Vietnam National University Ho Chi Minh City
University of Science
Faculty of Information Technology
Module: Advanced Topics in Software Development Technology

Trading Rule Identification by CNN

Author: Group 9
20120454 - Lê Công Đắt
20120489 - Võ Phi Hùng
20120558 - Lưu Ngọc Quang
20120582 - Trần Hữu Thành

Instructor:
M.S. Trần Văn Quý
M.S. Trần Duy Quang
M.S. Đỗ Nguyên Kha

Table of Contents

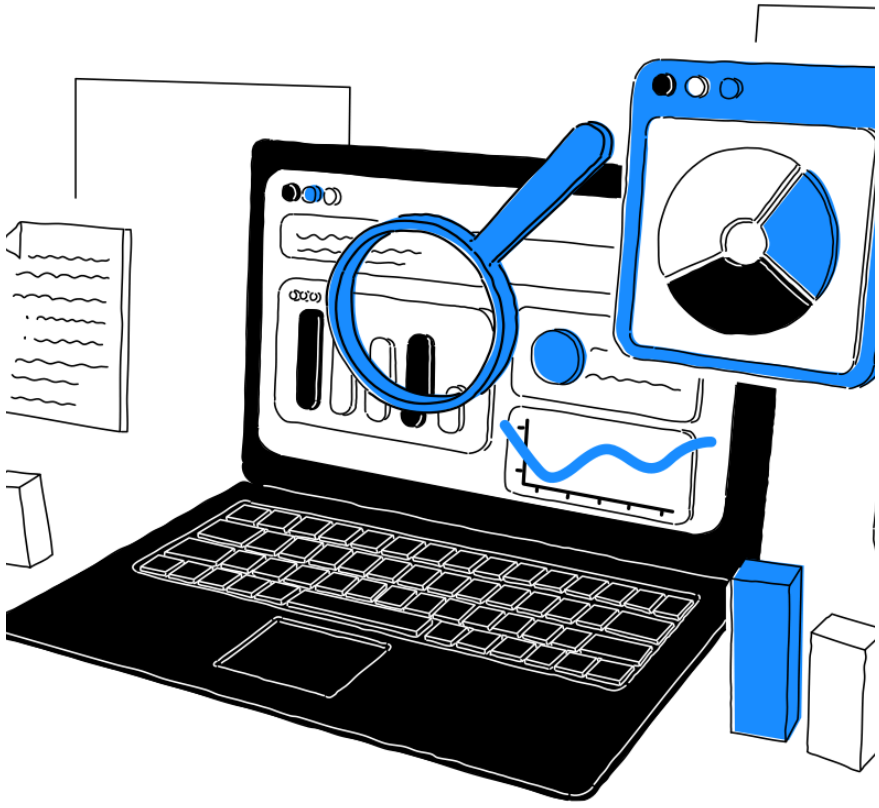
Introduction

Introduction

Trading signals with technical indicators

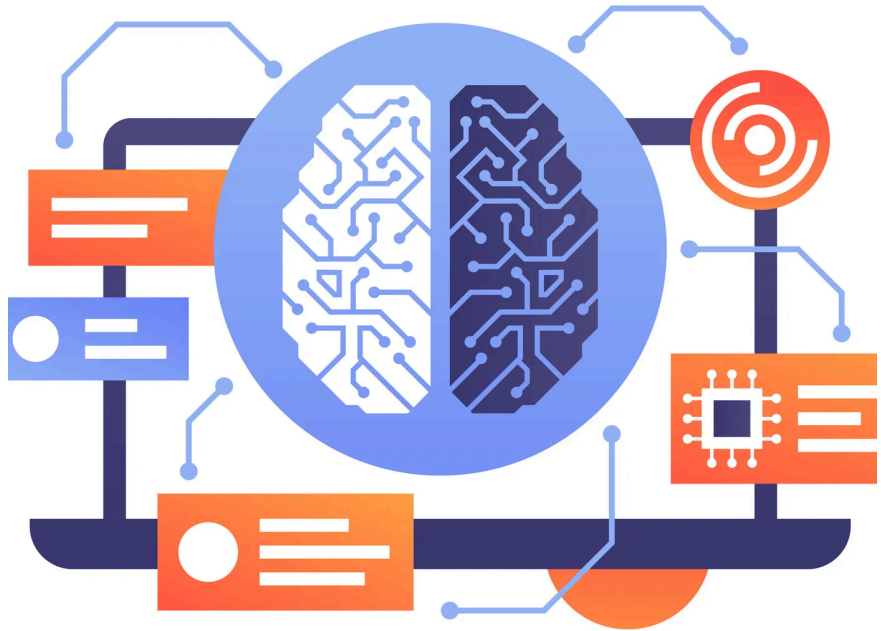
Trading signals with technical indicators

Data handling



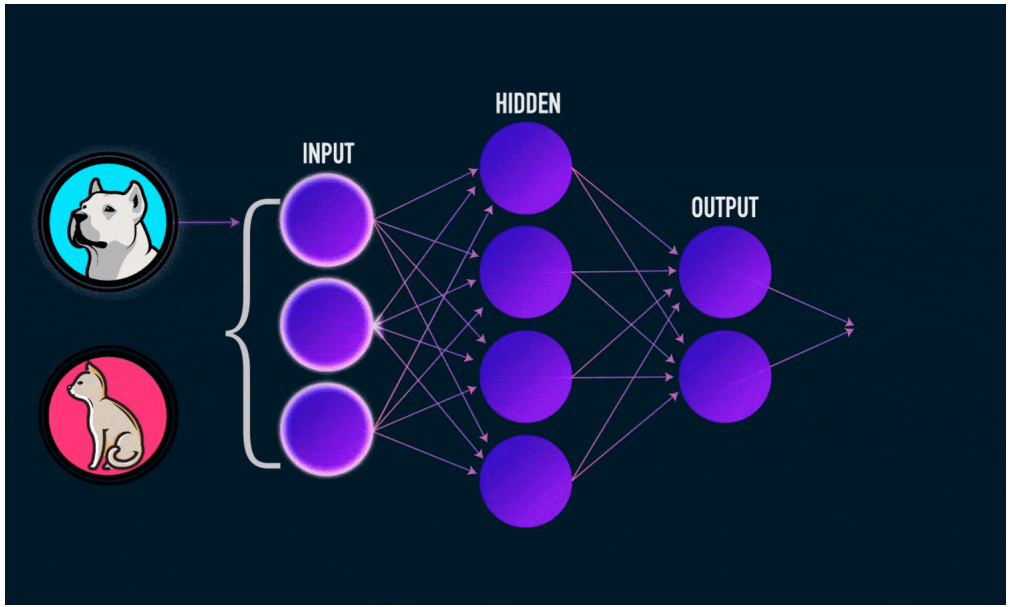
Data handling

Benchmarking alternative models



Benchmarking alternative models

Constructing a convolutional neural network



Human Decision Process in Trading

- Filter noise by observing multiple data points
- Confidence builds with consistent price trends
- Compare new data to historical patterns
- Act if patterns match confidently
- CNNs automate this decision-making
- Apply across multiple assets

Modeling Investment Logic

- Optimal observation period before trading
- Use moving-average data over 12 days
- Take most frequent action
- Hold if multiple crossovers or confusion
- If 8 out of 12 actions are to buy, consider buying
- Identify patterns to validate actions

Selecting the Network Architecture

Selecting the appropriate architecture depends on the input data:

- Input defined as a tensor to capture temporal features
- Utilize multiple data points to capture volatility
- Convolutional layer size depends on data analysis depth
- Use bigger convolutional windows for capturing moving averages

Architecture selection is an art rather than a science:

Heuristics can guide rather than dictate the process

CNN Architecture

- An input layer: 12 x 3 tensor representing 12 observations of 3 features
- A convolutional layer: Convolution to a space of a 6 x 6 tensor to capture patterns
- A pooling layer to flatten the data
- An output layer with 3 classes

Setting up the data in the correct format

Explanation of Data Segmentation

- Data is segmented into smaller segments.
- Each segment typically contains 12 observations.
- Overlapping windows capture patterns across data points.
- Segmented data is converted to NumPy arrays for CNN input.
- Arrays are saved in h5 files for efficient retrieval.
- Utility functions aid TensorFlow code implementation.

Setting up the data in the correct format

Split the data into windows of the required time horizon

```
def windows(data, size):
    # Hàm này tạo ra các cửa sổ (windows) từ dữ liệu với kích thước được định nghĩa bởi 'size'.
    start = 0  # Khởi tạo biến bắt đầu tại vị trí 0
    while start < data.count():  # Tiếp tục tạo cửa sổ cho đến khi vị trí bắt đầu vượt quá số lượng phần tử trong dữ liệu
        yield int(start), int(start + size)  # Trả về một cặp giá trị (start, start + size) đại diện cho cửa sổ dữ liệu hiện tại
        start += (size / 2)  # Di chuyển vị trí bắt đầu thêm nửa kích thước cửa sổ để tạo ra cửa sổ mới (các cửa sổ sẽ chồng lấn lé
```

Setting up the data in the correct format

Segment the data into a signal and its corresponding label

```
def segment_signal(data,window_size = 12):
    segments = np.empty((0,window_size,6))
    labels = np.empty((0))
    for (start, end) in windows(data['Date'], window_size):
        x = data["mav5"][start:end]
        y = data["mav10"][start:end]
        z = data["mav20"][start:end]
        a = data["mav30"][start:end]
        b = data["mav50"][start:end]
        c = data["mav100"][start:end]
        if(len(data['Date'][start:end]) == window_size):
            segments = np.vstack([segments,np.dstack([x,y,z,a,b,c])])
            unique_labels, counts = np.unique(data["Action"][start:end], return_counts=True)
            most_frequent_label = unique_labels[np.argmax(counts)]
            labels = np.append(labels, most_frequent_label)
    return segments, labels
```

Setting up the data in the correct format

Create batches from these segments that can be used to train our model

```
def get_batches(X, y, batch_size=100):
    """ Trả về một generator cho các batch """

    # Tính số lượng batch có thể tạo ra
    n_batches = len(X) // batch_size

    # Cắt bớt dữ liệu X và y để phù hợp với số lượng batch
    X, y = X[:n_batches*batch_size], y[:n_batches*batch_size]

    # Lặp qua các batch và trả về từng batch
    for b in range(0, len(X), batch_size):
        yield X[b:b+batch_size], y[b:b+batch_size]

# Mạng nơ-ron được huấn luyện theo từng batch để tránh tình trạng tràn bộ nhớ.
```

Setting up the data in the correct format

Create the training and test data in a format that can be used by TensorFlow

```
def create_tensorflow_train_data(csvfilename):
    df = pd.read_csv('sampledata/'+ csvfilename)
    df = df[['Date','symbolid','buyret','sellret','Action','mav5', 'mav10','mav20','mav30','mav50','mav100']]
    df['Action'].fillna('None', inplace=True)
    symbols = df.symbolid.unique()
    segments, labels = segment_signal(df[df.symbolid == symbols[0]])
    df = df[df.symbolid != symbols[0]]
    symbols = symbols[1:]
    for i in range(0,len(symbols)):
        x, a = segment_signal(df[df.symbolid == symbols[i]])
        segments = np.concatenate((segments, x), axis = 0)
        labels = np.concatenate((labels, a), axis = 0)
        df = df[df.symbolid != symbols[i]]
        print(str(round(i/len(symbols)*100,2)) + ' percent done')
    list_ch_train = pd.get_dummies(labels)
    list_ch_train = np.asarray(list_ch_train.columns)
    labels = np.asarray(pd.get_dummies(labels), dtype = np.int8)
    X_tr, X_vld, lab_tr, lab_vld = train_test_split(segments, labels, stratify = labels, random_state = 123)
```


Setting up the data in the correct format

Create the training and test data in a format that can be used by TensorFlow

```
def create_tensorflow_test_data(csvfilename):
    df = pd.read_csv('sampledata/'+ csvfilename)
    df = df[['Date','symbolid','buyret','sellret','Action','mav5', 'mav10','mav20','mav30','mav50','mav100']]
    df['Action'].fillna('None', inplace=True)
    list_ch_test = df.Action.unique()
    symbols = df.symbolid.unique()
    segments, labels = segment_signal(df[df.symbolid == symbols[0]])
    df = df[df.symbolid != symbols[0]]
    symbols = symbols[1:]
    for i in range(0,len(symbols)):
        x, a = segment_signal(df[df.symbolid == symbols[i]])
        segments = np.concatenate((segments, x), axis = 0)
        labels = np.concatenate((labels, a), axis = 0)
        df = df[df.symbolid != symbols[i]]
        print(str(round(i/len(symbols)*100,2)) + ' percent done')
    list_ch_test = pd.get_dummies(labels)
    list_ch_test = np.asarray(list_ch_test.columns)
    labels = np.asarray(pd.get_dummies(labels), dtype = np.int8)
    X_test = segments
```

Setting up the data in the correct format

The training and test data saved as h5 files

```
def savedataset(csvfilename, h5filename):
    dt = h5py.special_dtype(vlen=str)
    hf = h5py.File('h5files/'+ h5filename, 'w')
    for i in range(0,len(csvfilename)):
        csvfile = csvfilename[i]
        if(csvfile[:4] != 'test'):
            X_tr, X_vld, lab_tr, lab_vld, list_ch_train = create_tensorflow_train_data(csvfile)
            hf.create_dataset('X_tr',data=X_tr)
            hf.create_dataset('X_vld',data=X_vld)
            hf.create_dataset('lab_tr',data=lab_tr)
            hf.create_dataset('lab_vld',data=lab_vld)
            hf.create_dataset('list_ch_train',data=list_ch_train, dtype = dt)
            del(X_tr, X_vld, lab_tr, lab_vld, list_ch_train)
        else:
            X_test, y_test, list_ch_test = create_tensorflow_test_data(csvfile)
            hf.create_dataset('X_test', data = X_test)
            hf.create_dataset('y_test', data = y_test)
            hf.create_dataset('list_ch_test', data = list_ch_test, dtype = dt)
            del(X_test, y_test, list_ch_test)
```

Setting up the data in the correct format

The training and test data saved as h5 files

```
def get_tf_train_data(h5filename):

    hf = h5py.File('h5files/' + h5filename, 'r')
    X_tr = hf['X_tr'][:]
    X_vld = hf['X_vld'][:]
    lab_tr = hf['lab_tr'][:]
    lab_vld = hf['lab_vld'][:]
    list_ch_train = hf['list_ch_train'][:]
    hf.close()
    return X_tr, X_vld, lab_tr, lab_vld, list_ch_train

def get_tf_test_data(h5filename):

    hf = h5py.File('h5files/' + h5filename, 'r')
    X_test = hf['X_test'][:]
    y_test = hf['y_test'][:]
    list_ch_test = hf['list_ch_test'][:]

    return X_test, y_test, list_ch_test
```

Training and testing the model

Read the train data

```
traindtfile = 'hdf_50.h5'
testdtfile = 'hdf_50.h5'
losssavefig = 'cnn_train_50_loss.png'
accsavefig = 'cnn_train_50_accuracy.png'
resultsave = 'cnn_train_50.h5'
chkpointdir = 'cnn-50/'

X_tr, X_vld, y_tr, y_vld, list_ch_train = get_tf_train_data(traindtfile)
```

Training and testing the model

Define hyperparameters

```
batch_size = 600       # Batch size
seq_len = 12          # Number of steps
learning_rate = 0.0001
epochs = 1000
```

Summary