Informe técnico: Machine Learning para predicción de valores de criptomonedas

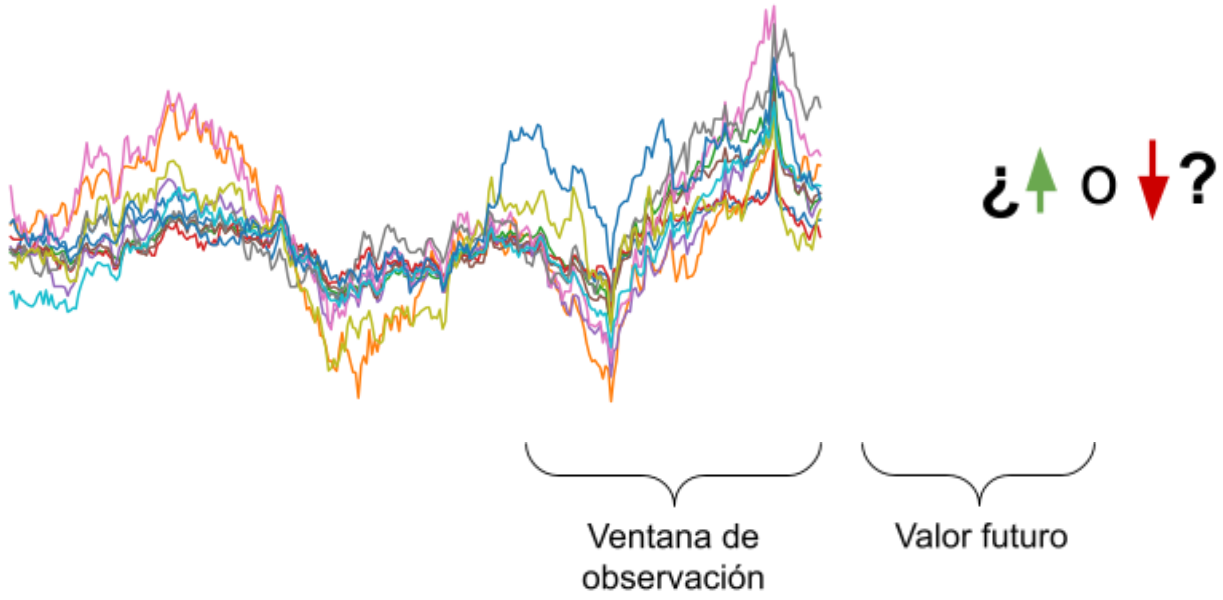
En este informe se presenta el algoritmo de Machine Learning desarrollado para pronosticar el precio de criptomonedas. En la primera sección se describen las características generales de funcionamiento del algoritmo. En la segunda, se presentan los detalles de implementación del algoritmo. Luego se describe la organización de los archivos de código para su uso. Por último se presentan los resultados y conclusiones de esta etapa del trabajo.

La implementación de los algoritmos descritos en este informe se encuentran en el repositorio de github: https://github.com/gon-uri/cripto_forecast .

1. Generalidades del algoritmo

1.1 Objetivo

El objetivo es pronosticar si el mercado para determinada criptomoneda va a subir o bajar en un dado lapso de tiempo. Para esto vamos a observar en una ventana de tiempo como es la variación de un conjunto de 11 criptomonedas.

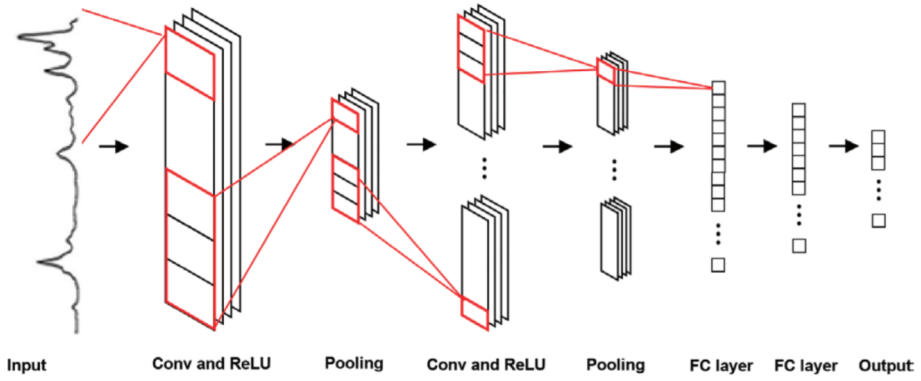


1.2 Algoritmo

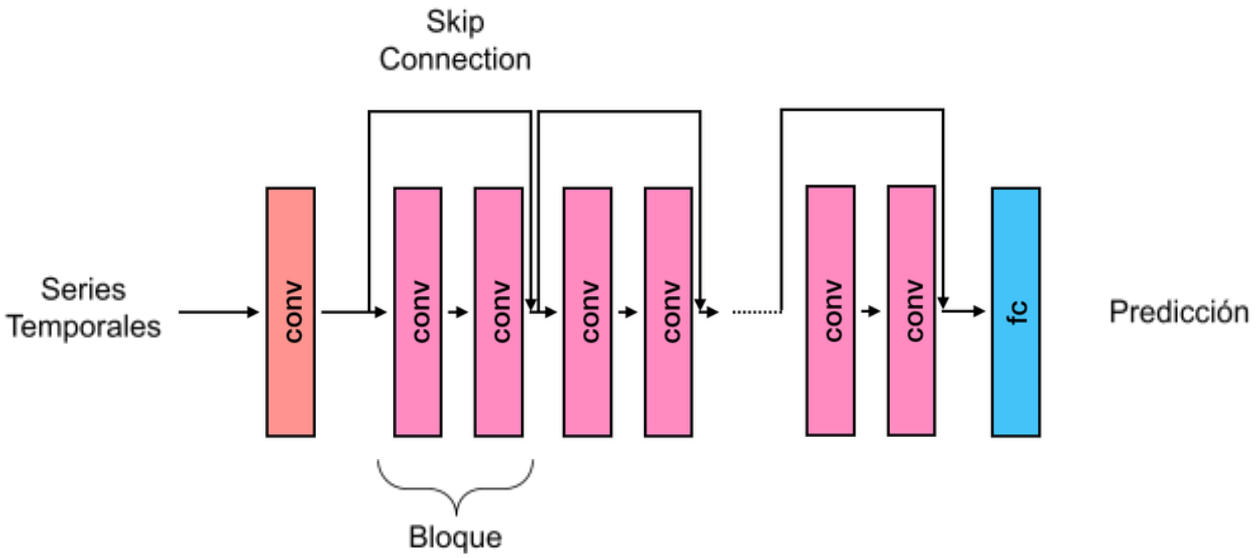
Para alcanzar el objetivo, se utilizó un algoritmo de Deep Learning. Se decidió esto luego de comprobar que algoritmos más simples de machine learning no resultaban efectivos para la

tarea. El tipo de algoritmo utilizado es una **Red Convolucional 1D** en una configuración tipo **ResNet**.

Una **Red Convolucional 1D** es un algoritmo equvalente a las famosas Redes Convolucionales utilizadas para imágenes, pero modificadas para procesar señales temporales (https://en.wikipedia.org/wiki/Convolutional_neural_network). La idea es que la red aprende a detectar automáticamente cuales son los features importantes a mirar en la serie temporal para realizar tarea objetivo (generalmente, minimizar la función de costo). A continuación se presentan un esquema de una Red Convolucional 1D:



Por otro lado, una configuración tipo **ResNet** es una manera de poder agregar capas a redes profundas y poder entrenarlas de manera efectiva (https://en.wikipedia.org/wiki/Residual_neural_network). La idea es agregar las capas en formas de bloques (en nuestro caso de dos capas convolucionales cada uno) que contengan además una conexión directa entre la entrada y la salida (skip connection). De esta forma, es posible entrenar una red profunda (con mucha expresividad) y no incurrir en problemas (como el vanishing gradient). A continuación se presenta un esquema de la configuración tipo **ResNet** utilizada en nuestro algoritmo:



2. Implementación del modelo

En esta sección detallaremos los distintos pasos presentes en el código para implementar el modelo de deep learning comentado en la sección anterior.

2.1 Dataset

En primer lugar se cargan los datos, se los ordena temporalmente y se verifica si tienen o no datos faltante. El dataset utilizado tiene información sobre el valor de las criptomonedas cada 1 minuto:

```
['close_ada',      'close_avax',      'close_btc',      'close_cake',
'close_dot','close_eth',   'close_link',   'close_matic',   'close_sol',
'close_theta','close_vet']
```

Para entrenar el algoritmo se utiliza la información de todas las criptomonedas.

Se tomó como inicio del dataset la fecha a partir de la cual se tenía información de los 11 activos (19/2/2021). Sin embargo, dentro de este rango de tiempo se encuentran valores faltantes: ciertos minutos para los cuales no se cuenta con la información de algunos valores. Este inconveniente se soluciona detectando estos faltantes y evitando usarlos para el entrenamiento o testeo del modelo, ya que estos consisten de secuencias de datos para tiempos consecutivos (sin saltos temporales debido a datos faltantes).

Notamos que para generar los conjuntos de entrenamiento se emplearon los datos de los últimos 2 meses, mientras que para el conjunto de testeo los datos de los últimos 15 días.

2.2 Preprocesamiento de los datos

En vez de trabajar con los valores de cierre de las criptomonedas, se trabaja con una cantidad que mide el cambio porcentual de la misma, denominada *log return*:

$$log\,(r_i) \;=\; log\left(\frac{Vi-Vj}{Vj}\right),$$

donde V_i es el valor de la criptomoneda para el tiempo i, mientras que V_j es el valor para un tiempo $j < i$.
Esta representación confiere varios beneficios: estacionaliza la serie, cuantifica variaciones relativas (no absolutas) y posee una propiedad aditiva.

2.3 Hiperparámetros

A continuación se comentan los hiperparámetros relevantes en las distintas partes del algoritmo.

Criptomoneda:

COIN: criptomoneda cuyo valor se quiere predecir.

PASOS_FUTURO: lapso de tiempo (en minutos) para el cual se busca dar una predicción sobre el valor de la criptomoneda seleccionada.

Dataset:

PORCENTAJE: porcentaje del dataset preprocesado que se va a utilizar. Conserva el porcentaje final de los datos, es decir, los más recientes.

TRAIN_RATIO: porcentaje del dataset anteriormente seleccionado que se utiliza como conjunto de entrenamiento. Asimismo, 1-TRAIN_RATIO es el porcentaje de datos que se destina al conjunto de testeo. Notar que el conjunto de testeo contiene los datos más recientes.

SEQ_LEN: los datos de testeo y entrenamiento se dividen en secuencias temporales, SEQ_LEN es la longitud de dichas secuencias.

REDUCTION_STEP: para cada valor de una secuencia temporal definida arriba, se suman los REDUCTION_STEP datos consecutivos. Ejemplo:

secuencia: (S_1, S_2, …, S_{SEQ_LEN})

secuencia reducida: (SR_1, SR_2, …, $SR_{SEQ\_LEN-REDUCTION\_STEP}$)

donde $SR_i = \sum_{j}^{REDUCTION\ STEP} S_i$

Esta reducción es importante para evitar fluctuaciones abruptas de los datos conservando las tendencias generales.

Los conjuntos de entrenamiento y testeo con secuencias reducidas son los datos de entrada del modelo de Machine Learning empleado.

Entrenamiento:

NOISE_LEVEL: porcentaje de ruido gaussiano que se utiliza como una estrategia para evitar sobreentrenamiento de los datos.

REGRESSION: si es 1 el modelo considera un problema de regresión, si es 0 el problema es considerado de clasificación y los valores *log return* son transformados a 1 o 0 si el valor sube o baja, correspondientemente.

Ademas de estos hiperparámetros presentados, también deben considerarse los usuales como batch size (BATCH_SIZE), learning rate (lr). Todos los hiperparámetros de la red neuronal utilizada se guardan en un archivo de texto y un diccionario (ver sección “Utilización del modelo”)

2.4 Entrenamiento del modelo

Una vez que se divide al dataset preprocesado en el conjunto de entrenamiento y testeo (y estos a su vez en secuencias reducidas) utilizando las clases y funciones definidas en el algoritmo, se realiza un proceso más sobre los datos.

Para evitar resultados con sobreentrenamiento y poca o nula precisión en la predicción se utilizan técnicas de data augmentation sobre el conjunto de secuencias de entrenamiento (al conjunto de testeo no se lo modificó). Para generar una mayor variedad de datos a partir de los existentes se aplica sobre cada secuencia una transformación elegida al azar sobre un conjunto de transformaciones posibles, que incluyen la introducción de ruido gaussiano, dropout, eliminar la información sobre una de las criptomonedas, reemplazar una sección de la secuencia con valores nulos o con valores otra sección, etc.

Es importante notar que los mecanismos de data augmentation se aplican sobre las secuencias y no sobre el conjunto entero de entrenamiento, y que resulta crucial para obtener buenos resultados con el algoritmo.

Finalmente, se procede a realizar el entrenamiento, para lo cual hay que definir una cantidad de épocas (epochs). Al finalizar cada una de las épocas se informa el valor de la función de pérdida y la precisión tanto para entrenamiento como test, el valor del área bajo la curva ROC (AUC), y se salva el modelo de forma automática si el AUC se incrementa (ver sección “Utilización del modelo”).

3. Utilización del modelo

3.1 Entrenamiento

El código de entrenamiento se encuentra en el notebook **Train_Algorithm.ipynb**. Este algoritmo se encarga de leer los datasets de las distintas monedas, limpiarlos, procesarlos, entrenar el modelo y generar los archivos necesarios para luego poder correr el modelo. Los archivos que genera son los siguientes:
_ **1DConv_close_btc_ADP.pt** : Contiene los pesos del modelo entrenado.
_ **1DConv_close_btc_ADP_dictionary.pickle** : Contiene los hiperparámetros del modelo, los cuales son necesarios para poder usar el modelo.
_ **1DConv_close_btc_ADP_parameters.txt** : Contiene un archivo txt legible con los hiperparámetros (no se usa para correr el modelo).

En el repositorio de github estos archivos se encuentran en la carpeta **cripto_forecast/models**.

3.2 Uso del modelo

Una vez generados estos archivos, se puede usar el modelo ya entrenado con el archivo **Use_Algorithm.ipynb**. Este archivo levanta los hiperparámetros del diccionario, define el modelo y carga los pesos aprendidos durante el entrenamiento. Como salida, devuelve el

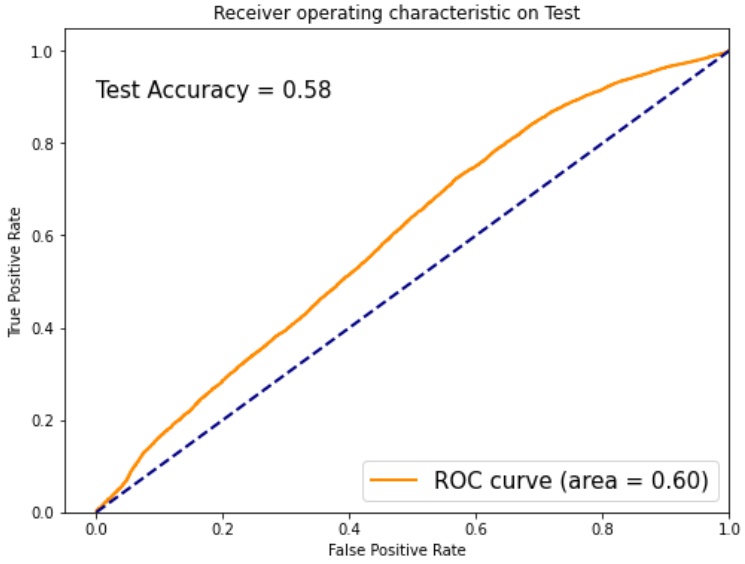
valor de predicción a futuro. Es decir que pronostica el valor de determinada cripto, en nuestro caso el Bitcoin, para el instante de tiempo posterior al último dato del dataset. Como agregado, decidimos incluir en **Use_algorithm** el cómputo de la curva ROC y el accuraccy sobre el test set.

Observación: Para presentar los resultados en Use_Algorithm.ipynp se levantan los datos desde el repositorio de github. No podemos subir al repositorio archivos mayores a 25mb, por lo cual subimos un archivo con los datos ya preprocesados que ocupa menos espacio. Pero en un modo operativo real, Use_algorithm levantará los datos en crudo de la base de datos.

4. Resultados y Conclusiones

Resultados

Se realizó una búsqueda (no sistemática y no exhaustiva) de parámetros para la moneda Bitcoin. En las regiones exploradas se encontró que el modelo con mayor área bajo la curva (0.6) se obtuvo en la predicción de la señal a 1 hora, mirando una ventana de valores de 10 horas previas. Para este modelo la curva ROC obtenida es la siguiente:



Nótese además que se obtiene un valor de accuracy en el test set del 57%. Esto quiere decir que el 57% de las veces se acierta si el mercado de dicha cripto está a la baja o a la alza.

Conclusiones

A continuación listamos las principales conclusiones más relevantes que obtuvimos en el proceso de construcción y evaluación del modelo:

- Dada la complejidad y cuasi-aleatoriedad de los datos, para lograr extraer información de las señales se precisa de un modelo complejo (red profunda y expresiva). Los modelos simples devuelven los mismos resultados que chance (50%).
- Dentro de los regímenes explorados, la arquitectura de red convolucional 1D funciona mejor que la arquitectura de Multilayer Perceptron y las arquitecturas Recurrentes.
- Los modelos entrenados en forma de regresión resultan mejores que los entrenados en forma de clasificación.
- Los modelos entrenados utilizando solo los valores más recientes de las series resultan mejores que aquellos entrenados con la totalidad de las mismas. De esto es posible concluir que las series poseen regímenes sistémicos distintos a lo largo del tiempo, es decir que la relación entre las distintas criptomonedas no se mantiene constante.
- El problema principal a evitar es el overfitting. Dado que la cantidad de instancias de entrenamiento no es muy grande para el tamaño de red necesaria, resulta fundamental el proceso de data augmentation para el éxito del entrenamiento.
- Resulta beneficioso realizar un subsampleo de la serie (hiperparámetro REDUCTION_STEP) y no utilizar secuencias muy largas con valores cada 1 minuto cuando se busca predecir en una escala temporal más grande.