UROP 1100 Spring 2017
Final Report

Artificial Intelligence with Application to Finance

Professor: YANG Qiang
Instructor: LU zhongqi

Author:

HUANG Kung-Hsiang (20215699)
HU Yao-Chieh (20216239)

Abstract

As the development of the artificial intelligence geared toward the unlimited applications to various fields of recent life, the financial world is not an exception. This paper aims at exploiting the underlying pattern or rule of the stock data that generated daily in an irresistible speed. By the real stock data, the paper utilizes an advanced recurrent neural network named LSTM (Long Short Term Memory) neural network to classify the stock data into two groups. One with ascending triangle, a meaningful pattern hidden in stock data, and one without it. This paper will lead readers through the development of the machine learning model and the application to real stock data.

Motivation

No denying that predicting the financial trend can be an intriguing topic to work on, especially the stock data that is important for people to make a decent and profitable investment. To make a forecast for a rising trend, there is always a use of ascending triangle that implicitly indicates the possibilities. But, it is hard to make a clear and accurate definition of ascending triangle. And more than that, specialists' judging might not be the exact ascending triangle pattern in the shadow, which we would never get a chance to know it. Instead of placing our trust on human judgement that might cause bias, we decide to train a machine learning model on plenty of real data to let the computational device figures out the answer for us.

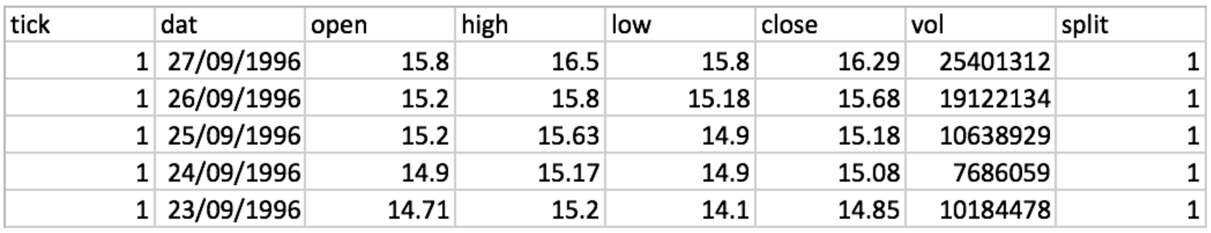
Development

Stage 1: Data Preparation

I. Dataset Introduction

There are around 788 million rows of data, each representing a single day of transaction in stock market for a specific equity. Locality existed for a range of data, standing for a period of time of transactions, that normally ranging from 20 days to 60 days, and has 50 days as average.

For each row of data, namely a day within a range of locality, the columns given consists of eight properties: tick, date (dat), open price (open), high price (high), low price (low), close price (close), volume (vol) and split. A set of rough data rows are displayed as Figure 1.



tick	dat	open	high	low	close	vol	split
1	27/09/1996	15.8	16.5	15.8	16.29	25401312	1
1	26/09/1996	15.2	15.8	15.18	15.68	19122134	1
1	25/09/1996	15.2	15.63	14.9	15.18	10638929	1
1	24/09/1996	14.9	15.17	14.9	15.08	7686059	1
1	23/09/1996	14.71	15.2	14.1	14.85	10184478	1

Figure 1. Raw dataset display in Microsoft Excel

II. Dataset Preprocessing

Considering the ascending triangle property in the preprocessing phase, there are only part of the properties are required for the machine learning training. The open price, close price, high price and low price properties of the raw dataset are kept for training preparation, along with the date of transaction.

III. Dataset Size Decision

The total size of given data is around 788 million rows of data, which is quite heavy to train in bulk for our machine. Taking this into consideration, the bulk dataset is splited into sets with smaller size, each of which contains 100 thousand rows of data.

IV. Dataset Segmentation

Owing to that the valid date ranges in localities are not consistent, mostly ranging from 20 days to 60 days, there are two feasible methods to handle the data segmentation:

Method 1: Segment by original date-range: This method generates fundamental unit data whose sizes are inconsistent, simply corresponding to its original size.

Method 2: Segment by specified date-range: This method will constraint the sizes of fundamental unit data to be consistent, yet it will result in the data discarding/loss, where the out-ranged data is rid of.

V. Dataset Visualization

Libraries used to support the visualization: matplotlib, numpy.

With the help of matplotlib.finance.candlestick_ohlc(), an public function of matplotlib's finance module, a fundamental unit data could be visualized as Figure 2.

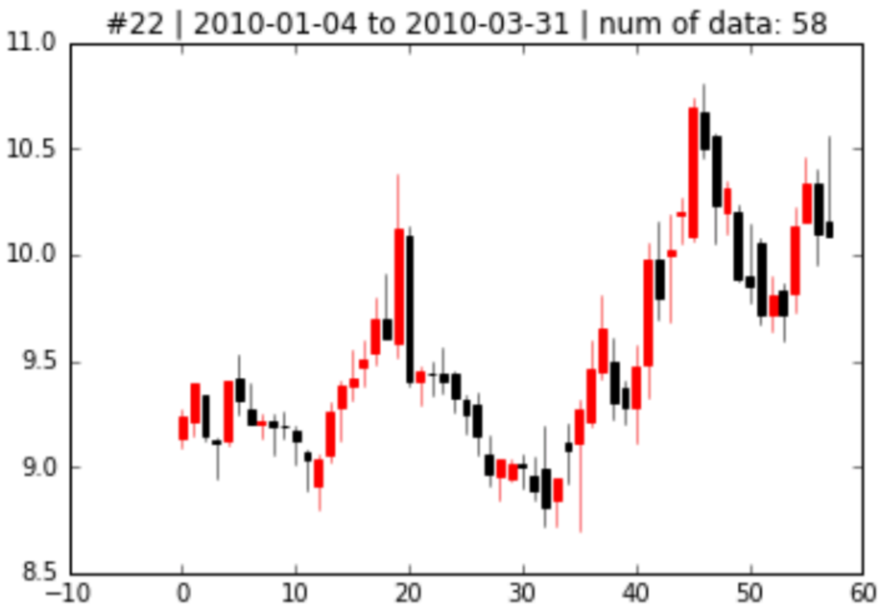


Figure 2. The visualization of stock data in a locality, segmented by original date-range

VI. Pattern Labeling (Ascending Triangle)

The labelling is currently performed by manual judgement. This might potentially lead to subjective judgements on the ascending triangle features. To solve the issue, it has been considered to ask helps from professionals who could provide more objective recognition of the pattern of ascending triangles, such as financial specialists.

Two examples are shown below to demonstrate the labeling of ascending triangle with a labeled example in Figure 3 and a non-labeled example in Figure 4:

Labeled Example: Ascending Triangle Pattern Detected (labeled as 1)

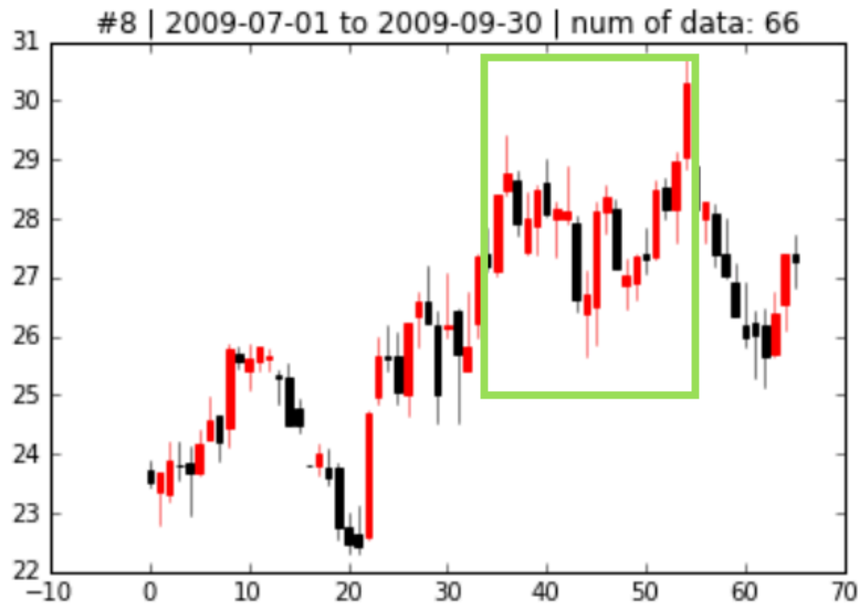


Figure 3. The labeled example with an ascending triangle pattern in the green square area.

Non-labeled example: No Pattern Detected (labeled as 0)

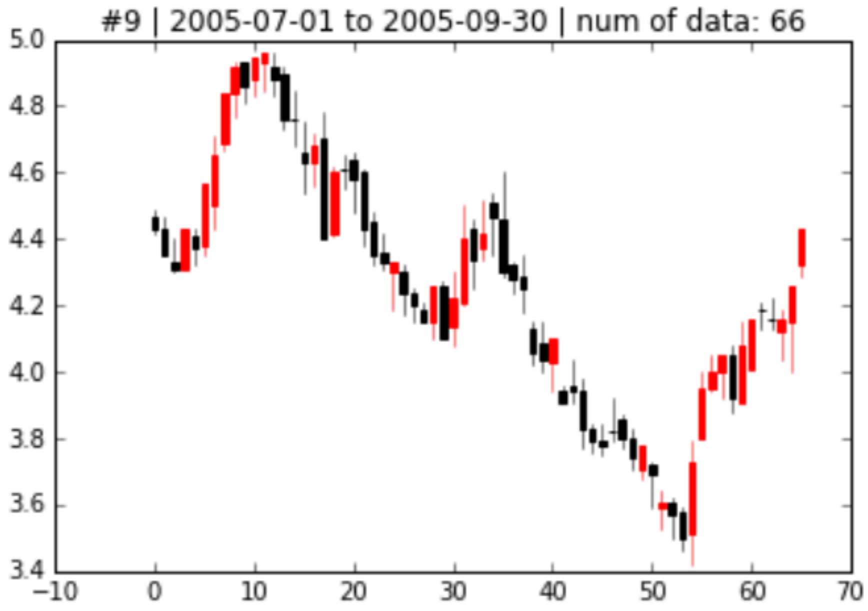


Figure 4. The non-labeled example without any pattern detected.

VII. Enhancement on Segmentation

Given the fact that the data for ascending triangle is not enough as less than 50 sets of data were selected out of the 1135 segments of data, a better way to collect data is the sliding-window approach, where the window size is default to be 25 days. With this data-collection method, it is possible to label and collect more data in an efficient manner.

Method 3: Sliding-Window Approach: For a set of data within a period of time, about 50 days in average, rather than collecting a single valid range of data, 25 days, from this set, the window can be slided rightward until the end of the range, collecting more valid data along the slides. For instance, a set of data contains 50 days. By the original approach, there is only one data generated as input for the later machine learning trainer; but with the implementation of sliding-window approach, (50 - 25) + 1 = 26 valid data can be gathered as the inputs for the trainer.

Stage 2: Machine Learning Model

This stage shows the design and implementation of RNN Pre-Training Model for ascending triangle pattern recognition and further classification.

I. Model Overview

The model is composed of one input layer, one output layer, and one hidden layer as shown in the figure below. The hidden layer is composed of 6 LSTM cells. On top of these hidden nodes, there is a sigmoid function that maps the weighted sum of output from these hidden nodes to a value between 0 and 1. The concept illustration is shown as Figure 5.

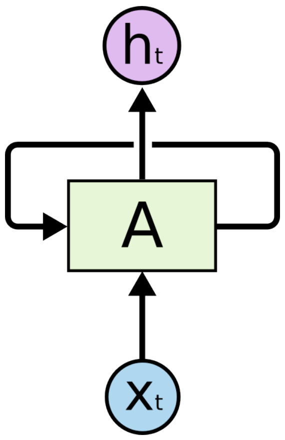


Figure 5. The conceptual structure of a LSTM cell in RNN

II. Data Explanation

Input data X and output data Y are composed of 24 data points, each is represented by the closed price of a stock in a day. Y is just an offset of X by 1 day, which means that each Y is a prediction of the next X. The illustration is shown in Figure 6 on the next page.

Before feeding data into layers, each bunch of input data is normalized so that the range of data lies between 0 and 1. Zt is generated by the sigmoid of the sum of all weighted $h_{ti,}$ where t represents step number and i represents state number. The sigmoid function is as Figure 7.

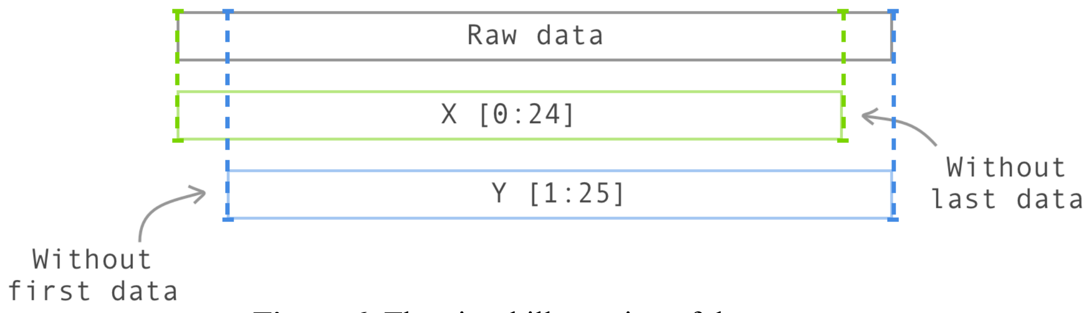


Figure 6. The visual illustration of data usage

$$Z_t = sigmoid(\sum_i H_{ti} \times W_{ti})$$

Figure 7. The sigmoid function's formula implemented for data normalization

Loss Function: The loss function is defined by the L2-norm between Zt and Yt, where t is the step number, shown as figure 8.

$$Loss = \sum_t \sqrt{(Z_t - Y_t)^2}$$

Figure 8. The L2-norm loss function

Final Step: Before fed into the classifier in the next step, all the input data, regardless of their labels, are delivered into the trained RNN model. The hidden state in each LSTM cell will be extracted, which will be the input data during the classification.

III. Classification:

Linear Regression: Linear Regression is one approach of classification that tries to predict the probability of each instance. The logistic function is defined as figure 9.

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

Figure 9. The definition of linear regression for classification

In the formula, θ represents weights, while x stands for input data. hθ(x) estimates the probability that y=1 given x and θ, namely p(y = 1 |x; θ).

It is not feasible to use the squared loss as linear regression since the logistic function defined above might results in a non-convex optimization. Therefore, a better way to define loss function is to take log and sum together because taking log does not affect the relative ordering of the number. Shown as figure 10.

$$\arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{n} \log p(y^{(i)} \mid \boldsymbol{x}^{(i)}; \boldsymbol{\theta})$$

Figure 10. The improved loss function avoiding non-convex situation

Support Vector Machine (SVM): Support Vector Machine is another machine learning method that maximizes the margin between two decision hyperplanes. The explanation is shown in figure 11.

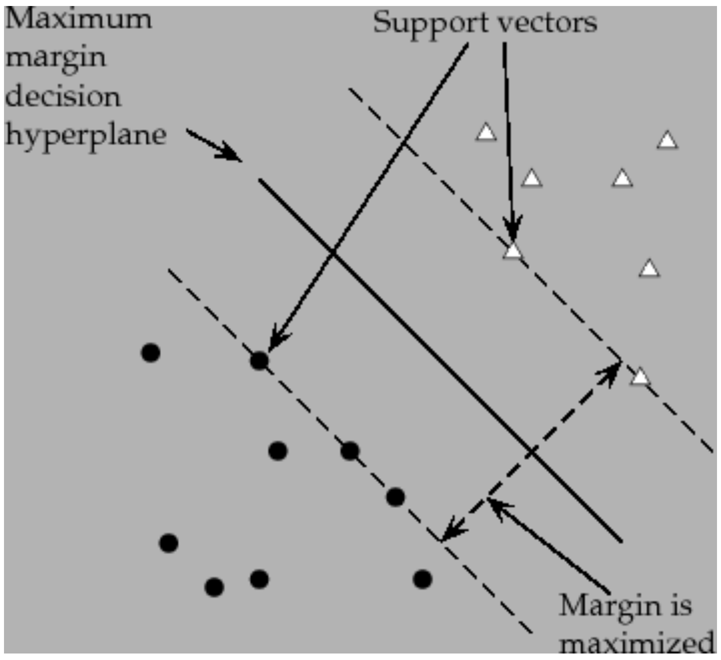


Figure 11. The explanation of support vector machine (SVM)

It introduces a hinge loss function that is very similar to the one used in logistic regression, except that y = 1 when θTx >0 and y= 0 when θTx <0. The hinge loss function is displayed in figure 12.

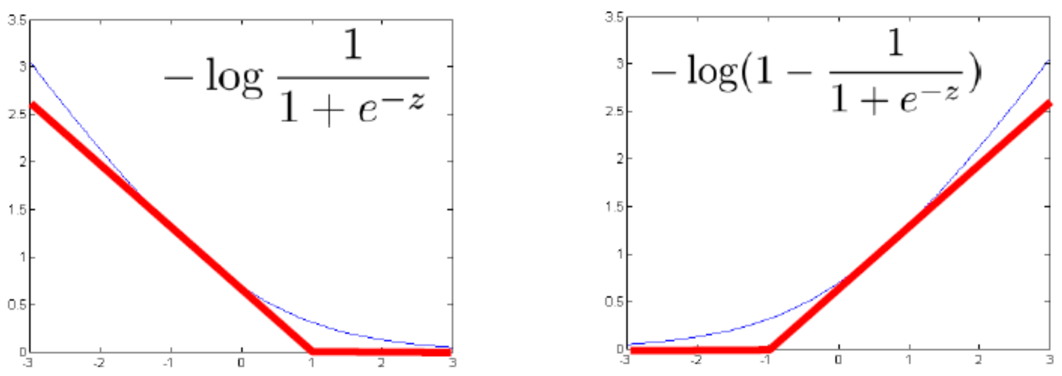


Figure 12. The hinge loss function

As a result, the objective function will be minimizing the following (figure 13):

$$\min_{\boldsymbol{\theta}} \; C\sum_{i=1}^{n}\left[y_i \mathrm{cost}_1(\boldsymbol{\theta}^\intercal \mathbf{x}_i) + (1-y_i)\,\mathrm{cost}_0(\boldsymbol{\theta}^\intercal \mathbf{x}_i)\right] + \frac{1}{2}\sum_{j=1}^{d}\theta_j^2$$

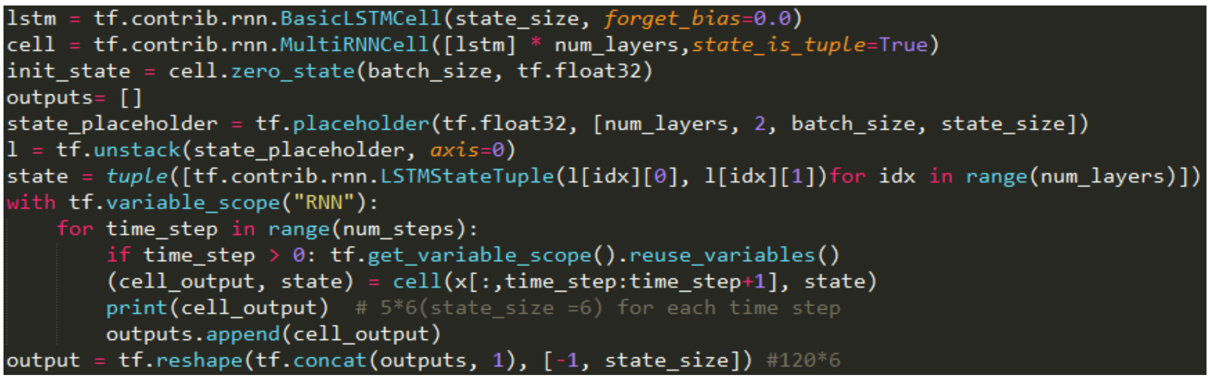
y = 1 / 0

Figure 13. The total loss function to be minimized

IV. Implementation

The codes developed in real are shown below as reference.

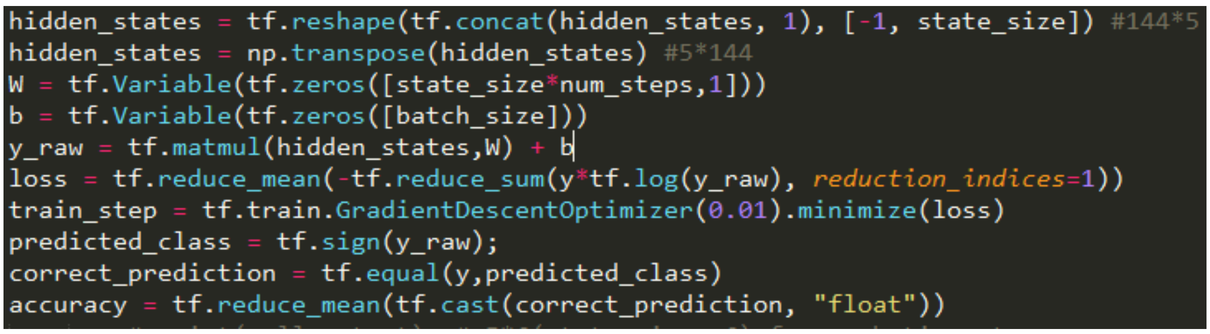
(1) RNN



```
lstm = tf.contrib.rnn.BasicLSTMCell(state_size, forget_bias=0.0)
cell = tf.contrib.rnn.MultiRNNCell([lstm] * num_layers,state_is_tuple=True)
init_state = cell.zero_state(batch_size, tf.float32)
outputs= []
state_placeholder = tf.placeholder(tf.float32, [num_layers, 2, batch_size, state_size])
l = tf.unstack(state_placeholder, axis=0)
state = tuple([tf.contrib.rnn.LSTMStateTuple(l[idx][0], l[idx][1])for idx in range(num_layers)])
with tf.variable_scope("RNN"):
    for time_step in range(num_steps):
        if time_step > 0: tf.get_variable_scope().reuse_variables()
        (cell_output, state) = cell(x[:,time_step:time_step+1], state)
        print(cell_output)  # 5*6(state_size =6) for each time step
        outputs.append(cell_output)
output = tf.reshape(tf.concat(outputs, 1), [-1, state_size]) #120*6
```

Figure 14. The code of Recurrent Neural Network (RNN)

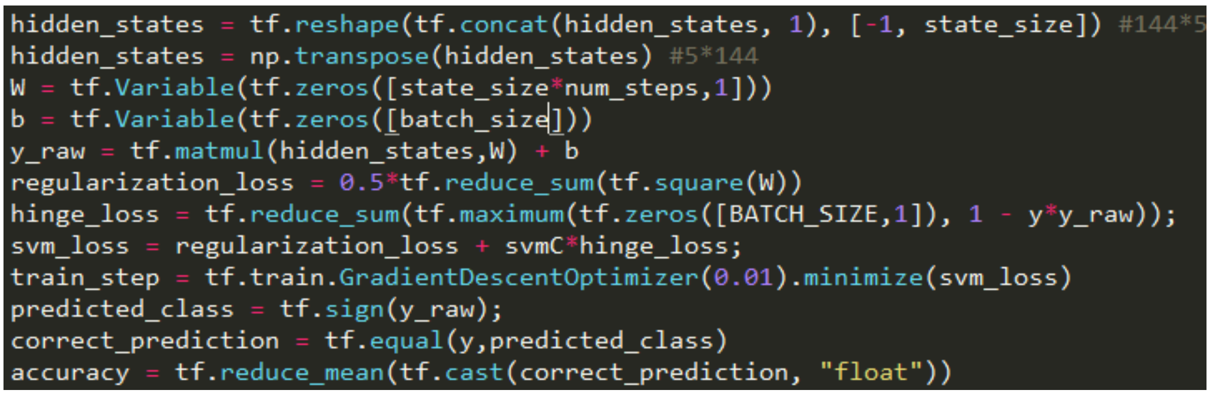
(2) Logistic Regression



```
hidden_states = tf.reshape(tf.concat(hidden_states, 1), [-1, state_size]) #144*5
hidden_states = np.transpose(hidden_states) #5*144
W = tf.Variable(tf.zeros([state_size*num_steps,1]))
b = tf.Variable(tf.zeros([batch_size]))
y_raw = tf.matmul(hidden_states,W) + b
loss = tf.reduce_mean(-tf.reduce_sum(y*tf.log(y_raw), reduction_indices=1))
train_step = tf.train.GradientDescentOptimizer(0.01).minimize(loss)
predicted_class = tf.sign(y_raw);
correct_prediction = tf.equal(y,predicted_class)
accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
```

Figure 15. The code of Logistic Regression

(3) SVM



```
hidden_states = tf.reshape(tf.concat(hidden_states, 1), [-1, state_size]) #144*5
hidden_states = np.transpose(hidden_states) #5*144
W = tf.Variable(tf.zeros([state_size*num_steps,1]))
b = tf.Variable(tf.zeros([batch_size]))
y_raw = tf.matmul(hidden_states,W) + b
regularization_loss = 0.5*tf.reduce_sum(tf.square(W))
hinge_loss = tf.reduce_sum(tf.maximum(tf.zeros([BATCH_SIZE,1]), 1 - y*y_raw));
svm_loss = regularization_loss + svmC*hinge_loss;
train_step = tf.train.GradientDescentOptimizer(0.01).minimize(svm_loss)
predicted_class = tf.sign(y_raw);
correct_prediction = tf.equal(y,predicted_class)
accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
```

Figure 16. The code of Support Vector Machine (SVM)

More Information

The full implementation of our machine model can be found in this Github repository.

https://github.com/UROP2017Spring/AscendingTriangleClassification

Contribution of Team Memebers

HU, Yao-chieh	Data Processing, Logistic Regression	50%
Huang, Kung-hsiang	RNN Pre-Training Model, SVM	50%

Reference

Ng, Andrew. "Machine Learning | Coursera". Coursera. N.p.