Meta-Adaptive Stock Movement Prediction with Two-Stage Representation Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract

Stock movement prediction has always been a tough but attractive task for researchers in machine learning and data mining. Generally speaking, two challenges for stock time series prediction remain not well-explored. One is the overfitting of deep learning models due to the data shortage and the other one is the potential domain shift that may happen during the evolution of stock time series. In this paper, we present Meta-Adaptive Stock movement prediction with two-Stage Representation learning (MASSER), a novel framework for stock movement prediction based on self-supervised learning and meta-learning. Specifically, we first build up a two-stage representation learning framework, the first stage representation learning aims for unified embedding learning for the data. And the second stage learning, which is based on the first stage, is used for temporal domain shift detection via self-supervised learning. Then, we formalize the problem of stock movement prediction into a standard meta-learning setting. Inspired by importance sampling, we estimate sampling probability for tasks to balance the domain discrepancy caused by evolving temporal domains. Extensive experiment results on two open source datasets show that our framework with two simple but classical architectures (GRU and ResNet) as model achieves improvements of 5% - 9.5% on average accuracy, compared to state-of-the-art baselines.

1 Introduction

One challenge for applying deep learning models to stock movement prediction is that there are always limited data and the models trained on small datasets are susceptible to overfitting [4]. Due to this difficulty, a common solution is applying pre-trained models [15]. Instead of training a large deep learning model from scratch, a model trained on relevant data in advance improves the model’s performance in time series classification [11]. However, the conventional transfer learning setting may be inappropriate for stock prediction because of significant heterogeneity among different items. The ideal framework should have strong generalization with flexibility that can be fast adapted for all the stock with this issue of limited data.

Apart from the limited data, another issue is the domain shift in temporal patterns. In real-world scenarios, when researchers are making predictions on stock market, an important concern is non-stationarity [28]. Due to the fact that the environment evolves with time, the streaming data distribution may change in unpredictable ways. In other words, the temporal distribution may shift within a large time scale [20] [34]. More precisely, three main problems lie in this issue, (i) how to detect the domain shift within time scales, (ii) how to perform well towards all the domain shifts since domain shifts could happen in both the training data and testing data, (iii) is there any importance or quantification for domain shifts when considering the shifting degree between different domains. A model aiming to train well on the training set can possibly have poor performance on testing sets because of the
potential domain shifts in testing data. Thus, a model we desire here should have a strong generalization ability towards non-IID tasks from different distributions, rather than improving performance on a specific target domain or distribution.

Before introducing our framework, we first identify two important definitions of our problem setup. **Stock Movement Prediction** Given a stock and its sequential features \(X = \{x_1, \cdots, x_T\} \in \mathbb{R}^{D \times T}\) within \(T\) timestamps, where \(D\) denotes the dimension of the feature at each timestamp. The goal is to learn a prediction function \(Y(X) = f(X; \theta)\) which maps the stock from its sequential features \(X\) to the label space, where the function \(f\) with parameters \(\theta\) aims to predict the movement of stock \(s\) at the next timestamp either rise (\(Y(X) = 1\)) or fall (\(Y(X) = 0\)).

**Temporal Domain Shift** Suppose \(X = \{X_1, \cdots, X_n\}\) is the sequence of stock and movement label sequence \(Y = \{Y_1, \cdots, Y_n\}\) corresponds to \(X\). \(X_i\) and \(Y_i\) are drawn from their corresponding distribution \(X_i \sim P_i(X)\) and \(Y_i \sim P_i(Y)\). the temporal domain shift can be defined as \(\exists k, l: P_k(X) \neq P_l(X)\) or \(P_k(X, Y) \neq P_l(X, Y)\) or \(P_k(Y | X) \neq P_l(Y | X)\).

To address two mentioned issues, we propose **Meta-Adaptive Stock movement prediction with two-Stage Representation learning (MASSER)**, a stock prediction framework that combines supervised learning, self-supervised learning and meta-learning. Extensive experiments show that MASSER outperforms the strong baseline models on several benchmarks in offline settings and online settings. The framework pipeline is shown in Figure 1. Furthermore, MASSER’s performance is better than a SOTA [33] even when MASSER has no access to the social media information in ACL18 dataset [43]. We summarize our contributions as follows,

- We propose a two-stage encoder for stock price representation learning. The first stage aims at learning unified representations of data. The second stage is used to detect temporal domain shift via self-supervised learning. To enhance the generalization ability, we formulate a meta-adaptive learning paradigm for stock movement prediction. The meta-learning framework efficiently learns knowledge across large temporal scales and different stocks and agilely adapts to unseen domains.
- Extensive experiments on two datasets demonstrate the effectiveness of the proposed method for improving prediction accuracy and increasing the generalization ability to temporal domain shift. We extend the offline setting of stock movement prediction to an online paradigm, admitting temporal domain shift could happen in testing streaming data, which is closer to the real day-trading scenarios.

2 Two-Stage Representation Learning for Encoders

2.1 First Stage: Macro Representation Learning

Due to the magnitude of the time scale of the datasets, forecasting decisions are often based on subsequence [45][12][33]. In our framework, we segment the raw data into several subsequences and feed them into the encoder. As it has been shown that temporal convolutional networks (TCN) can often produce superior prediction performance with sequential data [17] and are generally easier to train, we construct our encoder based on TCN. The first-stage encoder \(\theta_1\) aims to learn unified representation on the dataset level of the subsequences of raw data via (i) extracting useful features for prediction and (ii) matching the mapping between the inner structure of learned embeddings and their corresponding prediction labels. The loss function of the first stage is a convex combination of two terms, which match goals (i) and (ii). Suppose \(X = \{X_1, \cdots, X_n\}\) denotes the segmented subsequences input for the encoder and \(Y' = \{Y'_1, \cdots, Y'_n\}\) as the next day rate of change (ROC) of each corresponding subsequence. The first term is mean squared error (MSE),
which can be defined as \( \mathcal{L}_{MSE} = \frac{1}{n} \sum_{i=1}^{n} \left( Y'_i - \hat{Y}'_i \right)^2 \), where \( \hat{Y}'_i \) denotes the first-stage encoder’s prediction of \( X_i \). MSE is used to train the model to precisely predict the input subsequence. In order to achieve goal (ii), the second term is the Frobenius norm of the difference between normalized embedding distance matrix and labels distance matrix within a mini-batch. Let \( X_{i+1}, \ldots, X_{i+m} \) denote a mini-batch of subsequences, \( E_{i+1}, \ldots, E_{i+m} \) the embeddings generated by the TCN \( \theta_1 \), and \( Y'_{i+1}, \ldots, Y'_{i+m} \) the ROC, where \( m \) denotes the batch size. Then we pair every two items \( l \) and \( k \) inside the same batch and compute the pairwise MSE loss on \( \left( E^{q_1}_i, E^{q_k}_i \right) \) and \( \left( Y'_i, Y'_j \right) \). All the pairwise MSEs can be put into two matrices \( E^{q_1}_b \) and \( Y'_b \), and \( b \) represents the \( b \)-th mini-batch. These two matrices share the same size as \( m \times m \). We take the Frobenius norm of \( E^{q_1}_b - Y'_b \) and name it LabelSim: 
\[
\mathcal{L}_{LabelSim} = \| E^{q_1}_b - Y'_b \|_F^2.
\]
The loss function term above forces the model to learn the alignment of intra-batch embeddings \( E^{q_1}_b \) and ROC prediction label \( Y'_b \). Through this objective, the subsequences with a similar ROC could be mapped near each other in the latent space. Thus, the encoder for macro representation learning \( \theta_1 \) can be described as \( \theta_1 = \arg \min_{\theta} \mathbb{E}_{X \in X} a * \mathcal{L}_{MSE} + (1 - a) * \mathcal{L}_{LabelSim} \), where \( a \) is a factor to control the importance of two terms.

### 2.2 Second Stage: Micro Representation Learning

The macro representation learning neglects the temporal order among individual subsequences so that the encoder hardly captures the evolution continuously. Therefore, the macro representation learning may be fragile towards the domain shifts. In the second stage, we modify the encoder to be more sensitive to the temporal domain shift between the continuous subsequences. We name the second stage micro representation learning. Similar to TS-CP2 [9], we apply contrastive learning for temporal domain detection. We set two paired consecutive subsequences as dual windows, the embedding \( E^2_{i+1} \) and its dual \( E^2_{i+1} \), to train the second-stage encoder \( \theta_2 \) and make it adapt to domain shift detection. Specifically, InfoNCE [30] is used for the loss function to maximize the mutual information of dual windows. Suppose there is a mini-batch of embeddings \( E^2_{i+1}, \ldots, E^2_{i+m} \), with batch size \( m \), let \( \left( E^2_{i+1}, E^2_{i+1} \right) \) denote the dual windows of \( X_i \) and \( \left( E^2_{k}, E^2_{k} \right) \) represents the random pairs within the mini-batch. We take \( \left( E^2_{i+1}, E^2_{i+1} \right) \) as positive pair and \( \left( E^2_{k}, E^2_{k} \right) \) as its corresponding negative pairs. The predicting probability \( q_i \) of positive pair \( \left( E^2_{i+1}, E^2_{i+1} \right) \) in each batch is as follows, 
\[
q_i = \frac{\exp \left( \frac{d_{cos}(E^2_{i+1}, E^2_{i+1})}{r} \right)}{\sum_{j \neq i} \exp \left( \frac{d_{cos}(E_j, E_j)}{r} \right)}
\]
where \( r \) is a temperature hyper-parameter for scaling and \( d_{cos} \) is cosine similarity between each pair embeddings. The second-stage learning objective can be described as the minimization of binary cross entropy of the probabilities of all positive pairs within the mini-batch. 
\[
\theta_2 = \arg \min_{\theta} \mathbb{E}_{X \in X} \sum_{i,j} \mathbb{I}_{i=j} \log (q_i) + \mathbb{I}_{i \neq j} \log (1 - q_i)
\]
where \( \mathbb{I}_{i=j} \) and \( \mathbb{I}_{i \neq j} \) are indicator functions. We put more details in Appendix [B.1].

With the encoder \( \theta_2 \) in hand, we compute the similarity of dual windows \( d_{cos}(E^2_{i+1}, E^2_{i+1}) \) for all the input, where \( E^2_{i+1} \) is the embedding for \( X_i \) from encoder \( \theta_2 \). A threshold \( \eta \) is set for inferring whether the embedding contains a shift or not. For a specific embedding \( E^2_{i+1} \), we infer the the subsequence \( X_i \) is under a temporal domain shift if \( \eta \geq d_{cos}(E^2_{i+1}, E^2_{i+1}) \). Specific algorithm can be found in Appendix [B.2].

### 3 Meta-Adaptive Stock Movement Prediction

We elaborate on the meta-learning part of MASER. Details about (i) the specific process of how to utilize the learned two-stage encoder for constructing meta-learning tasks and (ii) how to make the meta-model adaptive to the domain shifts can be found in Appendix [C].

With meta-learning, we construct tasks for the meta-learning model. For classification, each task is further split into support set \( S \) and query set \( Q \) with items with balanced classes. Note that we don’t allow any constructed task contains temporal domain shift (detected by stage 2 representation learning). All the constructed tasks are evaluated by specific criteria of (i) alignment of support set and query set, (ii) temporal adjacency, and (iii) representation adjacency. Each task will be given appropriate probability for sampling to the training process of meta-learning model after the task evaluation process. Tasks with good quality are more likely to be sampled for the training. Details can be found in Appendix [C.3]
We use a 4-layers TCN as our two-stage representation learning encoder, and the window size for the TCN encoder to generate embedding is 25. Furthermore, we apply ResNet for time series classification \[40\] and GRU \[8\] as the meta-model architectures with MAML \[13\] as meta-algorithm for updating parameters. Specifically, We use the Adam \[21\] for the MAML outer loop training and stochastic gradient descent (SGD) for the inner loop training. Utilizing the proposed task constructor, we formulate the meta tasks as 2-way 5-shot. After the meta-training process of MASSER, we first test the performance of the well-trained meta-model on testing data by directly using it to make predictions, and then take an adaptation method to make the meta-model more specific to different stocks. Concretely, we randomly select data with the latest timestamps in the training set within individual stock and slightly update the model parameters based on these selected data to help MASSER better capture the feature of each stock. We report the mean of best testing performance over six different random seeds. Additional description about experiment setup and results of online setting, backtesting, and ablation study can be found in Appendix \[D\].

### 4.1 Offline Experiment Settings

The offline setting here means the model is frozen after the entire learning process and output prediction on all the testing data simultaneously. Table 1 compares the Acc and MCC of our model and baselines for stock movement prediction in ACL18 and KDD17. MASSER-ResNet\[*\] (\* denotes adaptation) achieves the best accuracy and MCC. Compared to the baselines, MASSER-ResNet\[*\] exhibits an improvement of 9.1\% and 64.9\% (2.3\% and 50.0\%) on the ACL18 (KDD17) dataset regarding Acc and MCC, respectively. It is worth noting that the Acc (MCC) of MASSER is 2.6\% (25.1\%) higher than MAN-SF, even though MAN-SF utilizes social media information. Another observation worth mentioning here is that the MASSER-GRU without adaptation averagely increases 1\% Acc on Adv-ALSTM. This justifies the effectiveness of two-stage representation learning and meta-adaptive training, which may be caused by well-learned embedding and meta-adaptive learning on good-quality tasks. With two simple architectures as the meta-model, MASSER shows its strong generalization ability for unseen data and good ability to learn representations, which beats complex architectures with attention module \[27\] and transformer \[59\].

### 4.2 Online Experiment Settings

To evaluate MASSER’s performance, we compare it with the strong baselines for stock movement prediction: Momentum(MOM), LSTM \[18\], GRU \[8\], ALSTM \[31\], StockNet \[43\], Adv-ALSTM \[12\], and MAN-SF \[33\]. We evaluate the prediction performance by two metrics, Accuracy (Acc) and Matthews Correlation Coefficient (MCC) \[43, 12, 33\]. Note that better performance is evidenced by the higher value of the metrics.

We use two open-source stock datasets for evaluation, ACL18 dataset \[43\] and KDD17 dataset \[47\].

<table>
<thead>
<tr>
<th>Model</th>
<th>ACL18 Acc</th>
<th>ACL18 MCC</th>
<th>KDD17 Acc</th>
<th>KDD17 MCC</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOM</td>
<td>0.470</td>
<td>-0.064</td>
<td>0.498</td>
<td>-0.013</td>
</tr>
<tr>
<td>LSTM</td>
<td>0.532</td>
<td>0.067</td>
<td>0.516</td>
<td>0.018</td>
</tr>
<tr>
<td>ALSTM</td>
<td>0.549</td>
<td>0.104</td>
<td>0.519</td>
<td>0.026</td>
</tr>
<tr>
<td>StockNet</td>
<td>0.550</td>
<td>0.017</td>
<td>0.499</td>
<td>0.499</td>
</tr>
<tr>
<td>Adv-ALSTM</td>
<td>0.572</td>
<td>0.148</td>
<td>0.531</td>
<td>0.052</td>
</tr>
<tr>
<td>MAN-SF</td>
<td>0.608</td>
<td>0.195</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>MASSER-ResNet</td>
<td>0.592</td>
<td>0.099</td>
<td>0.535</td>
<td>0.074</td>
</tr>
<tr>
<td>MASSER-GRU</td>
<td>0.579</td>
<td>0.141</td>
<td>0.543</td>
<td>0.073</td>
</tr>
<tr>
<td>MASSER-ResNet[*]</td>
<td>0.624</td>
<td>0.244</td>
<td>0.542</td>
<td>0.078</td>
</tr>
<tr>
<td>MASSER-GRU[*]</td>
<td>0.581</td>
<td>0.162</td>
<td>0.543</td>
<td>0.047</td>
</tr>
</tbody>
</table>

Table 1: Offline Setting Acc and MCC on ACL18 and KDD17 (\* means adaptation)

### 5 Conclusion
In this paper, We introduce MASSER, a novel framework for stock movement prediction based on self-supervised learning and meta-learning. Extensive experiments on two dataset show that MASSER outperforms baselines in both offline and online settings, which justifies MASSER’s strong generalization ability.

---

1. https://github.com/yumoxu/stocknet-dataset
References


A Related Work

Stock Movement and Trading Modeling The previous work of stock movement can be separated into two categories. Individual stock modeling, which captures complex repetitive patterns from the historical prices of each individual stock instead of finding the correlations between multiple stocks \cite{29,43,12,42}. And taking the correlation between different stocks into consideration because correlations between multiple stocks may help to make robust and consistent predictions \cite{31,33,45}. MASSER extends the view of correlation to meta-learn unified representations on the dataset level. In MASSER, not only is domain shift detection is considered, but also the importance of different domains, which could make our framework more consistent with the general pattern of historical data. Furthermore, our framework is flexible with online adaptation and offline scenarios. All the related works mentioned above are inappropriate with online settings, and those models haven’t taken domain shifts within testing data into consideration.

Time Series Modeling with Domain Shifts Deep learning-based methods for time series modeling gained popularity over the past years \cite{35,36}. Recently, domain shift and concept drift modeling for time series have become a popular topic in the community of data mining \cite{10,26}. And more recently, Li et al. \cite{23} construct a resampling strategy for data distribution generation for predictable concept drift adaptation. In MASSER, we aim to solve the temporal domain shift issue as well. The high stochasticity of stock data is a significant property. Therefore, we design a two-stage encoder learning process to handle this issue. The first stage encoder can learn the general features and denoise the data for the second stage domain shift detection. The second stage follows the first one by taking the encoder as a pre-trained model and learning to detect domain shifts via self-supervised learning.

Meta-learning By using the information gained from earlier assignments, meta-learning has emerged as an efficient model for learning with minimal data \cite{13,37,22,14,5}. In common scenarios for meta-learning, the assumption that all tasks are equally important is held, therefore each iteration samples tasks at random. In \cite{32,19,25}, the probability of sampling a task from existing meta-training tasks is proportional to the amount of information it provides. Recently, some work \cite{24,44,7} focused on task selection via reinforcement learning \cite{41} and submodular maximization \cite{5}. From the perspective of meta-learning, MASSER focuses on the same issue of task sampling but additionally takes domain shifts into consideration. We do not assume that tasks are equally significant. MASSER adaptively samples tasks for the meta-model based on learned temporal patterns via self-supervised representations to enhance the learning of general patterns against out-of-distribution (OOD) data.

Self-Supervised Learning Self-supervised learning has been utilized to capture informative and compact representations of video \cite{1}, image \cite{6}, and time series \cite{9}. Specifically, we consider contrastive learning for our framework. Contrastive learning uses a collection of training examples made up of positive and negative sample pairs to establish what makes the samples in a dataset similar or dissimilar. $N$-Paired loss \cite{38} and InfoNCE based on Noise Contrastive Estimation \cite{16,26} are examples of recent multiple negative learning loss functions that examine numerous negative sample pairs at the same time. Auto-regressive models are used in Contrastive Predictive Coding (CPC) \cite{50,47} to learn representations within a latent embedding space. CPC aims to learn inside a global, abstract representation of the signal rather than a high-dimensional, lower-level representation. Recently, TSCP2 \cite{9} uses CPC to learn latent representations and find dissimilarities in time series data.

B Detailed Two-stage Representation Learning

Figure 2 shows the pipeline and illustration of our two-stage representation learning.

B.1 The InfoNCE Loss

The InfoNCE loss was first proposed by Oord et al. \cite{30}. It has been proven that by minimizing the InfoNCE loss, we are actually maximizing the mutual information between dual windows. Take the same annotations in main context, $X = \{X_1, \cdots, X_n\}$ denotes the segmented subsequences input for the encoder, $E^{\theta_2}_{i+1}, \cdots, E^{\theta_2}_{i+m}$ is a mini-batch of embeddings with batch size $m$, let $(E^{\theta_2}_{i}, E^{\theta_2}_{(d),i})$
Figure 2: The two-stage encoder training framework pipeline. On the left side of the figure, the first-stage encoder is trained with a weighted sum of Label similarity loss and prediction loss to learn unified representations. Then on the right side, the second-stage encoder is trained based on the Temporal Conv encoder duplicated from the first stage to detect temporal domain shift via self-supervised learning of dual windows. The specific network structure of the Temporal Conv encoder is shown on the top of the middle.

denotes the dual windows of \( X_i \) and is taken as positive pair, \((E_{\theta_2}^{d_{i,j}}, E_{\theta_2}^{d_{(d,j,i)}})_{k \neq l}\) represents the random pairs within the mini-batch and they are all negative pairs. By the original definition of the InfoNCE loss, we are about to minimize:

\[
L_N = -E \log \frac{\exp \left( \frac{d_{\cos} \left( E_{\theta_2}^{d_{i,j}}, E_{\theta_2}^{d_{(d,j,i)}} \right)}{\tau} \right)}{\sum_{j=1}^{m} \exp \left( \frac{d_{\cos} \left( E_{\theta_2}^{d_{i,j}}, E_{\theta_2}^{d_{(d,j,i)}} \right)}{\tau} \right)}
\]  

Armed with the knowledge of multi-class classification, it is easy to show that Equation 1 is the cross-entropy loss of classifying the positive pair correctly, and the predicting probability \( q_i \) of positive pair \((E_{\theta_2}^{d_{i,j}}, E_{\theta_2}^{d_{(d,j,i)}}) \) in each mini-batch is exactly the interior part of \( L_N \):

\[
q_i = \frac{\exp \left( \frac{d_{\cos} \left( E_{\theta_2}^{d_{i,j}}, E_{\theta_2}^{d_{(d,j,i)}} \right)}{\tau} \right)}{\sum_{j=1}^{m} \exp \left( \frac{d_{\cos} \left( E_{\theta_2}^{d_{i,j}}, E_{\theta_2}^{d_{(d,j,i)}} \right)}{\tau} \right)}
\]

Finally, the optimal parameter \( \theta_2 \) can be obtained by:

\[
\theta_2 = \arg \min_{\theta} -E_{X \in X} \sum_{i,j} I_{i=j} \log (q_i) + I_{i \neq j} \log (1 - q_i)
\]

**B.2 Details about Detecting Temporal Domain Shift**

As shown in Figure 3, we slide dual windows in the time domain and apply second-stage encoder to transform them into embedding pairs \((E_{\theta_2}^{d_{i,j}}, E_{\theta_2}^{d_{(d,j,i)}}) \). Next we compute the similarity of adjacent embedding from the same pair and filter the pairs whose similarity is below the threshold \( \eta \), which can be computed as:

\[
\eta = \varepsilon \times (\delta \times \eta_{pos} + (1 - \delta) \times \eta_{neg})
\]

where \( \eta_{pos} \) is the average of positive pairs’ similarities of in training set, and \( \eta_{neg} \) represents the negative ones’. The adjustability coefficient \( \varepsilon \) and \( \delta \) are set to 0.35 and 0.66 empirically. For each
filtered embedding pair, we take the time point \( t_i \) which is corresponded to the junction of \( E^{θ_2}_i \) and \( E^{θ_2}_{(d_j,i)} \) as the midpoint and extent it to a forbidden time range \((t_i - τ, t_i + τ)\), which possibly contains domain shift. Let set \( S \) be the union of all the timestamps of the raw stock data. After dropping the timestamps in forbidden ranges, stock data in time domain is split into the consecutive segments \( S_1, \cdots, S_j \). Then, generate all their embeddings using first-stage encoder consecutively for constructing meta-learning tasks.

C Detailed Meta-Adaptive Stock Movement Prediction

In this section, we elaborate on the meta-learning part of MASSER, (i) the specific process of how to utilize the learned two-stage encoder for constructing meta-learning tasks, and (ii) the details of how to make the meta-model adaptive to the domain shifts. The motivation for (i) is that tasks containing domain shifts are not desired for the construction because of the assumption that the data inside the support set and query set is roughly IID in a meta-learning setting. When facing multiple domains following different distributions, the model needs to focus more on the general patterns within the whole time scales and prioritize the tasks with good quality. To demonstrate (ii), we first introduce how to evaluate task quality and then show how to make the meta-model adaptive.

C.1 Meta-learning Task Construction

We compute the similarity of dual windows \( \{d_{\cos}(E^{θ_2}_i, E^{θ_2}_{(d_j,i)})\}_{i=1, \cdots, n} \) and search all the dual windows whose similarity is below the threshold \( η \). Let set \( S \) be the union of all the timestamps that covered in inferred embedding entries \( \{E^{θ_2}_i\}_{i=1, \cdots, n} \) that are below the threshold and take the complement of \( S \) within all the timestamps as the feasible set for the following task construction. Let \( S_1, \cdots, S_j \) be the consecutive segments within the feasible set. Then, generate all their embeddings from first-stage encoder \( θ_1 \) consecutively and we consider the embedding group of each segments \( S_i \) as a domain \( D_i \), which doesn’t contain shift inside.

With meta-learning, we construct tasks for the meta-learning model. For classification, each task is further split into support set \( S \) and query set \( Q \) with items with balanced classes. For task \( T_{ij} \), all the data (subsequences) is selected from \( D_i \) and \( ij \) denotes the \( j \)-th task constructed inside domain \( D_i \).

Suppose there are \( n* \) classes in the whole dataset, and we intend to build a \( n \) way \( k \) shot task \( T_{ij} \) \((n* \geq n)\). We first build the support set. As for the time series data, we set the data in the support set ahead of data in the query set. We build a candidate pool for each class and scan the domain \( D_i \) by temporal order to divide data into the class candidate pools. The dividing process stops by checking all the class candidate pools have a number of data no less than \( k \). For the class candidate pools that have a number of data larger than \( k \), we use random sampling to draw \( k \) samples into the support set \( S_{ij} \). Then we start to build to query set \( Q_{ij} \) for task \( T_{ij} \) by the same process. Tasks in \( D_i \) are constructed in order, and we allow different tasks’ support sets have overlapped.

We show the detailed procedures of task construction in Algorithm[1]
Algorithm 1 Task Construction

Require: Domains $D_1, \ldots, D_m$, $n$ way $k$ shot, $k'$ for query set, window size $l$, retreated ratio $\alpha$
1: for $t = 1, 2, \ldots, m$ do
2: Construct class candidate pools $C_1, \ldots, C_{n}$ for all the classes
3: //Stage I: Support Set Construction
4: Sample $n$ classes for current task $T_{tj}$
5: while not all the item numbers for sampled class candidate pools $|C_i| \geq k$ do
6: Add data into corresponding class candidate pool $C_i$
7: end while
8: for sampled class candidate pools $|C_i| > k$ do
9: Random draw $k$ samples to keep in the class candidate pool and delete the rest
10: end for
11: Merge the current sampled class candidate pool $C_i$ and save as task $T_{tj}$’s support set $S_{tj}$
12: //Stage II: Query Set Construction
13: Clean all the sampled class candidate pools $C_i$
14: while not all the item numbers for sampled class candidate pools $|C_i| \geq k'$ do
15: Add data into corresponding class candidate pool $C_i$
16: end while
17: Merge the current sampled class candidate pool $C_i$ and save as task $T_{tj}$’s query set $Q_{tj}$
18: Initialize a new starting time stamp for the new task $T_{t(j+1)}$ with the retreated ratio $\alpha$ and last task $T_{tj}$’s starting point
19: Check whether the data in $D_t$ after the starting point is enough for constructing a new task or not. If enough, return to line 4 for $T_{t(j+1)}$ construction. If not, shift to $D_{t+1}$
20: end for

C.2 Task Evaluation

We present detailed criteria for evaluating task quality after task construction, according to the following three factors.

• Alignment of Support Sets and Query Sets Task $T_{tj}$ similarity between the support and query set embeddings $E_{S_{tj}}, E_{Q_{tj}}$ with respect to the first-stage encoder, i.e.,

$$S_c(E_{S_{tj}}, E_{Q_{tj}}) := \frac{E_{S_{tj}}^\theta, E_{Q_{tj}}^\theta}{\|E_{S_{tj}}^\theta\| \|E_{Q_{tj}}^\theta\|}$$

(2)

Here, we use cosine similarity as metric. Specifically, the embedding similarity signifies the generalization gap from the support set to the query set since segmentation results from the second-stage encoder could not strictly guarantee the unification and stationary of data in the support set and query set. Ideally, good quality tasks we desire should have close embeddings for support sets and query sets.

• Temporal Adjacency When facing a large temporal scale time series, the data with later timestamps may be more important than the former ones, particularly for those that have domain shifts inside the time series but no shift within the latest parts. In general case, any monotonically increasing function can be used here. For simplicity, we apply a linear monotonically increasing function, i.e., $S_{TA}(T_{tj}) = k \ast t_{T_{tj}}$, where $k$ ($k > 0$) is the slope of the linear function and $t$ is the average timestamp of $T_{tj}$.

• Representation Adjacency Since focusing on general patterns on the dataset level, we want to degrade the importance of OOD data. For simplicity, we consider this issue from the inter-domain level rather than the intra-domain (task) level. We propose to compute how close a single domain is to the others. In terms of efficiency, assume a large number of domains inside the dataset, it is computationally costive to compute all the distances between a specific domain and the rest. Thus, we intend to randomly partition domains into several groups and compute the sum of the distance inside each group. Suppose for each randomly partitioned group whose size is $m$, and then we can input all the pairwise domain embedding distance into a symmetric square matrix with the size of $m \times m$. To sum up all the columns or rows, the distance between a specific domain and all the rest is acquired. Let $S_{RA}(T_{tj})$ denote the distance sum of task $T_{tj}$ within its random partitioned group.
We demonstrate our pipeline of Meta-Adaptive Framework in **Algorithm 2**.

**Algorithm 2 Task Evaluation**

Require: Tasks $T_{11}, \ldots, T_{1j}, T_{21}, \ldots, T_{m_{jm}}$, Domains $D_1, \ldots, D_m$, Starting Timestamps of Tasks $t_{11}, \ldots, t_{1j}, t_{21}, \ldots, t_{m_{jm}}$, Temporal Adjacency Slope $k$, Representation Adjacency Group Size $m'$, Factor Rate $\beta_1, \beta_2, \beta_3$

1: //Alignment of Support and Query and Temporal Adjacency
2: for $T_{11}, \ldots, T_{1j}, T_{21}, \ldots, T_{m_{jm}}$ do
3: Get support embedding $E_{S_{ij}}^\theta$ and query embedding $E_{Q_{ij}}^\theta$ for task $T_{ij}$ through encoder $\theta_1$
4: Compute alignment of support set and query set:
5: 
6: Compute temporal adjacency:
7: //Representation Adjacency
8: Random partition tasks $T_{11}, \ldots, T_{1j}, T_{21}, \ldots, T_{m_{jm}}$ into different groups with the group size $m'$ (assume there are $l$ groups)
9: for $k = 1, \ldots, l$ do
10: Compute pairwise task similarity $S_C(E_{T_{ij}}^\theta, E_{T_{ij'}}^\theta)$ and fill the similarity into the pairwise similarity matrix $RA_k$
11: Compute the representation adjacency $S_{RA}(T_{ij})$ by sum up the similarity of itself with the other tasks within group $k$ (sum up the corresponding row/column of $RA_k$)
12: end for
13: for $T_{11}, \ldots, T_{1j}, T_{21}, \ldots, T_{m_{jm}}$ do
14: Compute the sampling probability of task $T_{ij}$ with normalization by:
15: 

**C.3 Training Meta-Models Adaptively**

Considering three factors simultaneously, we define the sampling probability $w_{ij}$ of each task $T_{ij}$ as

where $\beta_1, \beta_2, \beta_3$ are parameters controlling its corresponding factor.

We use sampling probability weight $w_{ij}$ to sample $B$ tasks from the task pool for current meta-training iteration, where a larger value of $w_{ij}$ represents higher probability.

We demonstrate our pipeline of Meta-Adaptive Framework in **Algorithm 3** with MAML[13] as the meta-learning algorithm. The two-stage representation learning process is not included.

**Algorithm 3 Meta-Adaptive Pipeline**

Require: Domains $D_1, \ldots, D_m$, Input Parameters of Task Construction $\phi_1$, Input Parameters of Task Evaluation $\phi_2$, Task Sampling Size $m'$

1: Get tasks $T_{11}, \ldots, T_{1j}, T_{21}, \ldots, T_{m_{jm}}$ by Task Construction (**Algorithm 1**) with input parameters $\phi_1$
2: Get task sampling probability $w_{11}, \ldots, w_{1j}, w_{21}, \ldots, w_{m_{jm}}$ by Task Evaluation (**Algorithm 2**) with input parameters $\phi_2$
3: Sample $m'$ tasks according to task sampling probability $w_{11}, \ldots, w_{1j}, w_{21}, \ldots, w_{m_{jm}}$
4: Update Meta-Adaptive model with MAML.
D Additional Experiment Results

D.1 Dataset Description

We choose two open-source stock datasets for evaluation, ACL18 dataset\(^3\) and KDD17\(^4\). ACL18 contains historical price and social media information from Jan-01-2014 to Jan-01-2016 of 88 high-trade-volume stocks in NYSE and NASDAQ markets. KDD17 includes longer history ranging from Jan-01-2007 to Jan-01-2016 of 50 stocks in U.S. markets. Six features are included in this dataset. Three technical indicators \(z_{\text{open}}, z_{\text{high}}, z_{\text{low}}\) are inter-day-based features. Two technical indicators \(z_{\text{close}}\) and \(z_{\text{adj}_\text{close}}\) are intra-day-based features, where \(z_{\text{adj}_\text{close}}\) denotes the adjusted closing price which reflects the stock’s value after accounting for any corporate actions, such as stock splits, dividend distribution, and rights offerings. For more details, please reach Table 2. The labels are assigned to each time window based on the next day’s movement percent. Movement percent \(\geq 0.55\%\) and \(\leq -0.5\%\) are labeled as positive and negative examples. In the offline setting experiment of ACL18, we use the time period Jan-01-2014 to Aug-01-2015 as the training dataset and Oct-01-2015 to Jan-01-2016 as the testing dataset. And for KDD17, we split the date from Jan-04-2007 to Dec-31-2014 as the training set, Jan-02-2015 to Jan-04-2016 as a validation set, and Jan-04-2016 to Dec-30-2016 as the testing set. The setting above is the same as \(^{43}\) and \(^{12}\). Furthermore, for our proposed online setting, we set Oct-01-2015 to Sep-01-2016 as the online testing set.

D.2 Baselines

We compare MASSER with the following baselines for stock movement prediction.

- **MOM** Momentum (MOM) is a technical indicator that predicts negative or positive for each example with the trend in the last 10 days
- **LSTM** \(^{12}\) is a baseline for stock movement prediction, which uses a neural network with an LSTM layer and a prediction layer.
- **ALSTM** \(^{31}\) represents attention LSTM that correlates multiple time steps using a temporal attention mechanism.
- **StockNet** \(^{43}\) uses variational autoencoders to encode stock movements as latent probabilistic vectors.
- **Adv-ALSTM** \(^{12}\) is the previous SOTA (with only stock price) that trains ALSTM with adversarial data to improve the robustness of stock movement prediction.
- **MAN-SF** \(^{33}\) is the previous SOTA (with social media information) that uses graph neural network to blend chaotic temporal signals, social media, and inter-stock relationship.

D.3 Implementation Details

We set the learning rate for meta-adaptive model as 0.004-0.006, batch size as 16, 32 and 64. learning rate for online setting as 0.001. Grid search is used to find the appropriate learning rates and batch size of the model.

---

\(^3\)https://github.com/yumoxu/stocknet-dataset

\(^4\)https://github.com/z331565360/State-Frequency-Memory-stock-prediction/tree/master/dataset

Table 2: Features of ACL18 and KDD17 Dataset

<table>
<thead>
<tr>
<th>Features</th>
<th>Calculation</th>
</tr>
</thead>
<tbody>
<tr>
<td>(z_{\text{open}})</td>
<td>(z_{\text{open}} = \frac{\text{open}<em>t}{\text{close}</em>{t-1}})</td>
</tr>
<tr>
<td>(z_{\text{high}})</td>
<td>(z_{\text{high}} = \frac{\text{high}<em>t}{\text{close}</em>{t-1}})</td>
</tr>
<tr>
<td>(z_{\text{low}})</td>
<td>(z_{\text{low}} = \frac{\text{low}<em>t}{\text{close}</em>{t-1}})</td>
</tr>
<tr>
<td>(z_{\text{close}})</td>
<td>(z_{\text{close}} = \frac{\text{close}<em>t}{\text{close}</em>{t-1}})</td>
</tr>
<tr>
<td>(z_{\text{adj}_\text{close}})</td>
<td>(z_{\text{adj}<em>\text{close}} = \frac{\text{adj}</em>\text{close}<em>t}{\text{adj}</em>\text{close}_{t-1}})</td>
</tr>
<tr>
<td>(z_{\text{volume}})</td>
<td></td>
</tr>
</tbody>
</table>
D.4 Online Experiment Setting

We further demonstrate MASSER can deal with streaming data through an online experiment setting as well. Under the online setting, future data flows in batches according to temporal order. When the model gets access to a new batch of future data, we first use the model updated with the last batch of data to make movement predictions, and then leverage the information of this new batch of data to update the model once again. When all the future data has been tested, we can calculate the Acc and MCC of the online setting. To cope with domain shift, we additionally introduce Bayesian Online Changepoint Detection (BOCPD) \cite{2} to assist MASSER in this online setting as an advanced detector. If there is no domain shift within a batch of new data according to the BOCPD result, we use the previously updated model to make predictions on this batch of data and update the model as usual. However, when domain shift occurs within a batch of new data, we use the previously updated model to make predictions on the data before the domain shift point, and use the meta-model without any update to predict the data after the time domain shift occurs, and re-update the meta-model from scratch.

Table 3 compares the Acc and MCC of our model and baselines for stock movement prediction on ACL18 in the online setting. MASSER-ResNet outperforms baselines, with an average improvement of 18.6% on Acc. Furthermore, BOCPD helps MASSER-ResNet to be more agile towards potential domain shifts in the streaming data. From the experiment results of the online setting, we can tell that meta-adaptive training precisely capture the general pattern of training data and make the model has a powerful generalization ability.

<table>
<thead>
<tr>
<th>Model</th>
<th>ACL18 Acc</th>
<th>ACL18 MCC</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSTM</td>
<td>0.516</td>
<td>0.045</td>
</tr>
<tr>
<td>GRU</td>
<td>0.509</td>
<td>0.041</td>
</tr>
<tr>
<td>ALSTM</td>
<td>0.487</td>
<td>0.012</td>
</tr>
<tr>
<td>MASSER-ResNet</td>
<td>0.612</td>
<td>0.216</td>
</tr>
<tr>
<td>MASSER-ResNet-BOCPD</td>
<td>0.625</td>
<td>0.251</td>
</tr>
</tbody>
</table>

Table 3: Online Setting Acc and MCC on ACL18

D.4.1 Additional Brief Introduction to BOCPD \cite{2}

Let \( r_{i,t} \) be the run length of \( x_i \) at time \( t \) and \( x_{i,t} \) be the data observation at time \( t \) of \( x_i \). We denote a set of consecutive discrete observations between timestamp \( a \) and \( b \) inclusive as \( x_{a:b} \). Let \( x_{i,t}^{(r_{i,t})} \) denote the data observations since \( x_i \)'s most recent change point of covariate shift. For simplicity, we use \( x_{i,t}^{(r)} \) instead of \( x_{i,t}^{(r_{i,t})} \) when there is no ambiguity. Through marginalization of \( r_{i,t} \), we calculate the distribution of the next data \( x_{i,t+1} \) given the data up to time \( t \),

\[
\Pr(x_{i,t+1} \mid x_{i,1:t}) = \sum_{r_{i,t}} \Pr(x_{i,t+1} \mid x_{i,t}^{(r)}) \Pr(r_{i,t} \mid x_{i,1:t}) \tag{7}
\]

The exact inference on the run length distribution \( \Pr(r_{i,t} \mid x_{i,1:t}) \) is done recursively as follows,

\[
\Pr(r_{i,t}, x_{i,1:t}) = \sum_{r_{i,t-1}} \Pr(r_{i,t} \mid r_{i,t-1}, x_{i,t-1}^{(r)}) \cdot \Pr(x_{i,t} \mid r_{i,t-1}, x_{i,t-1}^{(r)}) \Pr(r_{i,t-1}, x_{i,1:t-1}) \tag{8}
\]

The first term in Equation (8) can be substituted by \( \Pr(r_{i,t} \mid r_{i,t-1}) \) under the assumption that the current run length only depends on the previous run length. Here, the conditional prior of \( r_{i,t} \) is given by

\[
\Pr(r_{i,t} \mid r_{i,t-1}) = \begin{cases} H (r_{i,t-1} + 1) & r_{i,t} = 0 \\ 1 - H (r_{i,t-1} + 1) & r_{i,t} = r_{i,t-1} + 1 \\ 0 & otherwise \end{cases} \tag{9}
\]
where $H(\tau)$ is the hazard function $H(\tau) = \frac{P_{gap}(g=\tau)}{\sum_{g=1}^{\infty} P_{gap}(g=\tau)}$. $P_{gap}(g)$ is the a priori probability distribution over the interval between change points. When $P_{gap}(g)$ is a geometric distribution with timescale $\lambda$, the hazard function becomes constant as $H(\tau) = 1/\lambda$. The second term in Equation 8 can be calculated using a Gaussian Process. Thus, we can calculate the overall distribution $P(x_{i,t+1} | x_{i,1:t})$ by recursive message passing scheme with $P(r_{i,t}, x_{i,1:t})$. $H(\tau)$ is the hazard function with $H(\tau) = \frac{P_{gap}(g=\tau)}{\sum_{g=1}^{\infty} P_{gap}(g=\tau)}$.

### D.4.2 Backtesting

To illustrate how accurate stock movement prediction contributes to real investment tasks, we design a straightforward strategy by taking long-short decisions only based on the signals from the model. Considered trading on one type of stock, this strategy sells all holding shares for cash if the model predicts the stock price to fall, and purchases the maximum affordable shares if the model provides rising signals. We construct the portfolio based on this strategy with predictions from MASSER-ResNet and three baseline predictive models (LSTM, GRU and ALSTM). On day $t$, our strategy reacts directly based on the movement prediction $\hat{y}_t$. We assume that at the start of the backtesting, the trader has 1 unit of asset, and on day $t$, the market value of his/her asset is $w_t$. If the utilized model predicts the price of stock to rise from day $t$ to $t+1$, say $\hat{y}_t = 1$, the trader will keep all his/her wealth as stock. The action ‘keep’ indicates that if the trader holds stocks on day $t$, he/she would not take actions for the position, and if the trader holds cash, he/she will purchase stocks with all cash on day $t$. Otherwise ($\hat{y}_t = 0$), he/she will keep all his/her wealth as cash. Following this strategy, if one model can predict all rises and falls accurately, we may take the advantages of all rises and avoid all falls, and the maximum final accumulative return will be \( \prod_{t=0}^{T-1} (1 + r_t^+) \), where $r_t^+ = \max (r_t, 0)$ and $r_t$ represents the daily return from day $t$ to $t+1$. Intuitively, the more accurate one movement prediction model is, the better investment return we can obtain through this signaling strategy.

We list and compare the performance of our algorithm and baselines in terms of the average achieved return rate of all 88 considered stocks within the 205-day testing period in Table 4. MASSER outperforms all baselines, achieving a 29.52% return rate, beating LSTM by 20.71%, GRU by 20.97% and ALSTM by 16.38%.

<table>
<thead>
<tr>
<th>Models</th>
<th>Achieved return rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stock</td>
<td>5.59%</td>
</tr>
<tr>
<td>LSTM</td>
<td>8.81%</td>
</tr>
<tr>
<td>GRU</td>
<td>8.55%</td>
</tr>
<tr>
<td>ALSTM</td>
<td>13.14%</td>
</tr>
<tr>
<td>MASSER-ResNet</td>
<td>29.52%</td>
</tr>
</tbody>
</table>

Table 4: Average Online Return Rate on ACL18

### D.5 Ablation Study

We conduct extensive ablation experiments on the overall proposed framework, including whether to use macro representation learning (1st encoder) to derive good representations for downstream meta-learning, whether to perform micro representation learning (2nd encoder), whether to use the task sampler to adaptively meta-train the model (sampler), and whether to use a domain shift detector (BOCPD). The experiment results are shown in Table 5. It is worth noting that the complete MASSER framework yields the best result among all the combinations in both offline and online settings. Models with macro representation learning already achieve better performance compared with other baseline models. After the second-stage self-supervised learning, potential domain shift points are detected and left out, the overall accuracy further improves by 1% to 2%. In general, the task sampler contributes to the improvement of Acc according to the experiment results. It is also evident that BOCPD indeed helps MASSER get more accurate online movement prediction results, which can be attributed to BOCPD’s ability to detect potential domain shifts. The complete MASSER framework also gets the best MCC compared with framework without self-supervised learning, framework without task sampler, and framework with only first-stage representation learning.
<table>
<thead>
<tr>
<th>Model</th>
<th>ACL18</th>
</tr>
</thead>
<tbody>
<tr>
<td>MASSER-ResNet(w/o encoders)</td>
<td>0.544</td>
</tr>
<tr>
<td>MASSER-GRU(w/o encoders)</td>
<td>0.548</td>
</tr>
<tr>
<td>MASSER-ResNet(w/o 2nd encoder)</td>
<td>0.552</td>
</tr>
<tr>
<td>MASSER-GRU(w/o 2nd encoder)</td>
<td>0.580</td>
</tr>
<tr>
<td>MASSER-ResNet(w/o sampler)</td>
<td>0.553</td>
</tr>
<tr>
<td>MASSER-GRU(w/o sampler)</td>
<td>0.580</td>
</tr>
<tr>
<td>MASSER-ResNet</td>
<td>0.552</td>
</tr>
<tr>
<td>MASSER-GRU</td>
<td>0.580</td>
</tr>
<tr>
<td>MASSER-ResNet*(w/o 2nd encoder)</td>
<td>0.607</td>
</tr>
<tr>
<td>MASSER-GRU*(w/o 2nd encoder)</td>
<td>0.571</td>
</tr>
<tr>
<td>MASSER-ResNet*(w/o sampler)</td>
<td>0.619</td>
</tr>
<tr>
<td>MASSER-GRU*(w/o sampler)</td>
<td>0.580</td>
</tr>
<tr>
<td>MASSER-ResNet*</td>
<td>0.624</td>
</tr>
<tr>
<td>MASSER-GRU*</td>
<td>0.581</td>
</tr>
<tr>
<td>MASSER-ResNet(w/o 2nd encoder)</td>
<td>0.613</td>
</tr>
<tr>
<td>MASSER-ResNet</td>
<td>0.612</td>
</tr>
<tr>
<td>MASSER-ResNet-BOCPD(w/o 2nd)</td>
<td>0.605</td>
</tr>
<tr>
<td>MASSER-ResNet-BOCPD</td>
<td>0.625</td>
</tr>
</tbody>
</table>

Table 5: Ablation Study Acc and MCC on ACL18 (* means adaptation)