Trading algorithmique sur cryptomonnaies

LE VU Mathis
VILLAMAUX Geoffroi

Sommaire

1) Introduction
2) Stratégie
3) Deep Learning
4) Structure Long Short-Term Memory
5) LSTM & Keras
6) Code
7) Résultats
8) Prédictions

Introduction

Au cours de ce tp, nous avons travaillé à l'élaboration d'un bot de conseil en trading sur cryptomonnaies.

Ce bot récolte des données issues de Yahoo Finance via l'API yfinance, puis traite ces données via la librairie de deep learning Keras.

Les paramètres optimaux en sortie du réseau de neurones de deep learning permettent de prédire l'évolution du spot.

Sur la base de ces prédictions, notre bot peut conseiller à l'utilisateur d'acheter ou non un actif.

Dans le cadre de nos mesures, nous avons travaillé avec les cotations du Bitcoin / Ethereum / Tézos.

Stratégie

Nous avons basé notre stratégie sur le sens de marché prédit.

Ce choix s'est basé sur le fait qu'il existe un delta entre nos valeurs de sorties et les valeurs réelles. On constate toutefois que le sens d'évolution prédit (valeur croissante / décroissante / constante) est majoritairement le bon.

Deep Learning

Le réseau de neurones utilisé pour le bot est un réseau de neurones RNN ayant une architecture LSTM proposée par la librairie Keras.

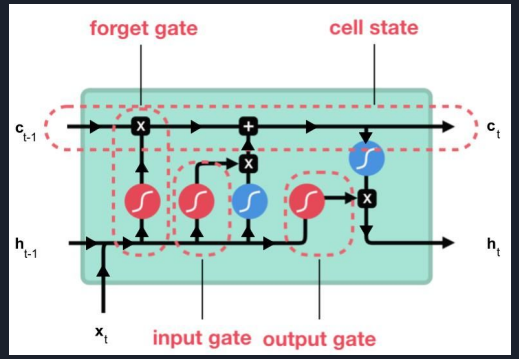
RNN : Réseau de neurones ayant un système de connexion récurrentes. Cela procure alors au réseau de la mémoire, permettant notamment de détecter des paternes.

LSTM : Réseau de neurones récurrents disposant d'une mémoire à court et long terme, permettant de contrer les problèmes d'explosion de gradient ou de vanishing.

Structure Long Short-Term Memory

Chaque neurone du réseau est composé de trois portes (input/output/forget) et de deux sorties ou états (Hidden / Cell).

Associées à des fonctions d'activation, le neurone est capable d'oublier ou de conserver des informations. L'information pertinente est alors transmise en sortie.



Long Short-Term Memory & Keras

Via un processus de forward propagation, on peut alors définir une valeur de sortie estimée.

Les paramètres sont ensuite ajustés lors du processus de back propagation. Ici, nous l'effectuons via la méthode Adam, une méthode de descente de gradient stochastique approximant les moments d'ordre 1 et 2.

En itérant un certain nombre de fois ce processus de forward propagation et de back propagation, nous définissons les paramètres optimaux (i.e minimisant la mean square error.)

Nous avons implémenté un LSTM via Keras en définissant le nombre de neurones par couches tout en associant au modèle une méthode de back propagation, ici Adam.

Code

Dans cette partie :

-nous importons les librairies et API nécessaires au bon fonctionnement du code

-nous choisissons sur quelles cryptomonnaies nous souhaitons investir (on peut en ajouter autant que l'on veut, dans la liste "produit")

-nous choisissons l'ancienneté des valeurs historiques

-nous affichons la courbe historique de chaque cryptomonnaie

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
from keras.models import Sequential
from keras.optimizers import Adam
from keras.layers import Dense, LSTM, LeakyReLU, Dropout
import yfinance as yf
from sklearn.preprocessing import MinMaxScaler

Produit=["BTC-EUR","ETH-EUR","XTZ-EUR"]

tab_decision=[]


for i in range(len(Produit)):

    crypto = yf.Ticker(Produit[i])

    # get historical market data
    #"1d", "5d", "1mo", "3mo", "6mo", "1y", "2y", "5y", "10y", "ytd", "max"

    Periode="2y"
    data = crypto.history(period=Periode)
    data['Date']=data.index
    data=data.rename_axis('index').reset_index()
    del data['index']

    price = data[['Close']]

    plt.figure(figsize = (15,9))
    plt.plot(price)
    plt.xticks(range(0, data.shape[0],50), data['Date'].loc[::50],rotation=45)
    plt.title(Produit[i] + " Price",fontsize=18, fontweight='bold')
    plt.xlabel('Date',fontsize=18)
    plt.ylabel('Close Price (EUR)',fontsize=18)
    plt.show()

    price.info()

    min_max_scaler = MinMaxScaler()

```

Code

Dans cette partie :

-nous splittons nos valeurs afin de déterminer notre ensemble de test et notre ensemble d'entraînement

-Plus loin, nous définissons également un set de validation. L'algorithme définit alors autant de paramètres optimaux que "d'epochs". Parmi le set de paramètres optimaux, via un procédé de cross validation, nous conserverons celui minimisant la MSE (c.f Courbe de validation loss).

```
49
50      norm_data = min_max_scaler.fit_transform(price.values)
51
52      """
53      print(f'Real: {price.values[0]}, Normalized: {norm_data[0]}')
54      print(f'Real: {price.values[500]}, Normalized: {norm_data[500]}')
55      print(f'Real: {price.values[1200]}, Normalized: {norm_data[1200]}')
56      """
57
58
59      def univariate_data(dataset, start_index, end_index, history_size, target_size):
60        data = []
61        labels = []
62
63        start_index = start_index + history_size
64        if end_index is None:
65          end_index = len(dataset) - target_size
66
67        for i in range(start_index, end_index):
68          indices = range(i-history_size, i)
69          # Reshape data from (history_size,) to (history_size, 1)
70          data.append(np.reshape(dataset[indices], (history_size, 1)))
71          labels.append(dataset[i+target_size])
72        return np.array(data), np.array(labels)
73
74      past_history = 5
75      future_target = 0
76
77      TRAIN_SPLIT = int(len(norm_data) * 0.8)
78
79
80      x_train, y_train = univariate_data(norm_data,
81                                         0,
82                                         TRAIN_SPLIT,
83                                         past_history,
84                                         future_target)
85
86      x_test, y_test = univariate_data(norm_data,
87                                       TRAIN_SPLIT,
88                                       None,
89                                       past_history,
90                                       future_target)
```

Code

Dans cette partie :

-nous déterminons les valeurs caractéristiques permettant la régression. nous avons en particulier déterminé un learning rate à 0,0002 après plusieurs tests

-nous compilons la RNN

-nous entraînons le modèle grâce aux valeurs déterminées précédemment

```
92
93      num_units = 100
94      learning_rate = 0.0002
95      activation_function = 'sigmoid'
96      adam = Adam(lr=learning_rate)
97      loss_function = 'mse'
98      batch_size = 5
99      num_epochs = 500
100     # Initialize the RNN
101     model = Sequential()
102     model.add(LSTM(units = num_units, activation=activation_function, input_shape=(None, 1)))
103     model.add(LeakyReLU(alpha=0.5))
104     model.add(Dropout(0.1))
105     model.add(Dense(units = 1))
106
107     # Compiling the RNN
108     model.compile(optimizer=adam, loss=loss_function)
109
110
111     model.summary()
112
113
114     # Using the training set to train the model
115     history = model.fit(
116         x_train,
117         y_train,
118         validation_split=0.1,
119         batch_size=batch_size,
120         epochs=num_epochs,
121         shuffle=False
122     )
123
124     loss = history.history['loss']
125     val_loss = history.history['val_loss']
126
127     epochs = range(len(loss))
128
129     plt.figure()
```

Code

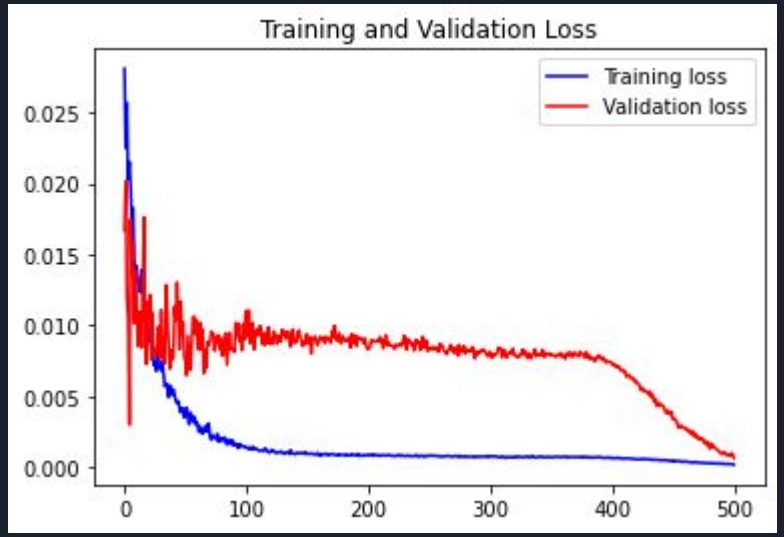
Dans cette partie :

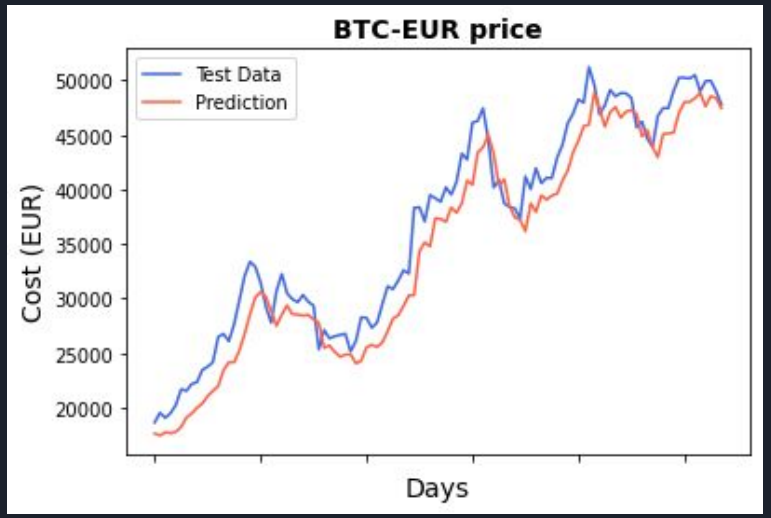
-nous affichons la courbe “Training and Validations Loss” et la courbe de Prédiction de prix

-nous renvoyons l’indication d’acheter ou non

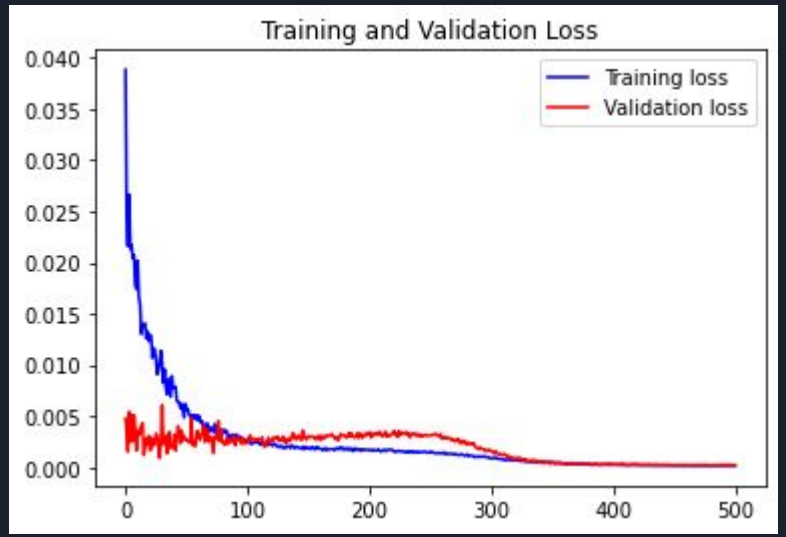
```
131     plt.plot(epochs, loss, 'b', label='Training loss')
132     plt.plot(epochs, val_loss, 'r', label='Validation loss')
133     plt.title("Training and Validation Loss")
134     plt.legend()
135
136     plt.show()
137
138
139
140
141     original = pd.DataFrame(min_max_scaler.inverse_transform(y_test))
142     predictions = pd.DataFrame(min_max_scaler.inverse_transform(model.predict(x_test)))
143
144     ax = sns.lineplot(x=original.index, y=original[0], label="Test Data", color='royalblue')
145     ax = sns.lineplot(x=predictions.index, y=predictions[0], label="Prediction", color='tomato')
146     ax.set_title(Produit[i] +  ' price', size = 14, fontweight='bold')
147     ax.set_xlabel("Days", size = 14)
148     ax.set_ylabel("Cost (EUR)", size = 14)
149     ax.set_xticklabels('', size=10)
150
151     def decision():
152         if predictions[0][len(predictions)-1]>predictions[0][len(predictions)-2]:
153             tab_decision.append(["Buy ", Produit[i]])
154         else:
155
156             tab_decision.append(["Don't Buy ",Produit[i]])
157     decision()
158
159 print(tab_decision)
```

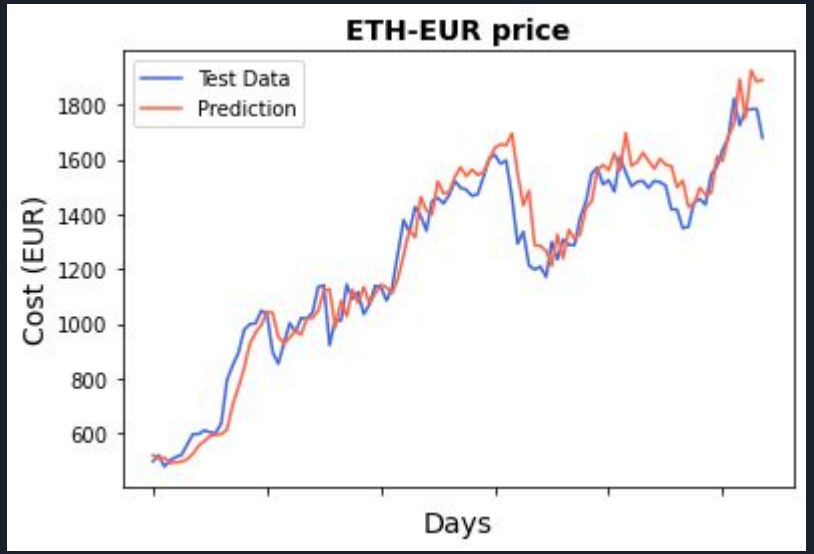
Résultats BTC sur les deux dernières années



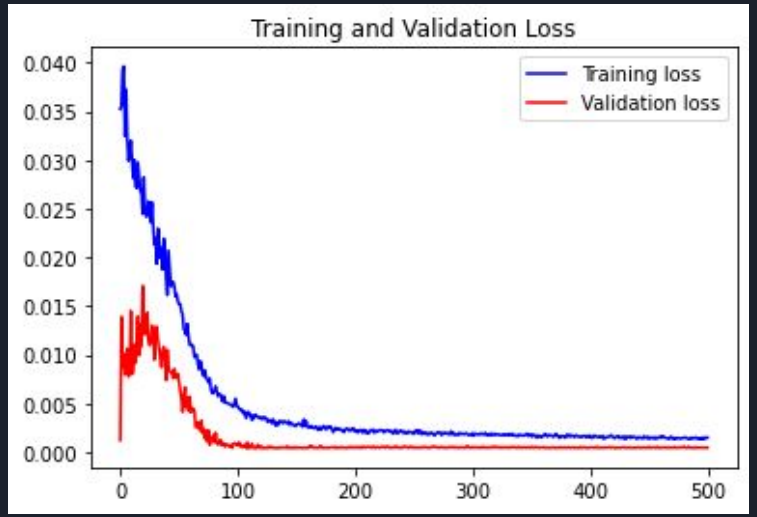


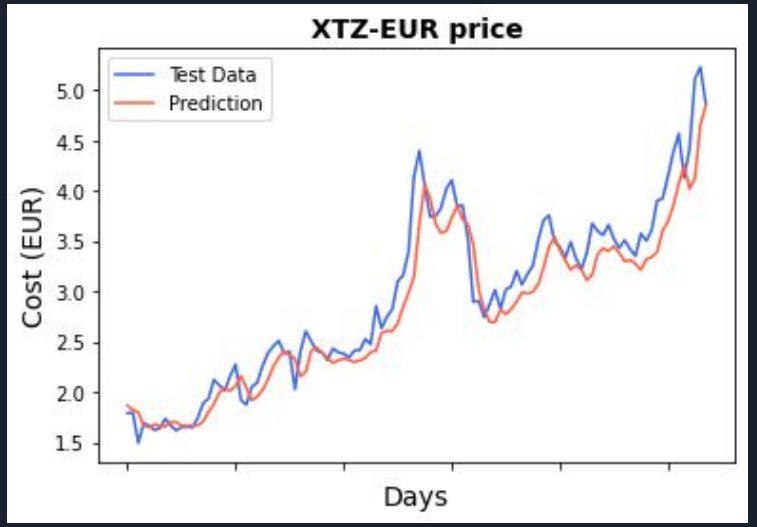
Résultats ETH sur les deux dernières années





Résultats XTZ sur les deux dernières années





Analyse des courbes obtenues

Courbes “Training and Validation Loss” :

Les MSE du train set et validation set convergent vers 0, le modèle semble bien calibré.

Courbes de Prédiction du prix :

Les différentes courbes de prédiction suivent la tendance des courbes de test, avec un léger offset (la courbe de prédiction a du retard sur celle de test).

Prédictions Achats / Pas Achat

```
def decision():
    if predictions[0][len(predictions)-1]>predictions[0][len(predictions)-2]:
        tab_decision.append(["Buy ", Produit[i]])
    else:

        tab_decision.append(["Don't Buy ",Produit[i]])
```

La prédiction permet de déterminer le prix de la cryptomonnaie pour le jour qui suit celui de la dernière valeur en donnée. Ainsi nous regardons si la prédiction est supérieure à la précedente, ou non, pour donner l'information d'acheter ou ne pas acheter.

Nous avons ainsi obtenu le 7/04/2021 pour le 8/04/2021 :

```
[["Don't Buy ", 'BTC-EUR'], ['Buy ', 'ETH-EUR'], ['Buy ', 'XTZ-EUR']]
```