Predicción de bolsa

Oscar Fernández Armengol, Javier Martínez Riberas

[1] **minería de datos**

[2] **universidad de Burgos**

Resumen. Nuestro trabajo se basa en dada la introducción de Siraj Raval, probar el algoritmo con modificaciones en los hiperparametros y con diferentes datos y comprobar los resultados, para poder determinar si la teoría de la bolsa es correcta o no y si hay alguna manera de predecir qué tendencias tendrá. Esto se llevará a cabo con redes neuronales.

Palabras clave: Bolsa, redes neuronales.

1 Introducción

Analizando la bolsa de valores podemos ver que no es completamente aleatoria, con lo cual, podemos crear un modelo para intentar predecir los puntos que subirá o bajará. Vamos a probar varios códigos, colgados en github, los testearemos, analizaremos los resultados y si es necesario arreglaremos para poder ejecutarlos. Uno de los programas creado por Siraj Raval, antes de empezar a probar hay que tener en cuenta que estos algoritmos no son, ni mucho menos, para hacerte millonario, el postula que, si es mejor invertir con estos algoritmos, pero es mejor, por el simple hecho, de que es un poco mejor que de manera aleatoria. Hay algoritmos que aparte de analizar los atributos de un histórico, como puede ser cualquier empresa que se pueda mirar en "Google Finance" (aplicación de Google donde puedes ver los históricos a tiempo real de las acciones de una empresa en la bolsa) también analizan el factor social, por ejemplo, analizando los tweets que hablen de una empresa concreta y mirar si son positivos o negativos y añadir esos resultados a nuestro algoritmo, para poder mejorarlos. Nosotros no llegaremos a ese nivel.

2 Descripción general del problema

La bolsa se basa en puntos que se actualizan a tiempo real dependiendo de muchos factores (ventas que hacen, compras, opiniones, opas hostiles, absorciones, …). sabiendo los puntos que tiene una empresa puedes invertir en ella, si cuando vas a vender tus acciones, hay menos puntos que cuando compraste, perderás dinero, si hay más, ganarás. Es algo más complejo, pero esto es en esencia el funcionamiento del mercado.

El problema al que nos enfrentamos es el de intentar predecir si es buena idea o no el invertir en una empresa gracias a instancias de los históricos.

3 Descripción de los algoritmos y los datos

3.1 DATOS

Hemos conseguido los datos de Google Finance, descargando los datos como csv una funcionalidad que nos proporcionan, y nos hemos quedado con los puntos con los que cierra el día ya que creemos que es el valor menos aleatorio para poder intentar predecir

las subidas o bajadas de la bolsa. Esta intuición nuestra se debe a que los puntos al principio del día pueden cambiar mucho por cosas que han pasado durante la noche a las cuales la empresa tiene que responder, el alto del día y el bajo del día pueden ser arbitrariamente altos y bajos ya que lo que pase a lo largo del día puede causar acciones impulsivas por parte de los accionistas. Alphabet (NASDAQ:Googl) es la empresa seleccionada para intentar predecir sus valores futuros, esto se debe a que es de las pocas empresas que tenían datos disponibles (la mayoría en España que es donde miramos primero casi no tenían históricos disponibles ni en google ni en yahoo finance). El periodo para permitir el entrenamiento ha sido de 5 años ya que solo se puede acceder a un dato por día. Nos hemos planteado el recolectar datos a mano, pero resultaba muy complicado ya que tendríamos que aprender a hacer scraping. El conjunto de datos que hemos usado al final es el que está el googl5.csv.

3.2 RNN

La red neuronal recurrente es el clasificador que utilizamos en el código de predicción de bolsa, es una clase de red neuronal artificial, la cual supera algunas restricciones de las redes neuronales convolucionales y las tradicionales: tanto el no tener un estado que poder pasar entre entradas como el hecho de que la entrada y la salida siempre son del mismo tamaño. Las RNN en contraste si pueden tener entradas de distintos tamaños y pasar estados entre distintas predicciones o entrenamientos.

Las RNN siguen teniendo algunos problemas como el hecho de que la información de los estados anteriores va desapareciendo con el tiempo. Este problema causa grandes problemas cuando hay dependencias a lo largo de mucho tiempo y se soluciona usando una arquitectura distinta a la habitual para las neuronas habituales (LSTM).

3.3 LSTM

Las LSTM son redes neuronales en las que cada neurona recibe la salida de la anterior propagación como entrada al igual que las RNN. Estas se diferencian de las RNN ya que pasan tanto un estado a través del tiempo (la línea horizontal de arriba en el gráfico) y la salida de la última propagación. Con la entrada actual y la salida anterior hacemos tres cosas:averiguamos qué partes del estado que queremos mantener y cuáles borrar , qué información queremos añadir al estado. La última parte es la salida, pero para esta parte debemos haber conseguido las otras dos ya que el estado va a pasar por la tanh para pasar a ser un valor entre [-1, 1] para así ponderar la información de esta entrada y de la entrada anterior. Estas estructuras muchas veces se reinterpretan como puerta de entrada, puerta de salida y puerta de olvido. Hay arquitecturas simplificadas herederas de las ideas de estas.

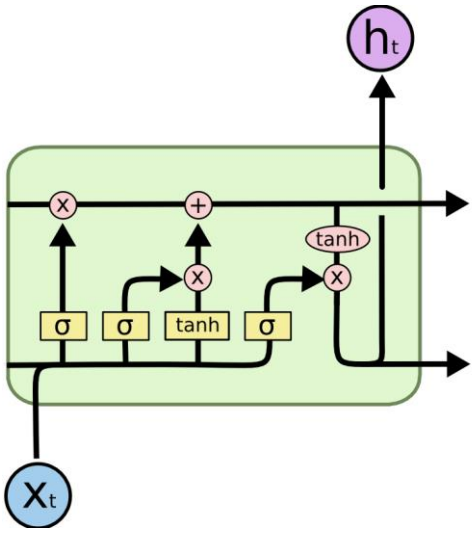
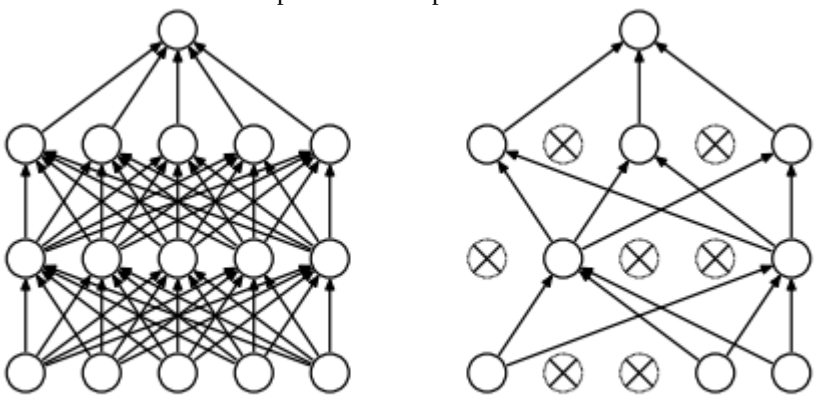


Ilustración 1Arquitectura LSTM

3.4 Dropout

Básicamente el dropout consiste en mantener encendidas o apagar las neuronas con una probabilidad haciendo que las neuronas que están encendidas se actualicen y las apagadas ni propaguen hacia adelante ni actualicen con la propagación hacia atrás, lo que conseguimos con este proceso es acelerar el aprendizaje y reducir el sobreajuste de nuestro entrenamiento, las razones por las que esto funciona no están muy claras, pero se ha demostrado de forma empírica en múltiples ocasiones.



(a) Standard Neural Net (b) After applying dropout.

Ilustración 2 Dropout

3.5 CÓDIGO

```
from keras.layers.core import Dense, Activation, Dropout
```

De keras importamos la capa densa, la capa de activación y la de dropout.
La capa densa nos proporciona la posibilidad de crear una capa cuyas entradas (de cada neurona) sean todas las salidas de la capa anterior.
La capa de activación permite añadir una capa con una función de activación.
La capa de dropout nos permite generar una capa de dropout (de vez en cuando no realizará la forward prop y la back prop) para incrementar la velocidad de entrenamiento.

```
from keras.layers.recurrent import LSTM
```

Importamos la capa LSTM ya que nos proporciona el soporte necesario para poder aprender datos en una secuencia periódica.

```
from keras.models import Sequential
```
Importamos el modelo secuencial (fuerza a que se ejecuten las capas una detrás de otra).

```
import time
```

Importamos time, para medir los tiempos de entrenamiento.

```
import matplotlib.pyplot as plt
```

Importamos matplotlib para poder dibujar las gráficas.

Función que utilizamos para cargar los datos, le pasamos el archivo, el tamaño de la secuencia y hace una ventana de la longitud que le digamos, separa los datos en entrenamiento y validación.

```
def load_data(filename, seq_len, normalise_window):
    #. . .
```

Esta función lo que nos permite es predecir los valores futuros, le pasamos el modelo, los datos, tamaño de ventana, y los x valores que vamos a predecir sin entrada.

```
def predict_sequences_multiple(model, data, window_size,
prediction_len):
    # . . .
```

Función que nos dibuja los resultados.

```
def plot_results_multiple(predicted_data, true_data, pre-
diction_len):
    # . . .
```

tamaño de secuencia de entrada

```
seq_len=100
#Step 1 Load Data
```

Separamos los datos, unos para entrenarlos y otros para la validacion de los resultados

```
X_train, y_train, X_test, y_test =
load_data('googl5.csv', seq_len, True)
```

Step 2 Build Model

El modelo será secuencial

```
model = Sequential()
```

Con una capa LSTM con Dropout del 20%

```
model.add(LSTM(input_dim=1, output_dim=50, return_sequen-
ces=True))
model.add(Dropout(0.2))
```

Otra capa LSTM con Dropout del 20% también

```
model.add(LSTM(100, return_sequences=False))
model.add(Dropout(0.2))
```

Una capa totalmente conectada con una función de activación lineal

```
model.add(Dense(output_dim=1))
model.add(Activation('linear'))
```

Compilamos el modelo e imprimimos cuanto tarda en hacerlo

```
start = time.time()
model.compile(loss='mse', optimizer='rmsprop')
print('compilation time : ', time.time() - start)
```

Entrenamos el modelo dándole una parte de los datos como validación de manera que haya una parte oculta de los datos que usaremos para test.

```
#Step 3 Train the model
model.fit(X_train, y_train, batch_size=512, nb_epoch=150,
validation_split=0.05)
#Step 4 - Plot the predictions!
predictions = predict_sequences_multiple(model, X_test,
seq_len, seq_len)
```

imprimimos la gráfica.

```
plot_results_multiple(predictions, y_test, seq_len)
```

4 Resultados experimentales

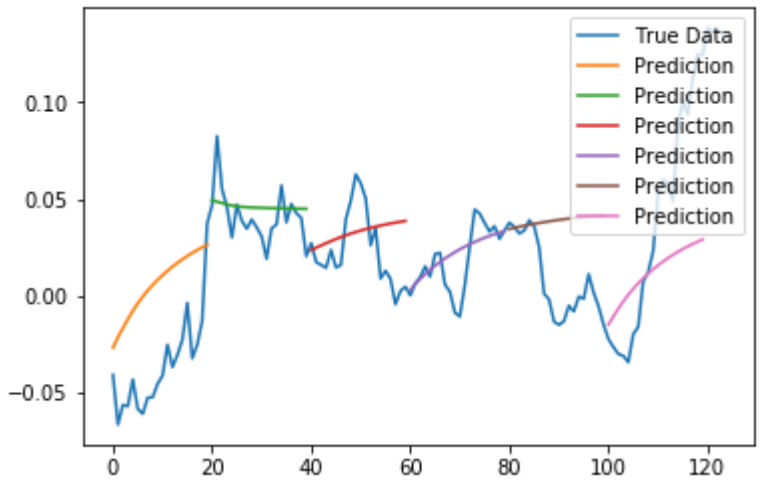
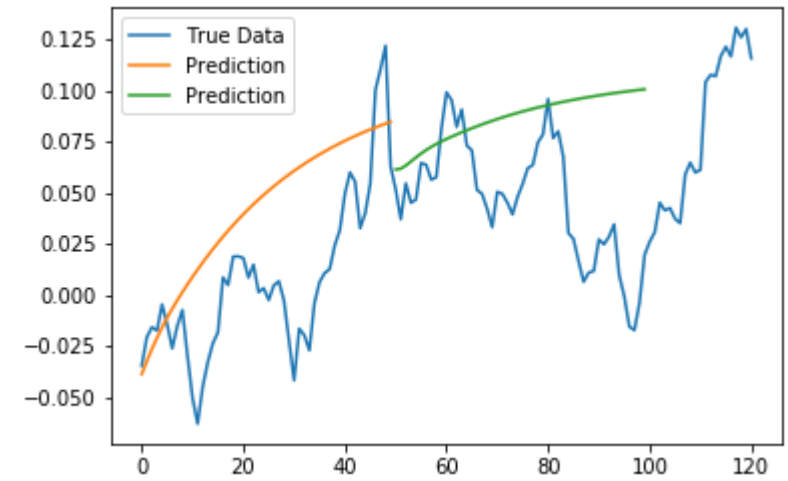


Ilustración 3 Entrenamiento con epoch 500, batch_size 512, validation_split 0.05, longitud de secuencia 20, , normalización Si.

Podemos ver en azul los datos reales, y las demás líneas son los datos predichos. Comprobamos que predice más o menos, las tendencias de cada periodo, aunque, esto podría ser un caso de sobreajuste, lo dudamos, ya que, se han intentado hacer entrenamientos más intensivos y alargados y la ha sido imposible predecir las tendencias. En general hemos encontrado un problema con este método de predicción porque no parece capaz de cambiar la tendencia una vez empieza a predecir en ningún caso hemos sido capaces de hacer que un entrenamiento resultase en un cambio de tendencia en las predicciones: nunca ha pasado una predicción de crecer a decrecer o viceversa.



.

Ilustración 4 Entrenamiento con epoch 500, batch_size 512, validation_split 0.05, longitud de secuencia 50, normalización Si.

Este es otro buen entrenamiento, vemos que las predicciones si aciertan bastante la tendencia, pero si hay crecidas o decrecidas muy pronunciadas le cuesta estabilizar la predicción, vemos que intenta corregirse, pero no puede con esos cambios tan bruscos.

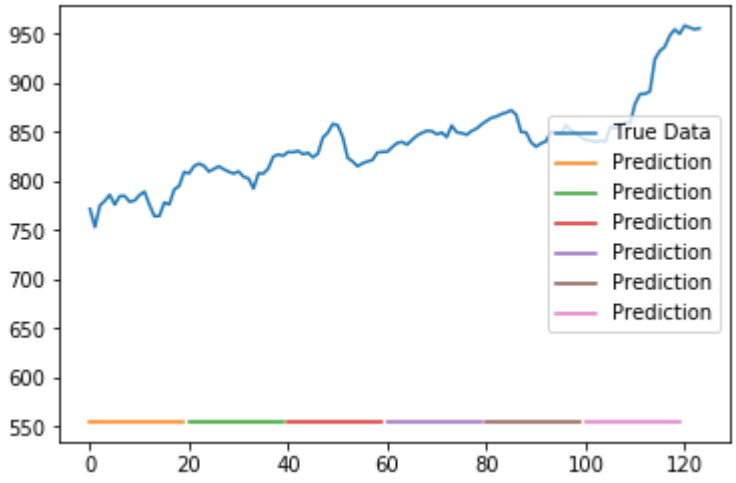


Ilustración 5 Entrenamiento con epoch 3500, batch_size 512, validation_split 0.05, longitud de secuencia 20, normalización No.

Esto simplemente fue una prueba para ver si quitando la normalización la red era capaz de ajustarse mejor a los datos, se ha comprobado que no se ajusta bien a los datos. Tras unas cuantas horas entrenando se estancó el ajuste y se quedó como se puede apreciar en la gráfica. Esto hace que el entrenamiento sin normalización sea inevitable ya que lleva mucho más tiempo y es mucho menos efectiva.

5 Conclusiones

Viendo los resultados obtenidos en las pruebas realizadas anteriormente, podemos decir, que en algunos casos predice la tendencia hasta cierto punto, pero esto no quiere decir que podamos empezar a invertir de forma segura, ya que en cambios bruscos le cuesta enormemente cambiar de tendencia una vez decide hacia qué sentido va a ir, y no suele acertar en la tendencia con suficiente certeza como para poder invertir.

Parece capaz de predecir la bolsa siempre y cuando tengamos muchos más datos y más continuados (no solo un dato al día). Se recomienda entrenar este tipo de redes con datos mucho más precisos, por ejemplo, todos los datos de un par de meses de manera continua (Google Finance les proporciona del último día por lo que en un futuro se podría conseguir esta información).

6 Bibliografia

1. LSTM (teoría y práctica): **http://colah.github.io/posts/2015-08-Understanding-LSTMs/**
2. RNN: **http://karpathy.github.io/2015/05/21/rnn-effectiveness/**
3. Código: **https://github.com/llSourcell/How-to-Predict-Stock-Prices-Easily-Demo/blob/master/stockdemo.ipynb**
4. Fuente base: **https://www.youtube.com/watch?v=ftMq5ps503w**
5. Fuente base: **https://www.youtube.com/watch?v=SSu00IRRraY&t=3s**