Agentes neuronales para invertir en bolsa en el simulador MAStock

Rafael Castillo, Gabriel Prat

Departament de Llenguatges i Sistemes Informàtics (LSI)
Universitat Politècnica de Catalunya (UPC)
rafael@lsi.upc.es; gprat@lsi.upc.es

1. Introducción

Presentamos una extensión del sistema MAStock con algunas clases que permiten crear fácilmente agentes que tengan una parte deliberativa y una parte neuronal. La extensión proporciona básicamente la parte neuronal del agente.

Hay muchas formas de enfocar el problema de ganar dinero invirtiendo en bolsa con redes neuronales. Seguidamente intentaremos explicar el por qué del conjunto de soluciones escogidas.

Para empezar debemos identificar nuestro problema. Podríamos pensar que para cumplir nuestro objetivo que es ganar dinero invirtiendo un posible problema a resolver es la predicción de subidas o bajadas en los diferentes valores de la bolsa. Otra manera, más genérica, de enfocarlo seria resolver el problema de predecir el siguiente valor de la serie temporal de los precios de los diferentes valores. Aún otro enfoque sería, dadas varias estrategias deliberativas decidir con redes neuronales cuál de ellas acertará en su decisión dadas ciertas circunstancias del mercado. Y sin duda, la mejor solución que podríamos dar, o al menos la más cómoda ya que nos libra de tener que conocer mucho sobre el problema es construir una red que decida qué acción tomar; una red que, por ejemplo, tenga una neurona para cada valor controlado y que si la salida para esta neurona es positiva tenemos que comprar acciones, si es cero no hacer nada y si es negativa tenemos que vender.

A primera vista ya podemos ver que la última solución es más que utópica: La función a modelar depende de todos los valores que queramos controlar, de las posesiones de acciones y dinero del agente que utilice la red y de la situación del mercado. Por esto es esperable que el número de parámetros de entrada de la red sea muy grande en comparación a las muestras significativas que podamos obtener dado que las condiciones en los mercados no son nada estables. Además también es de esperar que la información provinente de uno de los valores controlados por la red sea contradictoria con la información proporcionada por otros valores de manera que imposibiliten o dificulten mucho la tarea de aprendizaje.

Por esta razón hemos intentado proporcionar herramientas para desarrollar agentes que intenten resolver el problema de forma híbrida aprovechando la información disponible sobre el problema para no sobrecargar la tarea de la red.

Así pues con nuestra extensión proporcionamos una manera fácil de crear agentes que tomen decisiones a partir de los valores futuros de las series temporales de los precios de los valores predichos por una red neuronal.

2. Puntos de decisión

Formalicemos el problema. Llamémosle a la función que dado un instante de tiempo t nos devuelve el precio del valor i (de los k valores totales) en este instante $v_i(t)$. Entonces nuestro problema es predecir el valor de $v_i(t+n)$, ..., $v_i(t+n+m)$ en el instante t.

Nos queda concretar muchas decisiones que no hemos tomado aún.

A continuación enunciamos las más relevantes:

- Objetivo de predicción
  - Valor exacto: $v_i(t)$
  - Tendencia: $\Delta v_i(t)$
- Número de valores a predecir
  - Uno: m=0
  - Múltiples: m>0
- Número de redes neuronales
  - Una

$$y = \begin{pmatrix} v_1(t+n) & \dots & v_1(t+n+m) \\ \vdots & \ddots & \vdots \\ v_k(t+n) & \dots & v_k(t+n+m) \end{pmatrix}$$

  - Una por valor

$$\begin{aligned} y_1 &= \big(v_1(t+n) \dots v_1(t+n+m)\big) \\ &\vdots \\ y_k &= \big(v_1(t+n) \dots v_k(t+n+m)\big) \end{aligned}$$

  - Una por valor e instante de tiempo

$$\begin{aligned} y_{11} = v_1(t+n) &\cdots y_{1m} = v_1(t+n+m) \\ \vdots &\quad \ddots \vdots \\ y_{k1} = v_k(t+n) &\cdots y_{km} = v_k(t+n+m) \end{aligned}$$

- Momentos de entrenamiento
  - Una sola vez al principio
  - Varias veces reiniciando los pesos
  - Varias veces sin reiniciar los pesos
- Parámetros de entrada
  - Valores precedentes de la serie
  - Valores derivados de los anteriores
  - Parámetros macro-económicos

    - Precio de las ofertas en curso
    - Opinión de expertos
    - Noticias
    - ...
- Y finalmente: Qué hacemos con los valores?

Nuestro desarrollo pretende dar un marco de trabajo que fijados unos valores concretos para cada una de las decisiones especificadas arriba podamos crear fácilmente agentes que respondan de distintas maneras a la última pregunta. Así pues tenemos una clase que implementa la parte neuronal del agente para una toma de decisiones ya dada y que permite una redefinición de la función que toma la decisión de acción de manera que podemos crear múltiples agentes neuronales fácilmente con diferentes estrategias deliberativas en que se basan no sólo en los parámetros que ya teníamos disponibles en MAStock si no también en los valores predichos por la red.

A continuación explicaremos cuál ha sido nuestra toma de decisiones para la parte neuronal de los agentes.

3. Especificaciones del sistema

En nuestro caso hemos decidido crear una red neuronal para cada valor por la razón, ya comentada anteriormente, que las entradas provinentes de diferentes valores bursátiles podrían ser contradictorias para la red e imposibilitar o al menos dificultar mucho su aprendizaje. También decidimos intentar predecir un solo valor en el futuro por una razón parecida: Al intentar predecir precios en más de un instante de tiempo con la misma red, tendríamos dos neuronas que estarían computando distintas salidas de nuestra función $v_i(t)$, p.e., $v_i(t+1)$ y $v_i(t+2)$. Pero estas dos salidas no son independientes entre sí y por lo tanto no está claro que sea buena idea propagar información de error de predicción con un algoritmo estilo *backpropagation* en una red neuronal ya que nuevamente podríamos estar propagando informaciones contradictorias para la red.

Por otro lado predecir tendencias parece algo más fácil que predecir valores exactos, pero fijémonos en que realmente lo que estamos haciendo al poner como objetivo de predicción la tendencia (subida o bajada) es discretizar nuestro objetivo real. Así pues perdemos información al hacerlo y puede que simplifiquemos la labor de la red o puede que esta información simplificada la confunda al tener entradas continuas muy distintas de precios pasados que llevan exactamente al mismo resultado (i.e. pasar de 1 a 2 es subir y pasar de 1 a 15 genera la misma salida).

Por lo que respecta al entrenamiento, parece bastante optimista pensar que bastará con un entrenamiento inicial para predecir el resto de valores de una serie dado el elevado grado de aleatoriedad que se observa en las series temporales de los precios de valores en bolsa. Así pues deberemos reestrenar la red periódicamente para no intentar predecir precios con una información obsoleta. Así mismo, para que esta información adquirida en un cierto momento y

que seguramente ya será obsoleta no perjudique al entrenamiento actual, hemos decidido resetear la red antes de cada reentreno. De todas formas también podríamos reentrenar sólo con los nuevos valores sin reiniciar pesos de forma que la información adquirida previamente tendería a perderse. De todas formas no está muy claro que el resultado fuera diferente.

La última decisión que nos queda son los parámetros de entrada de la red. En MAStock no disponemos de información de noticias, opinión de expertos y disponemos de parámetros macro-económicos calculados sólo a partir de los precios de los valores controlados. Además en nuestra versión estos parámetros no son consultados por ningún agente de forma que no tienen efecto en el mercado. Así pues, disponemos de información de nuestra propia serie de tiempo de precios y de las ofertas actuales. En nuestra primera implementación sólo tendremos en cuenta valores pasados de la serie y valores derivados de estos. Se podría considerar añadir información de ofertas.

Así pues nuestra decisión ha sido:

- Objetivo = valor exacto
- Valores a predecir = 1
- Número de redes = número de valores
- Reentrenos periódicos con reinicio de pesos
- Entrada = valores de la serie temporal y derivados

Por lo tanto tenemos k redes y la salida de la red *i-éssima* sería:

$$y_i = v_i(t+1)$$

La elección de los parámetros de entrada también es una cosa que hemos intentado que sea fácilmente cambiable de manera que, cómo veremos en la descripción de la arquitectura, se ha encapsulado en un método sobrecargable.

De todas formas en la implementación base se calculan algunos valores derivados de la serie temporal, en particular el ratio de volatilidad (o volatility rate) y el grado de aleatoriedad (o random walk). Calculados cómo:

$$VI = \frac{(P_{HIGH} - P_{LOW})(P_{CLOSE} - P_{OPEN})}{|\text{average\_of\_numerator}|}$$

$$RWI = \frac{\Delta p_t(N_\tau)}{\Delta p_t \sqrt{N}}$$

Directamente parametrizable cómo parámetros de arranque del agente en el fichero *xxx.conf* tenemos:

- El tiempo de muestreo de valores en la serie temporal por día (0.1=10 veces por dia)
- El tamaño de la ventana utilizado para calcular RWI y VI
- El tamaño de la ventana utilizada para entrenar la red
- Los ticks de utilizacin de la red antes de reentrenar (por el momento utilizamos el valor del parámetro anterior).

4. Arquitectura

Nuestra extensión consta de dos nuevos packages: **timeSeries** y **neural**. Para minimizar el acople con el sistema MAStock y as permitir fácilmente la adaptación de la extensión a nuevas versiones de la plataforma, los únicos cambios que se han realizado en el sistema son en las clases: **ScenarioAgent** y **SABuySellBehav**. En estas dos clases hemos añadido un par de llamadas a nuestras clases del package neural en los métodos de inicialización y de llegada de un nuevo tick.

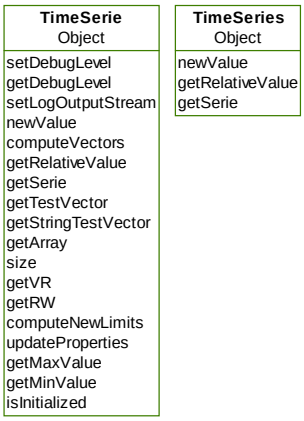

Figura 1. Diagrama de clases del package timeSeries

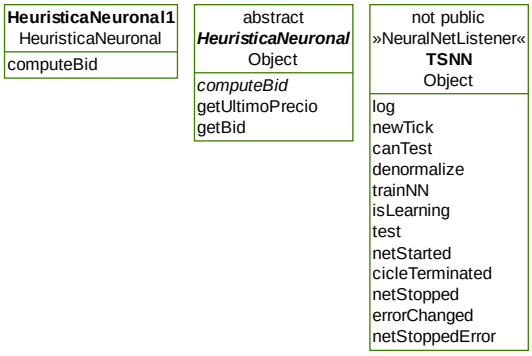

Figura 2. Diagrama de clases del package neural

Así pues nuestro código es suficientemente independiente de la plataforma en sí. En la fig. 1, podemos ver las dos clases que conforman este paquete. Ba-

sicamente **TimeSerie** contiene una implementación de una serie de tiempo y **TimeSeries** es sólo una clase que contiene varios objetos de la clase anterior. Así estas dos clases serán utilizadas por la **HeuristicaNeuronal** para calcular los valores de entrenamiento y test. Esta sólo tiene que instanciar un objeto y hacerle llamadas a su método **newValue** con cada nuevo valor. Cuándo la clase ya esté inicializada podrá llamar a **getArray** para que le devueva el array de entrenamiento. El metodo que define las columnas de este array es **computeVectors** y sólo se encarga de, una vez calculados los parametros RWI y VI al llegar un nuevo valor construir una nueva fila de valores para la matriz que será la entrada de la red. Así pues si queremos utilizar variables de entrada diferentes sólo tenemos que crear una subclase que redefina este método. Por defecto los valores de un vector de entrada serán:

$$v_i(t-4) \dots v_i(t-1)\ RWI_i(t-3) \dots RWI_i(t-1)\ VI_i(t-3) \dots VI_i(t-1)$$

Dónde:

- $v_i(t)$ Es el precio del valor i en el instante t
- $RWI_i(t)$ Es el indicador de camino aleatorio para el valor i calculado con la ventana del instante t
- $VI_i(t)$ Es el indicador de volatilidad para el valor i calculado con la ventana del instante t

En la fig. 2, podemos ver también el diagrama de clases para el paquete de redes neuronales. La clase **TSNN** (Time Series Neural Network) es una implementación de una red neuronal que utiliza una serie temporal cómo entrada. Sólo tenemos que llamar al método **newTick** i se pasará el nuevo valor a la serie temporal. Despues se comprobará si toca (re)entrenar y si la serie ya tiene valores para hacerlo y en este caso se entrenará la red en un *thread* concurrente. Finalmente se comprobará si tenemos la red entrenada y de ser así se pasará el valor recibido por la red y se devolverá el valor predicho. Esta clase no implementa el método **computeBid** que es el responsable de devolver una acci'on (comprar, vender o no hacer nada) dada la información disponible en MAStock y la proporcionada por la red.

Resumiendo, para implementar un agente neuronal utilizando nuestro sistema se tiene que:

1. (Opcional) Sobrecargar el método **computeVectors** en una subclase de **TimeSerie** para personalizar los valores de entrada de la red.
2. Implementar una subclase de **HeuristicaNeuronal** que defina el método **computeBid**

Y no tenemos que preocuparnos por crear, entrenar ni utilizar la red neuronal, sólo definir los parámetros configurables.

5. Trabajo futuro

Por el momento hemos creado un *framework* para la creación y prueba de agentes neuronales basados en predicción de valores futuros de los precios. Ahora se deberían probar diferentes implementaciones de estos agentes con diferentes valores de entrada de la red y sobretodo con diferentes estratégias en la parte deliberativa para poder sacar conclusiones sobre su efectividad. Otro trabajo interesante sería añadir más parámetros calculados a la serie temporal para poder jugar con más combinaciones de parámetros de entrada. También en esta línea seria interesante ver cómo afectan al comportamiento de la red el tener en cuenta otros parámetros de entrada a parte de los relacionados directamente con la serie cómo por ejemplo alguno de los mencionados en apartados anteriores. Finalmente una acción muy prometedora si la predicción resultase suficientemente acertada sería implementar varias redes para cada valor para predecir el precio en diferentes instantes futuros y así poder decidir mejor si aunque el precio suba me interesa vender (porque luego va a bajar) o no (porque seguirá subiendo).

Referencias

1. Eitan Michael Azoff. *Neural Network Time Series: Forecasting of Financial Markets*. John Wiley & Sons, 1994.
2. David Sánchez, Ignasi Belda, Pedro López, Esteve Almirall, Oriol Plazas, and Ulises Cortés. Mastock: A multi-agent system stock market test-bed. Description of the MAStock system, 2004.