de `suggested_portfolio.json` dans `current_portfolio.json`

Enfin, le fichier `portfolio_evolution.json` permet de suivre l'évolution de la valeur du portefeuille en fonction du temps. Lorsque le script portfolio_evaluation.py est lancé, la valeur actuelle du portefeuille est ajoutée à ce fichier, ainsi que la date du lancement.

Construction et entraînement des modèles prédictifs

Nous avons choisi comme modèle de prédiction les réseaux de neurones **LSTM** vu qu'ils sont bien adaptés aux séries temporelles comme les cours d'une action.

Après de nombreuses recherches, nous avons trouvé plusieurs modèles prédictifs sur différents sites internet. Ces derniers sont composés d'une combinaison de couches LSTM et Dense avec des paramètres différents. Nous avons donc décidé de tester plusieurs modèles pour chaque action et sélectionner celui qui a le meilleur score.

D'abord, l'entraînement des modèles s'est fait en considérant le vecteur de données $[N - 49, N]$ (50 dernières valeurs) pour prédire le jour $N + 1$ (lendemain).

Vecteur correspondant au jour N :

$N - 49$	$N - 48$	$N - 47$	$N - 46$	.....	$N - 3$	$N - 2$	$N - 1$	N

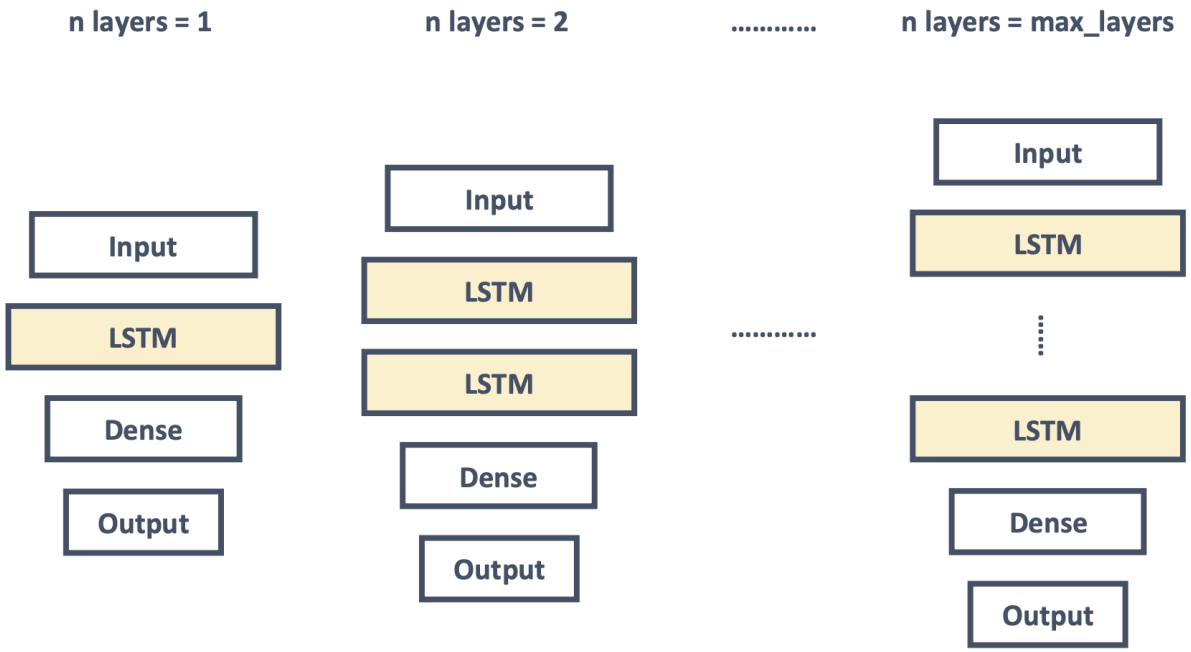
Par ailleurs, les données ont été normalisées dans l'interval $[0, 1]$. En effet, cela améliore la performance des modèles **LSTM**.

Ensuite, nous avons considéré trois méthodes d'évaluation possibles pour la sélection du meilleur modèle :

- Le **score R2** : métrique de régression qui évalue la ressemblance entre l'allure de la courbe de prédictions et celle des données réelles.
- L'erreur quadratique moyenne **MSE** : métrique qui évalue la proximité entre les données prédites et réelles.
- La **direction** de la prédiction : la part des actions correctement prédites **Bullish** et **Bearish**.

Le programme entraîne donc plusieurs modèles séquentiels en ajoutant à chaque itération une couche **LSTM**.

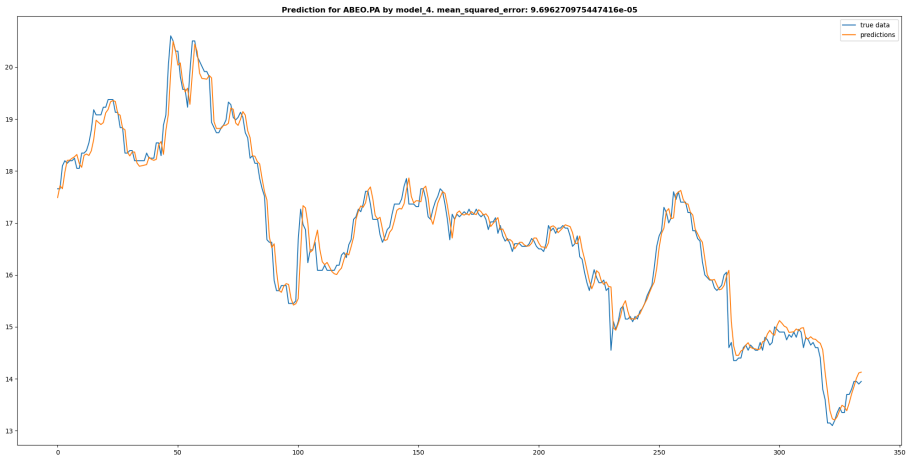
Construction des différents modèles à tester :



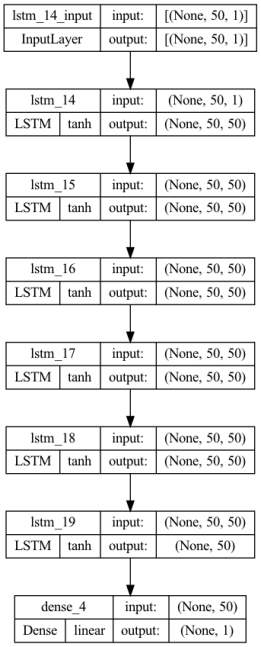
Comme chaque action a un comportement différent, il était nécessaire d'entraîner un réseau de neurones par action, et de le sauvegarder. Cependant, l'entraînement et la sélection du meilleur réseau de neurones pour une action prend environ 30 minutes. Il était donc impossible de réentraîner les réseaux à chaque lancement du programme, comme il était nécessaire de prédire l'entièreté des actions considérées (environ 240 actions).

Enfin, pour chaque réseau entraîné, le programme génère un comparatif des cours prédits et des cours réels test et un graphe représentant le modèle sélectionné. Ces graphes sont disponibles dans les dossiers `graphs/prediction/` et `graphs/sequential_models/`.

Visualisation de la prédiction test



Représentation du modèle



Analyse et critique des résultats

Afin de procéder à une évaluation de la prédiction, il était nécessaire d'analyser les graphes générés par l'entraînement de chaque réseau de neurones..

Exemple de la prédiction test lors de l'entraînement pour Medincell S.A. :

D'abord, en analysant l'entièreté des graphes générés, nous avons remarqué que les allures des courbes de prédictions sont quasi-identiques à celles des courbes de cours réels. Les modèles entraînés sont donc efficaces. Cependant, nous soulignons aussi un "retard" sur la courbe de la prédiction.

Ensuite, pour évaluer le fonctionnement global, il faudra observer le graphique `graphs/portfolio/portfolio_evolution.png` qui trace l'évolution de la valeur du portefeuille.

Enfin, en calculant des métriques standards, nous trouvons des résultats satisfaisants :

- Les **MSE** sur les prédictions sont de l'ordre de 10^{-4}
- Les **scores R2** sur les prédictions sont supérieures à 0.9
- Les **scores sur la direction de la prédiction** sont aux alentours de 0.5

Le score sur la direction est dû au "retard" sur la courbe de prédiction ci-dessus. Cependant, ce dernier est négligeable par rapport à la stratégie explicitée précédemment. En effet, l'erreur de prédiction concernant la **direction** n'impacte pas la stratégie d'investissement à long terme.

Guide d'utilisation

Après avoir explicité le travail effectué, il est maintenant nécessaire d'expliquer comment utiliser le module développé.

Téléchargement

Pour télécharger tout le module, il faudra lancer la commande suivante dans un terminal :

```
git clone https://github.com/romainmrad/UVProjet.git
```

Ensuite, il faudra installer toutes les librairies Python nécessaires avec les commandes suivantes :

En utilisant pip	En utilisant anaconda

`pip3 install -r requirements.txt`	`conda create --name UVProjet`
	`conda activate UVProjet`
	`conda install --file requirements.txt`

Fonctionnement

Jour 0

Étapes à réaliser :

- Configurer le portefeuille, la prédiction et le marché dans les fichiers de configuration
- Executer le script `portfolio_initialisation.py`
- Executer le script `portfolio_prediction.py`

Jour N

Étapes à réaliser si prédiction en $N - 1$ est **Bullish** :

- Executer le script `portfolio_evaluation.py`
- Executer le script `portfolio_prediction.py`

Étapes à réaliser si prédiction en $N - 1$ est **Bearish** :

- Executer le script `portfolio_modification.py`
- Executer le script `portfolio_prediction.py`

Conclusion

En somme, tout au long de ces quatre semaines de projet, nous avons développé un module d'optimisation de portefeuille et de prédiction des actions en bourse. Nous avons donc automatisé une optimisation avec contrainte, un entraînement de réseaux de neurones et une génération de graphiques. Ces dernières fonctionnalités permettent d'initialiser un portefeuille de manière optimale, de prédire son évolution, de suggérer une modification et de visualiser les données, les prédictions et les modèles.

Annexes

Diagramme UML Stock

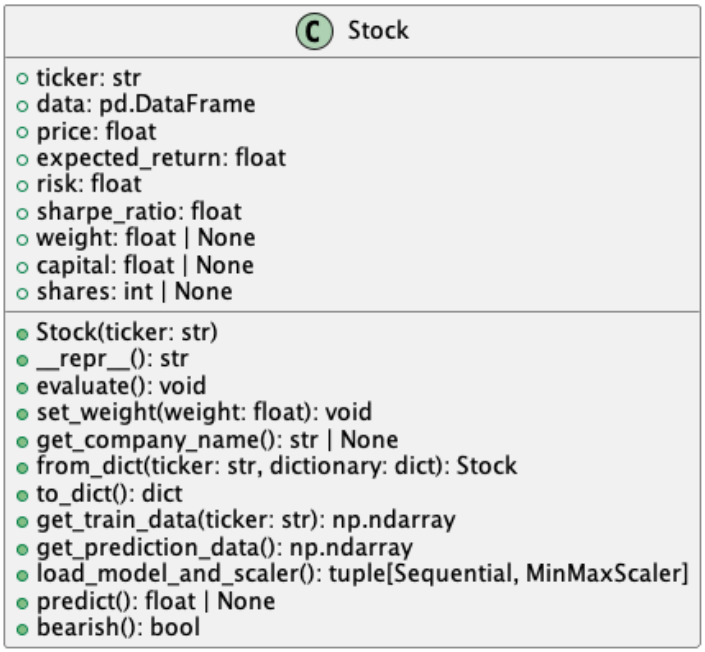


Diagramme UML Portfolio

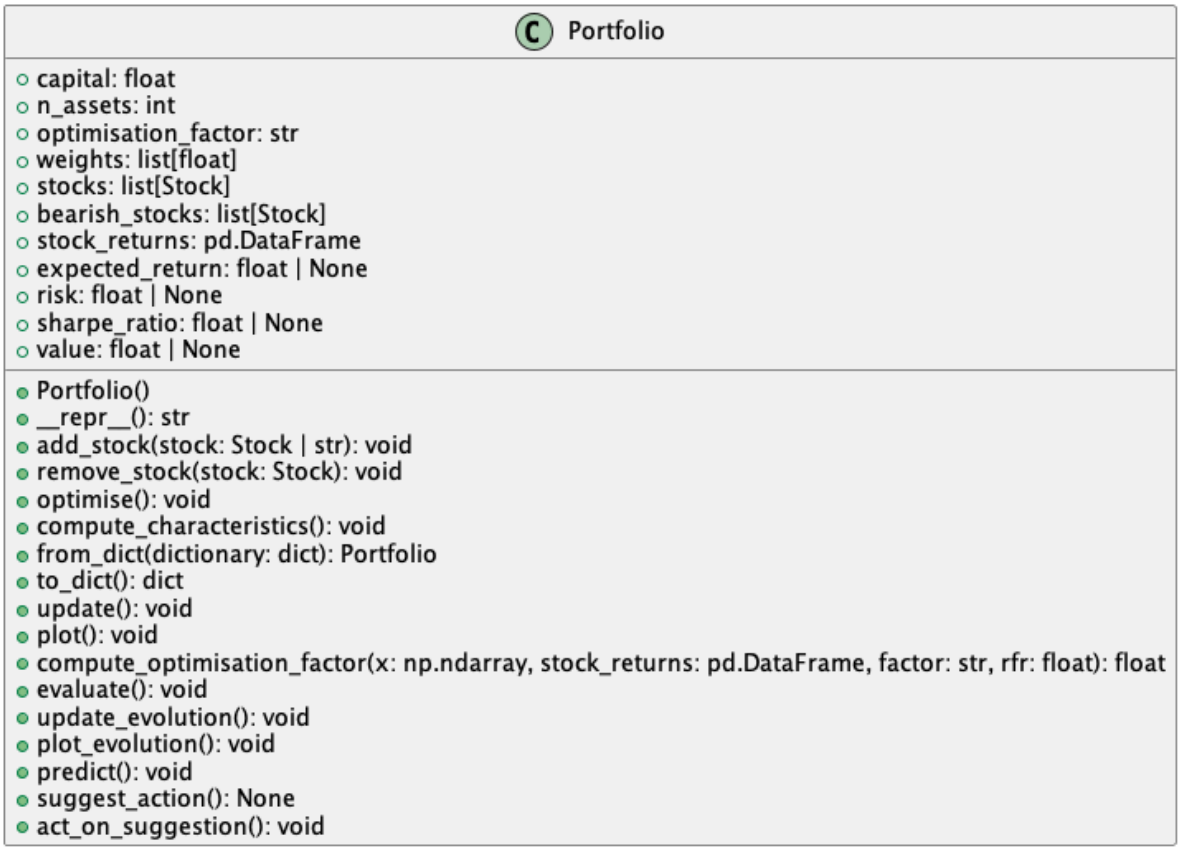


Diagramme UML Market

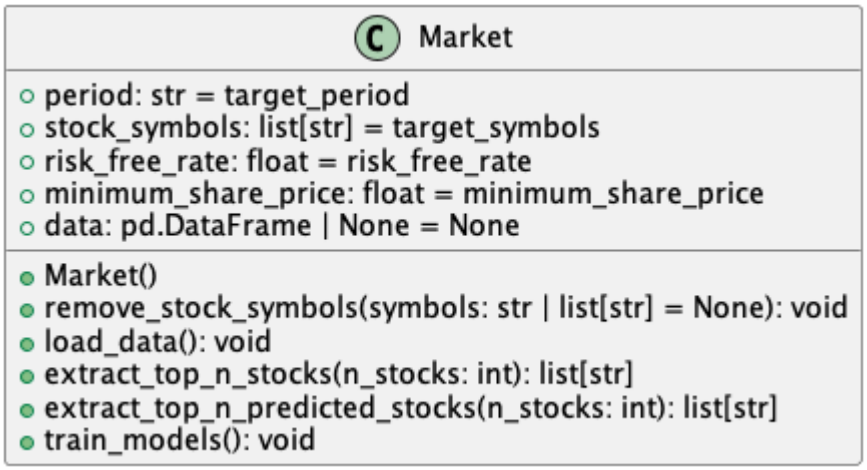


Diagramme UML portfolio_initialisation.py

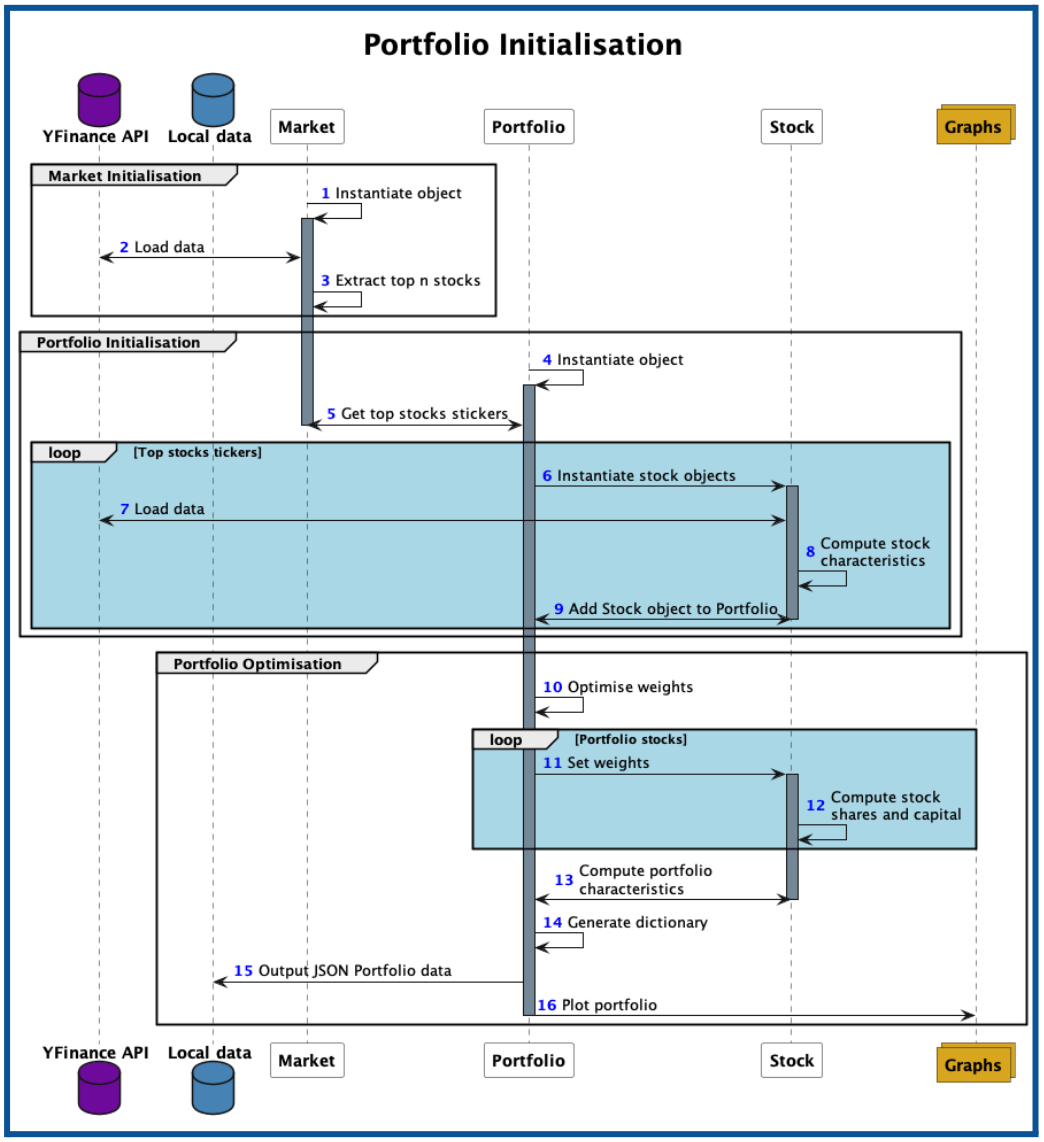


Diagramme UML portfolio_evaluation.py

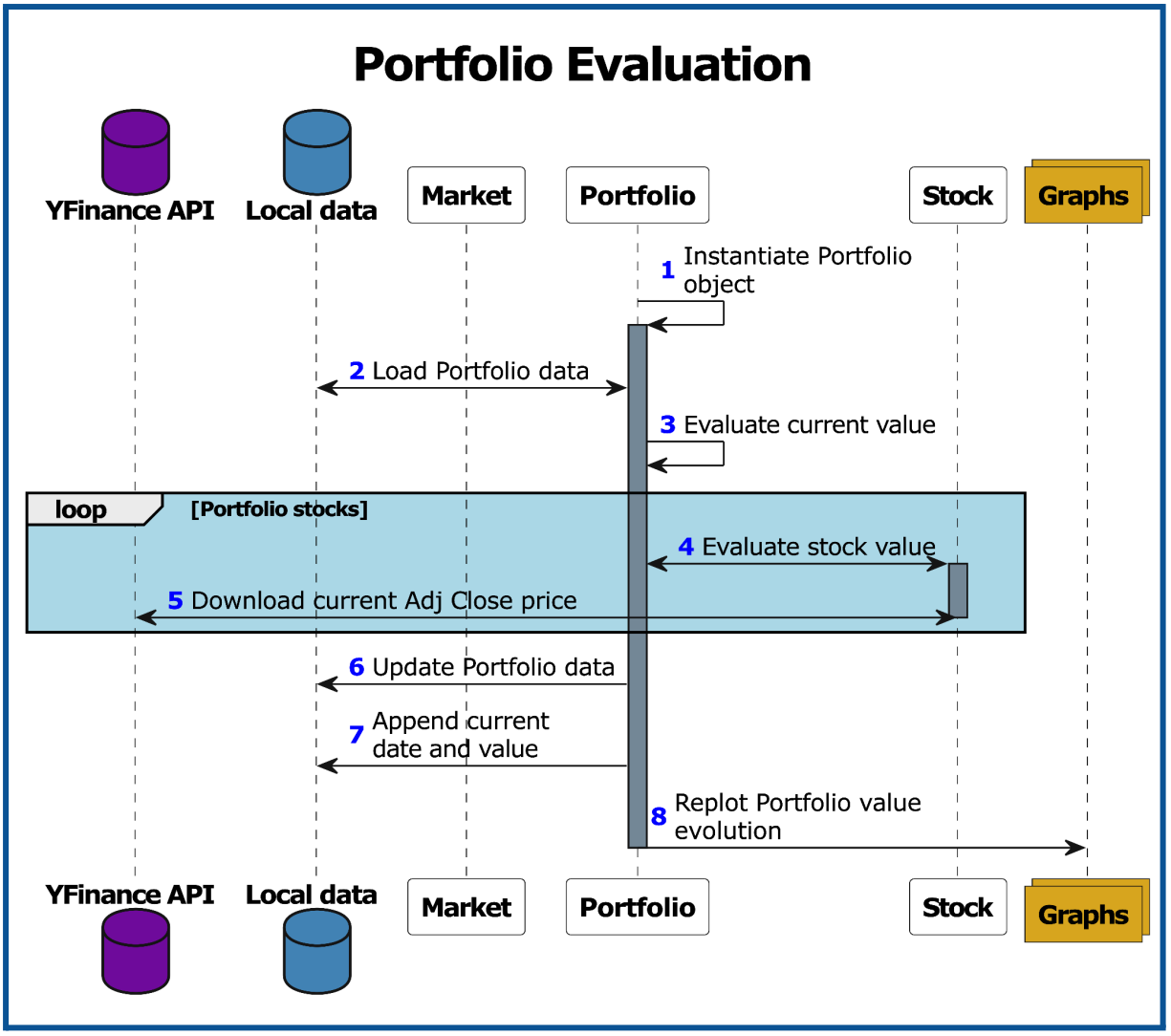


Diagramme UML portfolio_prediction.py

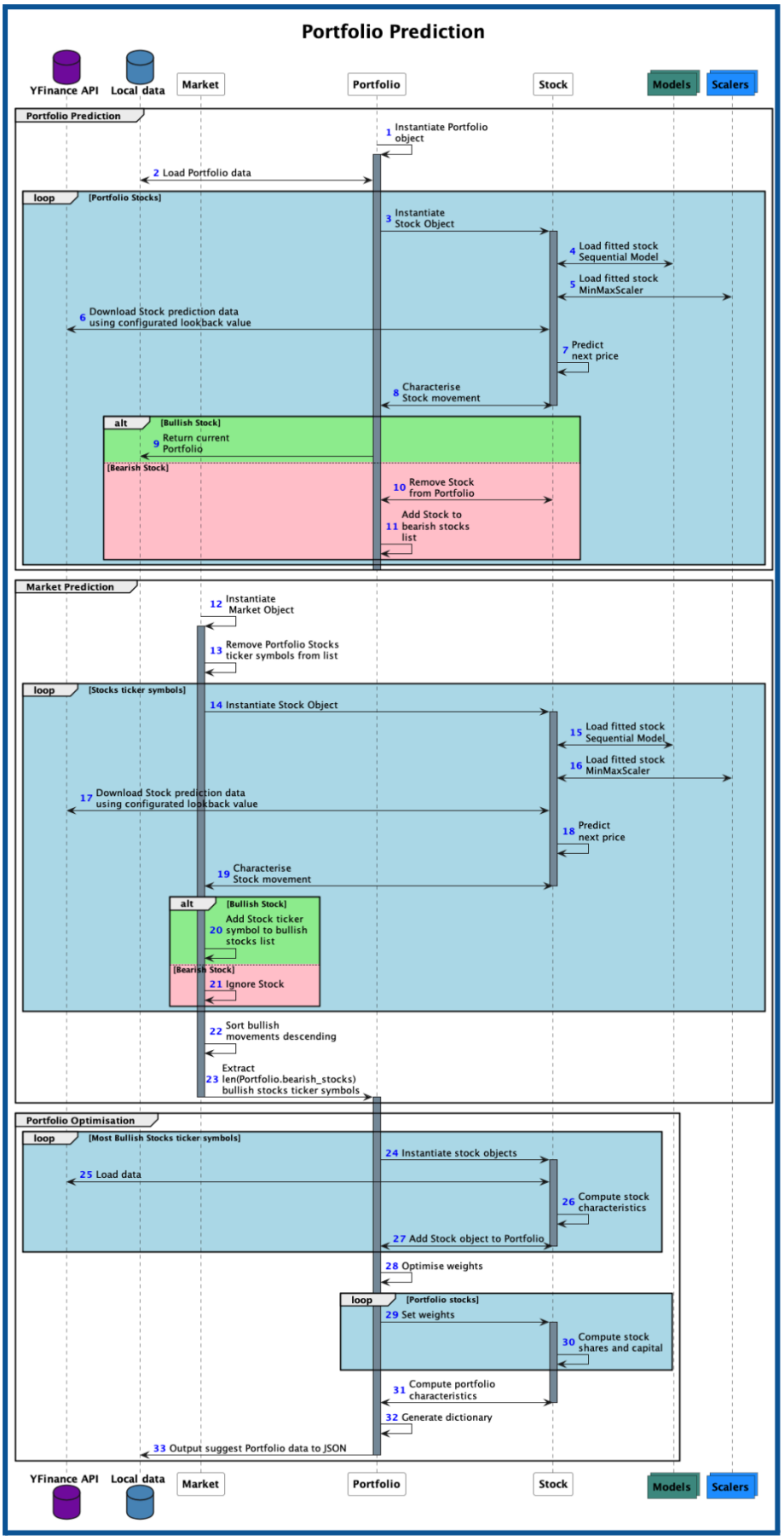
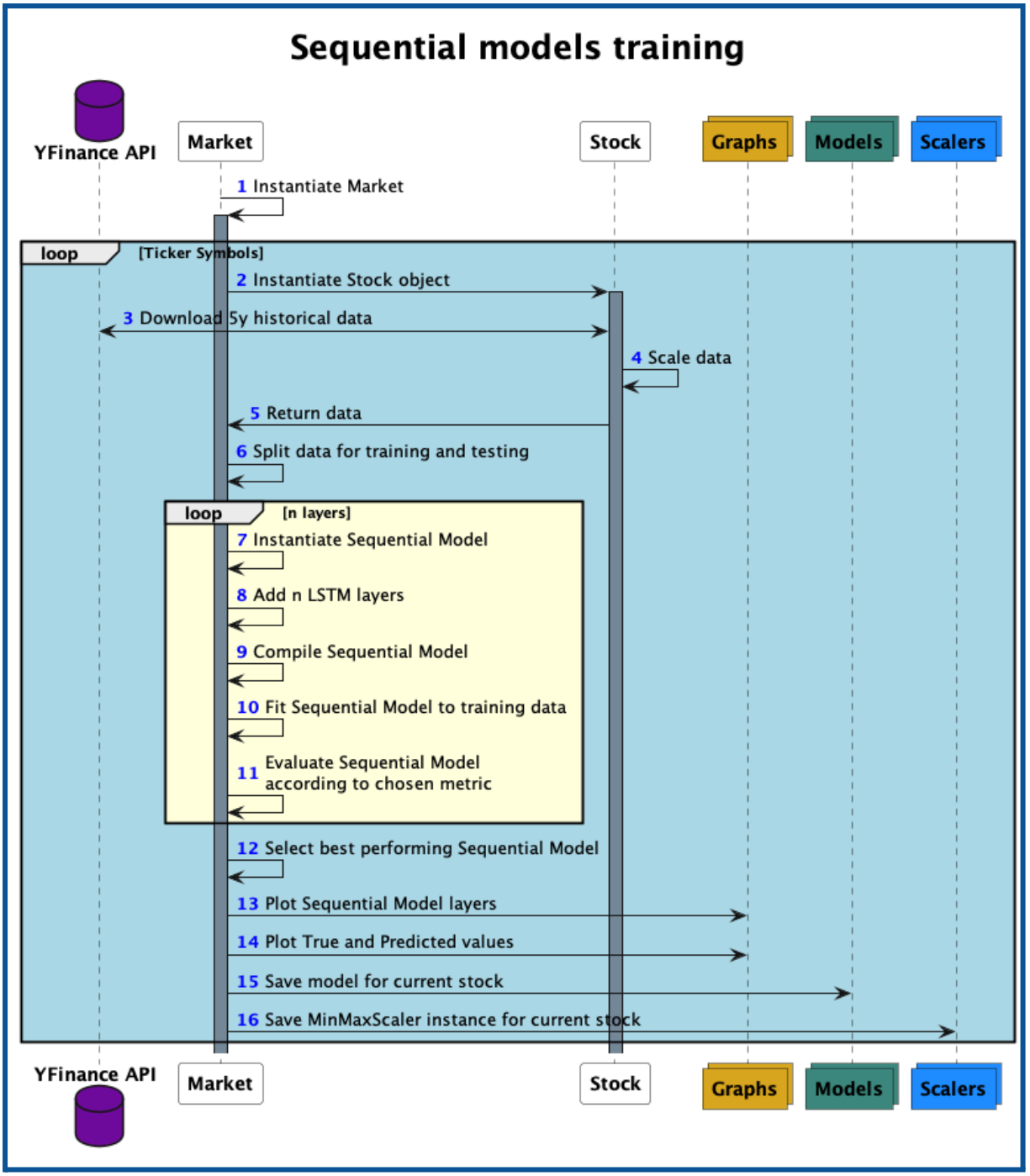


Diagramme UML train_models.py



Fichier current_portfolio.json


```
{
   "characteristics": {
       "numberOfAssets": 5,
       "return": 0.010124203105043996,
       "risk": 0.00010704227017568682,
       "sharpeRatio": 93.81530388473529,
       "capital": 100000,
       "value": 99339.94031524658
   },
   "stocks": {
       "VIRP.PA": {
           "companyName": "VIRBAC",
           "pricePerShare": 340.0,
           "return": 0.0007680301322678358,
           "risk": 0.00012876361078739855,
           "sharpeRatio": 5.32782614647658,
           "capital": 7762.581370414943,
           "weight": 0.07762581370414944,
           "shares": 22
       },
       "LIN.PA": {
           ...
       },
       "ARTO.PA": {
           ...
       },
       "ES.PA": {
           ...
       },
       "SAF.PA": {
           ...
       }
   }
}
```


Fichier suggested_portfolio.json


```
{
   "sell": {
       "VIRP.PA": 22
   },
   "suggestedPortfolio": {
       "characteristics": {
           ...
       },
       "stocks": {
           ...
       }
   }
}
```