Topics in Biostat Final Report: Use CNN to Predict Stock Returns

Li Xu

May 6, 2018

1 Introduction

In this project, we try to use the price chart of stocks to predict the next day's return. For data collecting, we developed a reliable software to scrap the data from financial website and plot the stock historical chart automatically. We use con convolutional neural network (CNN) and fully connected neural network to build our deep learning structure. We perform a two stage prediction. One is binary classification that we predict whether the price will increase or decrease. The other is to predict the next-day return of a stock (continues output).

2 Data Description and Collecting

First step of this project is to define our input and output of our model. We wants to use the previous 30 days data of one stock to predict the return on 31 days. Let P_1 be the close price on the first day and P_{31} be the close price on 31st day. The return R we want to predict is calculated by:

$$R = \frac{P_{31} - P_1}{P_1} \tag{1}$$

For our input, we wants to use the image. Stock chart is good visualization tools to display the change of history. Since CNN gives us a change to directly use image for input. We use the chart built by 30-days stock price as our model input. An example stock chart is shown in Figure (1).

We randomly selected 200 stocks from the Standard & Poor's 500 index(SP500) as our data source. We use the data after 2016-01-01. To get enough data, we

further assume the stocks in SP500 are "identical". That means each stocks has similar behavior and they can be combined to build our training set. For example, we have one chart from AAPL and one chart form ABT, they are treated as two points in our training set. In the above assumptions, we totally got **3250** images.

We use quandl package for retrieving data and plotly for plotting charts. We encounter some issues when scraping data and making plot. If you request data too frequent (for example, use loop to download data), the remote server will refuse your request for a while. So we set a 3 second pause after we draw each plot.

3 Data Preprocessing

Before we train our model, we need to do some data preprocessing. For the resolution of image, we make them to be 500×500. We assume all stocks are "identical" so we remove the caption of chart. Another issue is the color of the image. The original image in Figure (1) is colorful. So if you read the image in python, it should be a $(500, 500, 4)$ tensor. This is too big considering we have 3250 images. Converting them to black and white seems we reduce the information too much. The color in image should makes sense. Eventually, we transfer the image to gray scale. One example for same chart before and after gray scaled is provided in Figure (2). The scaled one looks yellow because I did not specify the color space when printing. If you specify the color space to be gray, if should looks normal. After preprocessing, our model input should be a 500×500 matrix.

We also split the data into training set and testing set. Since we did not set valid set because our sample size is relatively small.

4 Deep Learning Neural Network Structure

4.1 Binary Prediction

First we predict given a chart built by the previous 30-day price, whether the stock will increase or decrease. The output is binary and we let "0" means decrease and "1" means increase. I use a model with CNN and fully connected layers. The code for build the architecture is in Figure (3). The training output is in Figure (4) The confusion matrix are also provided in Figure (5). We use categorical hinge for the loss function.

From the confusion matrix, we see that CNN does not have the power to classify the stock increase on our current dataset.

4.2 Continues Prediction

The second stage of our prediction is to predict the stock return on 31st days. We use the same structure in Chapter 3 but change the output layer to be continues output. The model code is in Figure (6). The training output is in Figure (7). Again, the model fails at predicting the stock price.

5 Results and Summary

In this project, we fail to make any meaningful prediction for the stocking price. Some reason may be concluded. First, our training sample is too limited. In the mnist examples. The training set is 60000 which is much larger than ours. Due to the limit time, we are not able to produce more dataset. We must pause between each data requesting to avoid being prohibited for some time due to the API's regulation. This greatly affects our data collecting speed.

Second, our deep learning structure is too simple. Stocking price is famous for its difficulty to be predicted. So we can image the latent architecture should be complex. Due our limit computing power, we are not able to train more complex model. We use a GTX1070 on my desktop for training and it is easy to get a memory overflow error.

Last but not lease, we create a framework for doing prediction on stock price. If giving more time, we may get better results.

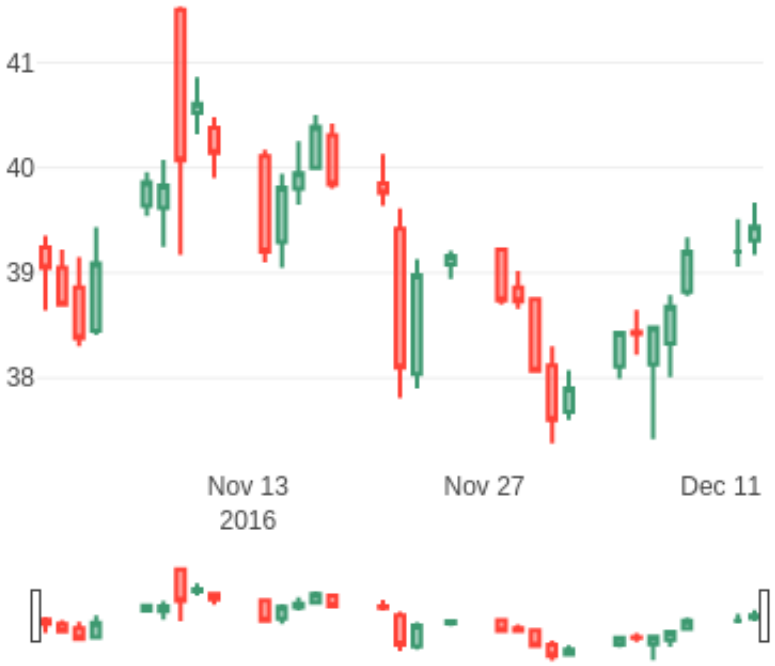

Figure 1: Example stock chart built from Abbott Laboratories(ABT)'s historical price.

Figure 2: Example of Gray Scale, Upper is the original chart and below is the scaled one.

```
model = Sequential()
model.add(Conv2D(64, kernel_size=(10, 10),strides=[5,5],
                 activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(Conv2D(64, (3, 3), activation='relu'))
#model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.compile(#loss=keras.losses.categorical_crossentropy,
              loss=keras.losses.categorical_hinge,
              optimizer=keras.optimizers.Adadelta(),
              #optimizer='sgd',
              metrics=['accuracy'])
```

```
batch_size = 128
num_classes = 2
epochs = 5

# input image dimensions
img_rows, img_cols = 500, 500

model.fit(X_train, y_train_Binary,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(X_test, y_test_Binary))
score = model.evaluate(X_test, y_test_Binary, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

Figure 3: Model code for Binary Output.

```
Train on 2925 samples, validate on 325 samples
Epoch 1/5
2925/2925 [==============================] - 10s 4ms/step - loss: 1.0408 - acc: 0.4793 - val_loss: 1.1385 - val_ac
c: 0.4308
Epoch 2/5
2925/2925 [==============================] - 10s 3ms/step - loss: 1.0359 - acc: 0.4821 - val_loss: 1.1385 - val_ac
c: 0.4308
Epoch 3/5
2925/2925 [==============================] - 10s 3ms/step - loss: 0.9842 - acc: 0.5080 - val_loss: 1.1385 - val_ac
c: 0.4308
Epoch 4/5
2925/2925 [==============================] - 10s 3ms/step - loss: 1.0591 - acc: 0.4704 - val_loss: 1.1385 - val_ac
c: 0.4308
Epoch 5/5
2925/2925 [==============================] - 10s 3ms/step - loss: 1.0441 - acc: 0.4779 - val_loss: 1.1385 - val_ac
c: 0.4308
Test loss: 1.138461538553238
Test accuracy: 0.4307692309526297
```

Figure 4: Training Output for Binary Output.

```
In [18]: #output predicted value
         Return_pre=model.predict_classes(X_test, verbose=1)
         y_test_nn=np.sign(y_test)
         y_test_nn[y_test_nn==-1]=0
         from sklearn.metrics import confusion_matrix
         confusion_matrix = confusion_matrix(y_test_nn, Return_pre)
         confusion_matrix

         325/325 [==============================] - 0s 1ms/step

Out[18]: array([[140,   0],
                [185,   0]])
```

Figure 5: Confusion Matrix for Binary Output.

```
model = Sequential()
model.add(Conv2D(64, kernel_size=(10, 10),strides=[5,5],
                 activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(Conv2D(64, (3, 3), activation='relu'))
#model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(128, activation='relu'))

model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1,kernel_initializer='normal'))

model.compile(loss=keras.losses.mean_absolute_percentage_error,
              optimizer=keras.optimizers.Adadelta(),
              #optimizer='sgd',
              metrics=['accuracy'])

batch_size = 128
#num_classes = 2
epochs = 5

# input image dimensions
#img_rows, img_cols = 500, 500

model.fit(X_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(X_test, y_test))
score = model.evaluate(X_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

Figure 6: Model code for Continues Output.

```
Train on 2925 samples, validate on 325 samples
Epoch 1/5
2925/2925 [==============================] - 16s 5ms/step - loss: 255835437.7687 - acc: 6.8376e-04 - val_loss: 4738
9.6459 - val_acc: 0.0000e+00
Epoch 2/5
2925/2925 [==============================] - 10s 3ms/step - loss: 121009.8098 - acc: 0.0010 - val_loss: 108.8500 -
val_acc: 0.0000e+00
Epoch 3/5
2925/2925 [==============================] - 10s 3ms/step - loss: 3502.9016 - acc: 0.0010 - val_loss: 100.7330 - va
l_acc: 0.0000e+00
Epoch 4/5
2925/2925 [==============================] - 10s 3ms/step - loss: 546.6089 - acc: 0.0010 - val_loss: 100.7365 - val
_acc: 0.0000e+00
Epoch 5/5
2925/2925 [==============================] - 10s 3ms/step - loss: 1152.2370 - acc: 0.0010 - val_loss: 100.3505 - va
l_acc: 0.0000e+00
Test loss: 100.35051354041467
Test accuracy: 0.0
```

Figure 7: Training Output for Continues Output.

A Repo

All of our code is available online. The URL for the repo is here.