Learnware: Small Models Do Big

Zhi-Hua Zhou, Zhi-Hao Tan

National Key Laboratory for Novel Software Technology
Nanjing University, Nanjing 210023, China
zhouzh@nju.edu.cn

Abstract

There are complaints about current machine learning techniques such as the requirement of a huge amount of training data and proficient training skills, the difficulty of continual learning, the risk of catastrophic forgetting, the leaking of data privacy/proprietary, etc. Most research efforts have been focusing on one of those concerned issues separately, paying less attention to the fact that most issues are entangled in practice. The prevailing *big model* paradigm, which has achieved impressive results in natural language processing and computer vision applications, has not yet addressed those issues, whereas becoming a serious source of carbon emissions. This article offers an overview of the *learnware* paradigm, which attempts to enable users not need to build machine learning models from scratch, with the hope of reusing small models to do things even beyond their original purposes, where the key ingredient is the *specification* which enables a trained model to be adequately identified to reuse according to the requirement of future users who know nothing about the model in advance.

1. Introduction

Machine learning has achieved great success, while there are lots of complaints about the requirement of a huge amount of training data (particularly data with labels), the difficulty of adapting a trained model to changing environments, and the embarrassment of catastrophic forgetting when refining a trained model incrementally is demanded, etc. There are great efforts such as *weakly supervised learning* [23] trying to reduce the requirement

of labeled training data, *open-environment machine learning* [24] trying to enable learning models to adapt to environments, *continual learning* [4] trying to help deep neural networks resist forgetting; however, these issues are still far from solved.

Indeed, most efforts have been focusing on one of those concerned issues separately, paying less attention to the fact that most issues are entangled in practice. For example, a well-studied technique of weakly supervised learning for reducing the requirement of labeled training data is to collect and exploit a huge amount of unlabeled data drawn from the distribution the same as that of the labeled training data, paying less attention to the fact that in changing environments the data distributions are subject to change inherently. For another example, an effective approach to cope with changing environments is to emphasize data received in very recent timeslots since the changes have not yet caused significant differences, paying less attention to the fact that the emphasis on very recent data may tend to aggravate the severity of catastrophic forgetting.

There are many other issues, e.g., most ordinary users can hardly produce well-performed models starting from scratch, due to the lack of proficient training skills; in many real-world tasks the data privacy/proprietary issue may disable data sharing, leading to the difficulty of sharing experience among different users; in really big data applications, it is generally unaffordable or even infeasible to hold the whole data to support many passes of scanning.

The prevailing *big model* paradigm, which has achieved impressive results in natural language processing and computer vision applications [16, 3], has not yet addressed the above issues. Note that each big model is targeted to a task (or task class) planned in advance, generally helpless to others, e.g., a big model trained for face recognition can hardly be helpful to financial futures trading. It would be too ambitious to build a pre-trained big model for every possible task, because the amount of possible tasks can be unimaginably big or even infinite. In addition, sadly, the training of big models is becoming a serious source of carbon emissions threatening our environment.

Admitting the usefulness of big models in their specifically targeted tasks, is there any paradigm offering the possibility of tackling the above issues simultaneously?

This article overviews the progress of *learnware*, a paradigm offering promising answer to the above question. It attempts to systematically reuse small models to do things that may even be beyond their original purposes, and enables users not need to build their machine learning models from scratch.

2. The Learnware Proposal

The learnware paradigm was proposed in [22]. A learnware is a well-performed trained machine learning model with a *specification* which enables it to be adequately identified to reuse according to the requirement of future users who know nothing about the learnware in advance.

The developer or owner [1] of a trained machine learning model (no matter whether the model is a deep neural network, a support vector machine, or a decision tree, etc.) can spontaneously submit her trained model into a *learnware market*. If the learnware market decides to accept the model, it assigns a specification to the model and accommodates it in the market. The learnware market should not be small, and it would be common to accommodate thousands or millions of well-performed models submitted by different developers, on different tasks, using different data, optimizing different objectives, etc.

Once the learnware market has been built, when a user is going to tackle a machine learning task, she can do it in the following way rather than building her model from scratch. As the comic in Figure 1 illustrates, she can submit her *requirement* to the learnware market, and then the market will identify and deploy some helpful learnware(s) by considering the learnware specification. The learnware can be applied by the user directly, or adapted/polished by user's own data for better usage, or exploited in other ways to help improve the model built from the user's own data. No matter which mechanism for model reuse is adopted, the whole process can be much less expensive and more efficient than building a model from scratch by herself.

[1] There are situations where the developer and owner of a trained machine learning model are different. Here, for simplicity, we do not distinguish them and assume that the developer holds all rights of the model.

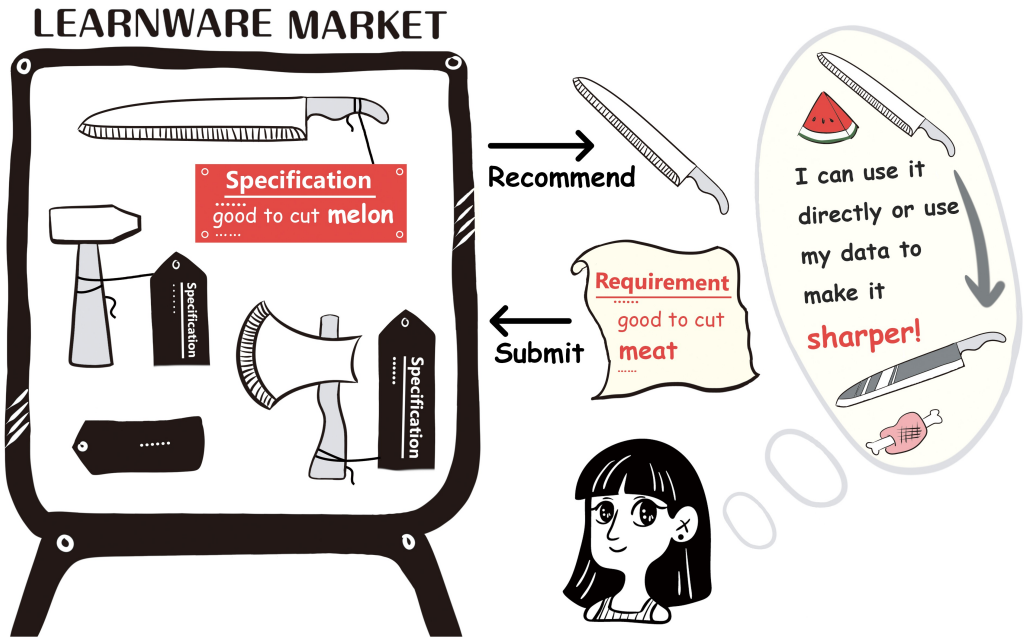

Figure 1: An analogy of learnware.

The learnware proposal offers the possibility of addressing most issues concerned in Section 1:

Lack of training data: Strong machine learning models can be attained even for tasks with small data, because the models are built upon well-performed learnwares, and only a small amount of data are needed for adaptation or refinement for most cases.

Lack of training skills: Strong machine learning models can be attained even for ordinary users with little training skills, because the users can get help from well-performed learnwares rather than building a model from scratch by themselves.

Catastrophic forgetting: A learnware will always be accommodated in the learnware market once it is accepted, unless every aspect of its function can be replaced by other learnwares. Thus, the old knowledge in the learnware market is always held. Nothing to be forgotten.

Continual learning: The learnware market naturally realizes continual and lifelong learning, because with the constant submissions of well-performed learnwares trained from di-

verse tasks, the knowledge held in the learnware market is being continually enriched.

Data privacy/proprietary: The developers only submit their models without sharing their own data, and thus, the data privacy/proprietary can be well preserved. Although one could not deny the possibility of reverse engineering the models, the risk would be too small compared with many other privacy-preserving solutions.

Unplanned tasks: The learnware market is to be open to all legal developers. Thus, there would exist helpful learnwares in the market unless a task is new to all legal developers. Moreover, some new tasks, though no developer has built models for them specially, could be addressed by selecting and assembling some existing learners.

Carbon emission: Assembling small models may offer good-enough performance for most applications; thus, one may have less interest to train too many big models. The possibility of reusing other developers' models can help reduce repetitive development. Besides, a not-so-good model for one user may be very helpful for another user. No training cost wasted.

Though the learnware proposal shows a bright future, there are much work to be done to make it a reality. The next sections will present some of our progress.

3. The Design

There are three important entities: developers, users, and the market. The developers are usually machine learning experts who produce and want to share/sell their well-performed trained machine learning models. The users need machine learning services but usually have only limited data and lack machine learning knowledge and skills. The learnware market accepts/buys well-performed trained models from developers, accommodates them in the market, and provides/sells services to users via identifying and reusing learnwares to help users tackle their present tasks.[2] The basic operation can be decomposed into two

[2]The learnware proposal implies some possible business relation among the three entities: The user who receives valuable services pays to the market, while market pays to developers according to the usage of their submitted learnwares. However, the business model is beyond the scope of this article.

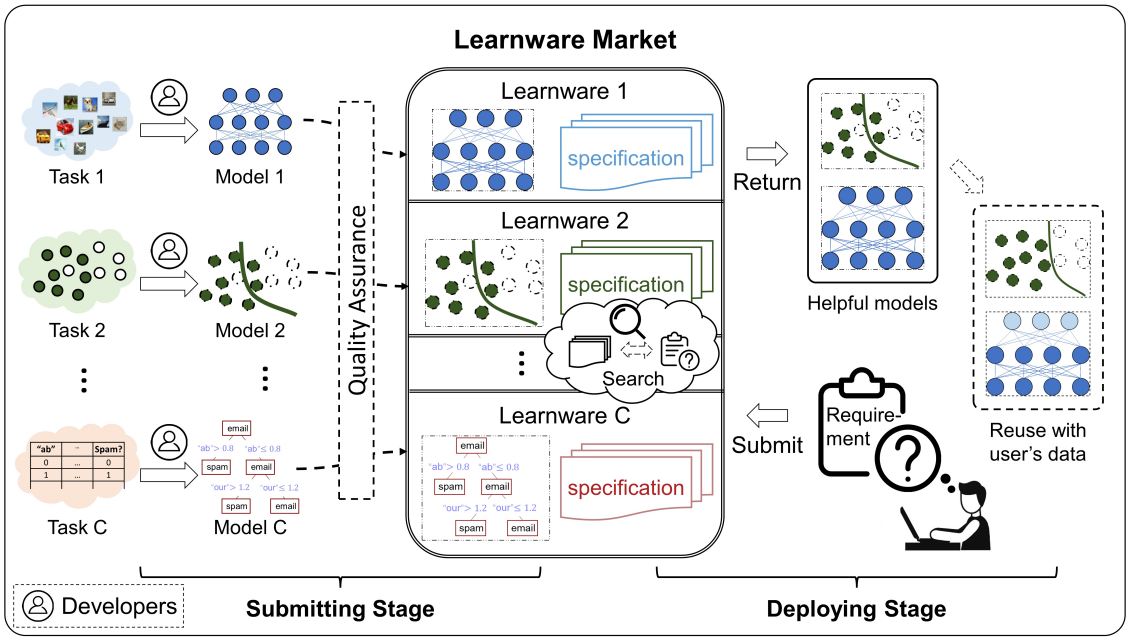

Figure 2: The learnware market and the involved two stages.

stages, as illustrated in Figure 2.

3.1. Submitting Stage

In the submitting stage, developers can spontaneously submit their trained models to the learnware market. The market will execute some quality assurance mechanism, e.g., performance validation, to decide whether a submitted model can be accepted or not. Considering a learnware market which has accommodated millions of models, how to identify potentially helpful models for a new user?

It is evidently undesired to request the user to submit her own data to the market for trials with the models, since this would be too tedious and costly, and more seriously, this could leak user's own data. It is also impossible to utilize straightforward ideas such as "measuring the similarity between the user data and the original training data of models", as the learnware proposal considers the fact that neither developers nor users would like to leak their own data due to privacy/proprietary issues (it would be easier if their data are free to the market). Thus, our design is based on the constraint that the learnware market

has access to neither the original training data of developers nor the original data of users. Besides, it is assumed that users know little about what models have been accommodated in the market.

The key of our solution lies in the *specification*, which is the core of the learnware proposal. Once the learnware market decides to accept a submitted model, it will assign to the model a specification, which conveys the specialty and utility of the model in some format, without leaking its original training data. For simplicity, consider models corresponding to functions realizing mappings from the input domain $\mathcal{X}$, to the output domain $\mathcal{Y}$, with regard to the objective `obj`; in other words, those models reside in a functional space $\mathcal{F}: \mathcal{X} \mapsto \mathcal{Y}$ *w.r.t.* `obj`. Each model has a specification. All specifications form a *specification space* where those of models that are helpful for the same tasks are nearby.

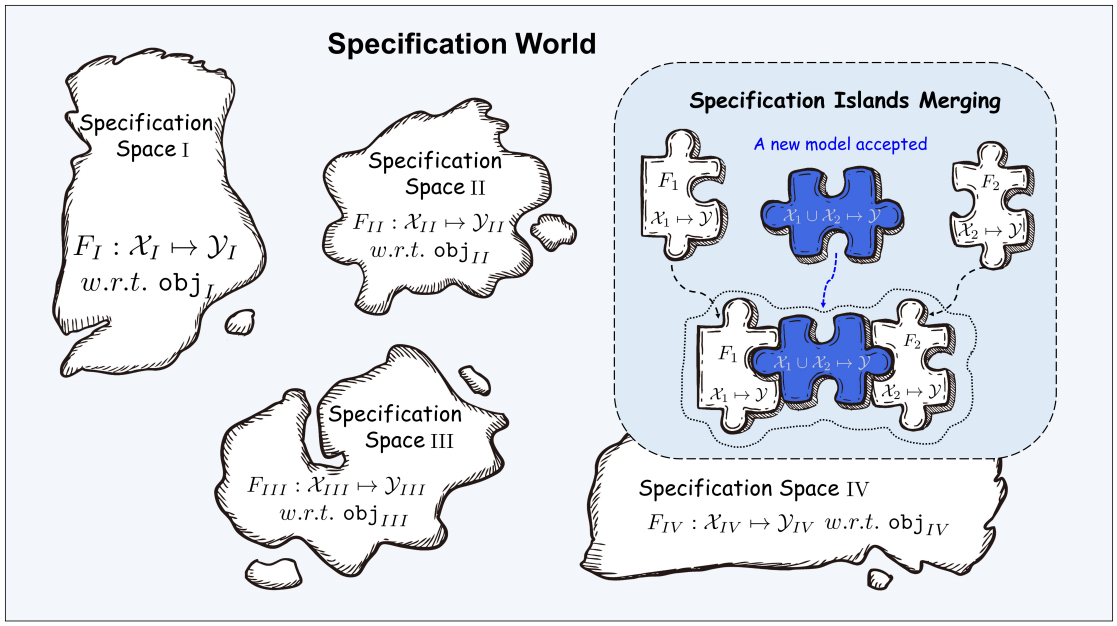

Figure 3: The learnware specification world.

In a learnware market, there will exist heterogeneous models with different $\mathcal{X}$, and/or different $\mathcal{Y}$, and/or different `obj`. If we call the specification space covering all possible models in all possible functional spaces as the specification *world* analogically, then each specification space corresponding to one possible functional space can be called a specification

island. Designing an elegant specification format covering the whole specification world, enabling all possible models to be efficiently and adequately identified is a grand challenge. Currently, we employ a practical design as follows. The specification of each learnware consists of two parts, where the first part explains which specification island the learnware locates, while the second part, to be introduced in Section 3.3, discloses at which location it resides in this island.

The first part can be realized by a string, consisting of a set of descriptions/tags given by the learnware market, about the task, input, output, and objective, etc. Then, according to the descriptions/tags provided in the user requirement, the corresponding specification island can be efficiently and accurately located. Generally, the designer of the learnware market can compose a set of initial descriptions/tags, and the set can grow when the market accepts some new models that could not be accommodated in existing functional spaces, resulting in the creation of new functional spaces and its corresponding specification islands.

The specification islands can merge into a larger one, as illustrated in Figure 3. Initially there are two islands, corresponding to functional spaces $F_1 : \mathcal{X}_1 \mapsto \mathcal{Y}$ *w.r.t.* `obj` and $F_2 : \mathcal{X}_2 \mapsto \mathcal{Y}$ *w.r.t.* `obj`, respectively. When a new model about $F : \mathcal{X}_1 \cup \mathcal{X}_2 \mapsto \mathcal{Y}$ *w.r.t.* `obj` is accepted by the learnware market, these two islands can be merged. For example, suppose there are some models on text data (i.e., on $\mathcal{X}_1$) and image data (i.e., on $\mathcal{X}_2$), respectively; once some multi-modal models involving both texts and images are accepted, these text-only and image-only models can become helpful to each other, and they appear to reside in the same extended functional space. Note that though the learnware market does not have access to the original training data of models, this is still possible because the market can have synthetic data by randomly generating some inputs and feeding them to models, and then concatenate each input with its corresponding output to construct a data set reflecting the function of a model. In principle, specification islands can be merged if there are common ingredients in $\mathcal{X}$, $\mathcal{Y}$, and `obj`. One can imagine that when all possible tasks are present, all the specification islands become connected to a non-fragmented specification world.

3.2. Deploying Stage

In the deploying stage, the user submits her requirement to the learnware market, and then the market will identify and return some helpful learnwares to the user. There are two issues, i.e., how to identify learnwares matching the user requirement, and how to reuse the returned learnwares.

The learnware market can accommodate thousands or millions of models. Different to previous machine learning studies about model reuse [18, 20] or domain adaptation [10] where all pre-trained models are assumed to be helpful, there may be only a tiny portion of learnwares helpful for the current user task. Efficiently identifying these learnwares is quite challenging, particularly when considering the fact that the learnware market has access to neither the original training data of learnwares nor the original data of current user.

With the specification design mentioned in Section 3.1, the learnware market can request user to describe her intention using the set of descriptions/tags, through a user interface or a kind of *learnware description language* to be designed in future. Based on such information, the task reduces to how to identify some helpful learnwares in a specification island. The learnware market can provide several *anchor* learnwares in the functional space corresponding to the specification island, request user to test them and return some information, and then identify potentially helpful learners based on these information, as to be explained in Section 3.3.

Once some helpful learnwares are identified and delivered to user, they can be reused in various ways. The most straightforward one is to apply the received learnware to user's own data directly; if multiple learnwares are received, they can be used to comprise an ensemble [21] for even better performance. The user can also adapt/polish the received learnware(s) by generating a model from her own training data and putting it to use together with the learnwares(s). Another possible usage is to regard each received learnware as a feature augmentor, by feeding user's data to the learnware, taking its output for each instance as an augmented feature, and then utilizing the augmented data to build the final model.

Note that some helpful learnwares may be trained from tasks that are not exactly the same as the user's current task. For example, there are cases where learnwares with different

objectives can be reused to help user's current task, such as that a model which optimizes *accuracy* can be reused to help a task optimizing AUC, by augmenting a δ function based on user's own data [9]. There are also cases where there is no single learnware that can tackle user's task as a whole, but there are multiple learnwares each can tackle a part of user's task separately. In such cases, user task can be tackled in a divide-and-conquer way, as proposed in reusable ensemble [21][pp.184]. Besides, if a very small set of potentially helpful learnwares have been returned, it is possible to reuse them collectively through measuring the utility of each model on each testing instance [5]. Sometimes user may find it difficult to express her requirement accurately. In such case, it is appealing to reuse the received learnware(s) by adapting/polishing them directly using user's own data. There are preliminary studies that might be somewhat helpful for this purpose, e.g., [9, 8, 18, 20].

3.3. Learnware Specification

The learnware specification, ideally, should express/encode important information about every model accommodated in the learnware market to enable them to be identified efficiently and adequately for future users. As mentioned in Section 3.1, our current specification design consists of two parts. The first part is a string of descriptions/tags given by the learnware market, based on information submitted by developers, aiming to locate the specification island in which a model resides. Different learnware market enterprises may employ different descriptions/tags.

The second part of the specification plays a crucial role in locating the appropriate place in the functional space $F : \mathcal{X} \mapsto \mathcal{Y}$ *w.r.t.* `obj` for the model. A recent effort is the RKME (Reduced Kernel Mean Embedding) specification [17], based on techniques of reduced set of KME (Kernel Mean Embedding) [14, 2]. The KME is a powerful technique to map a probability distribution to a point in RKHS (Reproducing Kernel Hilbert Space), whereas the reduced set reserves the ability with a concise representation which does not expose the original data.

The RKME specification [17] is based on the assumption that each learnware is a well-performed model on its own training data. Thus, identifying a suitable model for user task

can be approached by identifying a model whose original training data distribution is close to the distribution corresponding to user task. Suppose a developer is to submit a model trained from data set $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{\forall i \in [m]}$, $\boldsymbol{x}_i \in \mathcal{X}$, $\boldsymbol{y}_i \in \mathcal{Y}$. For simplicity, let $\boldsymbol{z}_i$ denote $(\boldsymbol{x}_i, \boldsymbol{y}_i)$, and the function of the model is encoded in the distribution of $\boldsymbol{z}_i$. Then, the market will generate the reduced set representation by minimizing the distance measured by the RKHS norm, as

$$\min_{\boldsymbol{\beta}, \boldsymbol{t}} \left\| \frac{1}{m} \sum_{i=1}^{m} k\left(\boldsymbol{z}_i, \cdot\right) - \sum_{j=1}^{n} \beta_j k\left(\boldsymbol{t}_j, \cdot\right) \right\|_{\mathcal{H}}^2 \tag{1}$$

where $k(\cdot, \cdot)$ is the kernel function corresponding to the RKHS $\mathcal{H}$, $n \ll m$, both decided by the learnware market. The solved $(\boldsymbol{\beta}, \boldsymbol{t})$, which offers a much more concise representation very different from the original data $\boldsymbol{z}$, will be submitted by the developer for the second part of the model specification.

In the deploying stage, if the user has many training data, the market can help her construct the RKME requirement to submit. Then, by matching the RKME specifications with the user requirement, the market can identify and return the learnware with the smallest distance in the RKHS norm. The market can also identify multiple helpful learnwares whose weighted combination of RKME specifications has the smallest distance to the user requirement. If the user does not have sufficient training data for constructing a RKME requirement, the learnware market can send several anchor learnwares to the user. By feeding her own data to these anchor learnwares, some information such as (precision, recall) or other performance indicators, can be generated and returned to the market. These information could help the market identify potentially helpful models, e.g., by identifying models that are far from anchors exhibiting poor performance whereas close to anchors exhibiting relatively better performance in the specification island.[3]

Note that in the procedures described above, neither the training data of developers nor that of users are leaked to the learnware market.

[3]Both above processes can be realized with a user interface at user's side, without leaking user's training data to the learnware market.

4. Some Theoretical Results

The RKME specification is based on RKME $\widetilde{\Phi}$, which aims to make a good representation by constructing a reduced set to approximate the empirical KME $\Phi = \int_{\mathcal{X}} k(\boldsymbol{x}, \cdot) \mathrm{d}P(\boldsymbol{x})$ of the underlying distribution. Theoretically, when the kernel function satisfies $k(\boldsymbol{x}, \boldsymbol{x}) \leq 1$ for all $\boldsymbol{x} \in \mathcal{X}$, with probability at least $1 - \delta$, we have the guarantee that [19, 11, 17]

$$\left\| \widetilde{\Phi} - \Phi \right\|_{\mathcal{H}} \leq 2\sqrt{\frac{2}{n}} + \sqrt{\frac{1}{m}} + \sqrt{\frac{2\log(1/\delta)}{m}}, \tag{2}$$

where n, m are the size of the RKME reduced set and the original data, respectively. It is known that when using characteristic kernels such as the Gaussian kernel, KME can capture all information about the distribution [15]. Besides, when the RKHS of the kernel function is finite-dimensional, RKME enjoys a linear convergence rate $O(e^{-n})$ to empirical KME [1]; even for infinite-dimensional RKHS, it has been proved constructively that RKME can enjoy $O(\sqrt{d}/n)$ convergence rate under L_∞ measure, where d is the dimension of original data [7, 12]. Therefore, the RKME is guaranteed to be a good estimation of KME and a valid representation for data distribution that encodes the ability of a trained model.

The risk on the user task can be bounded under some assumptions, such as the assumption that the distribution corresponding to the task of user matches that of a learnware, or the assumption that it can be approximated by a mixture of distributions corresponding to a set of learnwares' tasks, i.e.,

$$\mathcal{D}_u = \sum_{i=1}^{N} w_i \mathcal{D}_i \ , \tag{3}$$

where $\mathcal{D}_u$ is the distribution corresponding to user task, N is the number of learnwares and $\mathcal{D}_i$ are their corresponding distributions, $\sum_{i=1}^{N} w_i = 1$ and $w_i \geq 0$. These two assumptions are called *task-recurrent* and *instance-recurrent* assumptions in [17], respectively. Besides, assume that all learnwares are well-performed ones, i.e.,

$$\mathbb{E}_{\mathcal{D}_i} \left[\ell(\widehat{f}_i(\boldsymbol{x}), \boldsymbol{y}) \right] \leq \epsilon, \ \forall i \in [N], \tag{4}$$

where $\widehat{f}_i$ is the function corresponds to the i-th learnware, ℓ is the loss function, $\boldsymbol{y}$ is assumed to be determined by a ground-truth global function h.

Under these assumptions, recent studies have attempted to bound the risk on user task [17, 19]. Consider the task-recurrent assumption and select the learnware $(\widehat{f}_i, \widetilde{\Phi}_i)$ with the smallest RKHS distance η according to RKME, given the loss function

$$\left|\ell(\widehat{f}_i(\boldsymbol{x}), h(\boldsymbol{x}))\right| \leq U, \ \forall \boldsymbol{x} \in \mathcal{X}, \ \forall i \in [N], \tag{5}$$

we have

$$\mathbb{E}_{\mathcal{D}_u}\left[\ell(\widehat{f}_i(\boldsymbol{x}), \boldsymbol{y})\right] \leq \epsilon + U\eta + O\left(\frac{1}{\sqrt{m}} + \frac{1}{\sqrt{n}}\right) . \tag{6}$$

As for the instance-recurrent assumption and the 0/1-loss

$$\ell_{01}(f(\boldsymbol{x}), \boldsymbol{y}) = \mathbb{I}(f(\boldsymbol{x}) \neq \boldsymbol{y}), \tag{7}$$

a more general result has been attained [19]:

$$\mathbb{E}_{\mathcal{D}_u}\left[\ell_{01}(f(\boldsymbol{x}), \boldsymbol{y})\right] \leq \epsilon + R(g), \tag{8}$$

where $R(g) = \sum_{i=1}^{N} w_i \mathbb{E}_{\mathcal{D}_i}\left[\ell_{01}(g(\boldsymbol{x}), i)\right]$ represents the weighted risk of any learnware selector $g(\boldsymbol{x})$, which takes unlabeled data as input and assigns it to the proper model, $f(\boldsymbol{x}) = \widehat{f}_{g(\boldsymbol{x})}(\boldsymbol{x})$ is the final model for user task.

There are efforts trying to enable the learnware market to handle unseen jobs [19], where the user task involves some unseen parts that have never been handled by current learnwares in the market, and a more general theoretical analysis is presented based on mixture proportion estimation [13, 6].

5. A Simple Prototype

A simple prototype learnware market has been implemented for experiments, with an interface shown in Figure 4.

The market accommodates 51 models about sales forecasting. They are with different model types and trained from different data sets, though the input space, output space and objectives are the same. Thus, the specifications of these models reside in the same

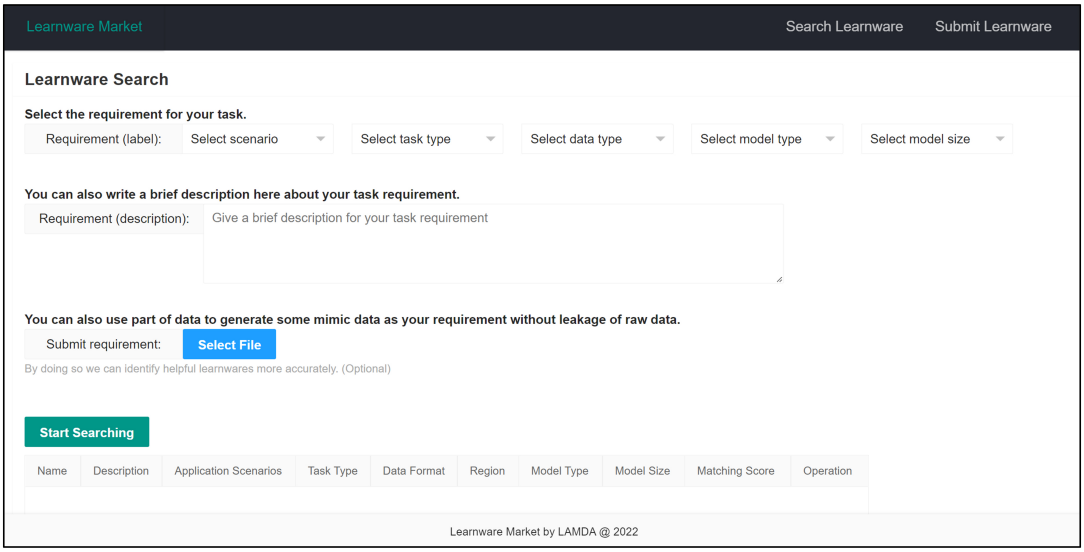

Figure 4: Interface of the simple prototype.

specification island realized with the RKME specification. Experiments are conducted to simulate the scenario where a new user, who plans to build her own sales forecasting model, is to get help from the learnware market. The following approaches are tried: to use the best-single model identified from the market directly, to use the best-three models returned from the market via ensemble averaging, to use the best-two models returned from the market together with a model trained from user's own data via ensemble averaging, respectively. Figure 5 shows the performance improvement ratio of those approaches against that of the model trained by using user's own data only.

Figure 5 exhibits that by resorting to the learnware market, the user can get much better models than simply building a model from scratch by using her own data, especially when she has only small amount of labeled data. This verifies that the learnware paradigm can offer a remedy to the lack of training data.

The prototype learnware market also accommodates 6 models about sentiment analysis on video data, and 6 models on textual data with the same output labels. A new user, who has some data involving both video and textual information, resorts to the learnware market, and the following approaches are tried: to use the best-single video-only or text-only model identified from the market directly, to use these two models together with a model trained

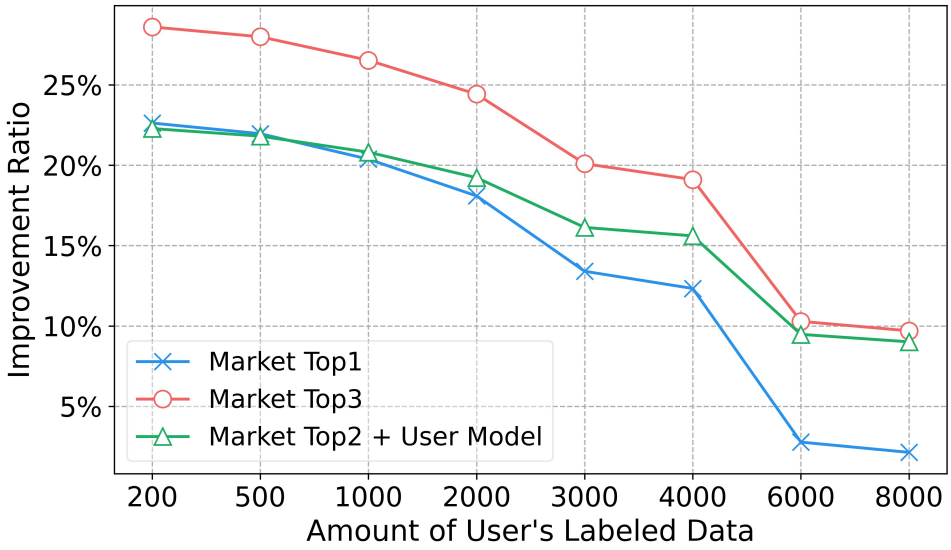

Figure 5: Sales forecasting task: Benefit from learnware market in contrast to building a model from scratch using user's own data.

from user's own data via ensemble averaging, to use the best-three models (no matter whether they are video-only or text-only) returned from the market via ensemble averaging, respectively. Figure 6 shows the performance improvement ratio of those approaches against that of the model trained by using user's own data only.

Figure 6 exhibits again that by resorting to the learnware market, the user can get much better results than using her own data only to train a model. The results also demonstrate that better performance can be achieved by exploiting user's own data to adapt/polish the models obtained from learnware market, even by simple mechanisms like ensemble averaging. It is worth noting that ignoring user's own data may lead to worse performance.

Note that such a task, i.e., building a model for sentiment analysis from multi-modal data involving both video and textual information, has never been tackled by previous developers and no model for the exact task exists in the learnware market. This verifies that some new tasks, though no developer has built model for them specifically, can be addressed by selecting and assembling some existing learners. It is interesting to see that the video-only and text-only models can help offer significant improvement, though they do not perform well by themselves since they were not developed for this task. This verifies that a

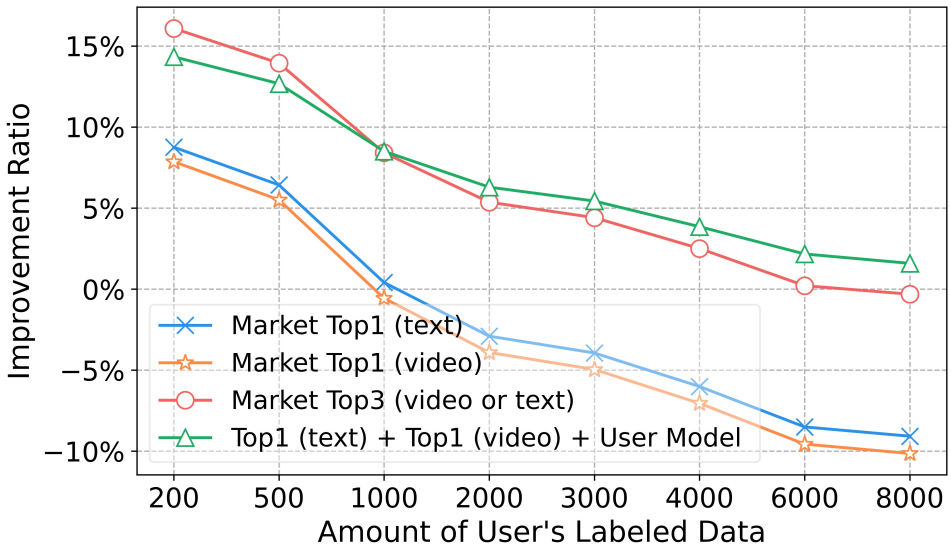

Figure 6: Sentiment analysis task: Benefit from learnware market in contrast to building a model from scratch using user's own data.

model can be helpful beyond its own purposes in training. Note that once the user model on multi-modal data is submitted to the learnware market, the two specification islands that correspond to video-only and text-only models, respectively, will merge just like that illustrated in Figure 3.

6. Conclusion and Future Issues

This article provides a brief overview of progress on *learnware*, a paradigm that seems promising to tackle many concerns of current machine learning techniques, such as the lack of training data and skills, catastrophic forgetting, continual learning, data privacy/proprietary preserving, unplanned tasks, carbon emission, etc. It would be great if, in the future, users who plan to build their own machine learning models would look into the learnware market at first rather than DIY by themselves from scratch, just like today's programmers looking for useful codes from Github or other codebases.

There are too many issues for future exploration. First, ideally, the learnware specification should enable well-performed models helpful for the same tasks to locate nearby, whereas

our current design is making models with similar functions locate nearby. Considering a user task which can be collectively tackled by several learnwares, one possibility is to tackle the task in a divide-and-conquer way and then look for helpful learnwares for each sub-task. This is to be explored in future. Second, currently the learnwares are assumed to be based on well-performed models whose function can be represented by its training data distribution, whereas in practice the models submitted by developers can be less well-performed. The quality assurance as well as its influence on the identification and reuse procedures are to be studied in future. Third, when the user does not have sufficient data for distribution estimation, as mentioned in Section 3.3, some anchor learnwares are to be sent to user. This can be realized by selecting prototype models through functional space clustering, and more interesting designs are to be explored in future. Note that our current design tries to assign each model to one location. It is, however, often the case that one model can be helpful for a variety of tasks. To enable one model to be located in multiple suitable specification islands simultaneously is another interesting future issue. Besides, to explore various ways to merge specification islands is also interesting. The learnware market also offers a platform to study the possible "intellectual ability emergence" when models in the market are allowed to have some kind of interaction. Furthermore, though there are some theoretical efforts, it is still far from establishing a thorough theoretical framework for the learnware paradigm. Besides, building an enterprise-level learnware market would be greatly helpful, not only for research platform purposes, but also for real applications.

References

[1] F. Bach, S. Lacoste-Julien, and G. Obozinski. On the equivalence between herding and conditional gradient algorithms. In *ICML*, pages 1355–1362, 2012.

[2] A. Berlinet and C. Thomas-Agnan. *Reproducing Kernel Hilbert Spaces in Probability and Statistics*. Springer, 2011.

[3] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler,

M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners. In *NeurIPS*, pages 1877–1901, 2020.

[4] M. Delange, R. Aljundi, M. Masana, S. Parisot, X. Jia, A. Leonardis, G. Slabaugh, and T. Tuytelaars. A continual learning survey: Defying forgetting in classification tasks. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 44(7):3366–3385, 2022.

[5] Y.-X. Ding and Z.-H. Zhou. Boosting-based reliable model reuse. In *ACML*, pages 145–160, 2020.

[6] M. C. du Plessis, G. Niu, and M. Sugiyama. Class-prior estimation for learning from positive and unlabeled data. *Machine Learning*, 106(4):463–492, 2017.

[7] Z. Karnin and E. Liberty. Discrepancy, coresets, and sketches in machine learning. In *COLT*, pages 1975–1993, 2019.

[8] I. Kuzborskij and F. Orabona. Stability and hypothesis transfer learning. In *ICML*, pages 942–950, 2013.

[9] N. Li, I. W. Tsang, and Z.-H. Zhou. Efficient optimization of performance measures by classifier adaptation. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 35(6):1370–1382, 2013.

[10] Y. Mansour, M. Mohri, and A. Rostamizadeh. Domain adaptation: Learning bounds and algorithms. In *COLT*, 2009.

[11] K. Muandet, K. Fukumizu, B. Sriperumbudur, and B. Schölkopf. Kernel mean embedding of distributions: A review and beyond. *Foundations and Trends in Machine Learning*, 10(1-2):1–141, 2017.

[12] J. M. Phillips and W. M. Tai. Near-optimal coresets of kernel density estimates. *Discrete & Computational Geometry*, 63(4):867–887, 2020.

[13] H. G. Ramaswamy, C. Scott, and A. Tewari. Mixture proportion estimation via kernel embeddings of distributions. In *ICML*, pages 2052–2060, 2016.

[14] B. Schölkopf, S. Mika, C. J. Burges, P. Knirsch, K. Müller, G. Rätsch, and A. J. Smola. Input space versus feature space in kernel-based methods. *IEEE Trans. Neural Networks*, 10(5):1000–1017, 1999.

[15] B. Sriperumbudur, K. Fukumizu, and G. Lanckriet. Universality, characteristic kernels and rkhs embedding of measures. *Journal of Machine Learning Research*, 12(7):2389–2410, 2011.

[16] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *NIPS*, pages 5998–6008, 2017.

[17] X.-Z. Wu, W. Xu, S. Liu, and Z.-H. Zhou. Model reuse with reduced kernel mean embedding specification. *IEEE Trans. Knowledge and Data Engineering*, page 10.1109/TKDE.2021.3086619, 2021.

[18] Y. Yang, D.-C. Zhan, Y. Fan, Y. Jiang, and Z.-H. Zhou. Deep learning for fixed model reuse. In *AAAI*, pages 2831–2837, 2017.

[19] Y.-J. Zhang, Y.-H. Yan, P. Zhao, and Z.-H. Zhou. Towards enabling learnware to handle unseen jobs. In *AAAI*, pages 10964–10972, 2021.

[20] P. Zhao, L.-W. Cai, and Z.-H. Zhou. Handling concept drift via model reuse. *Machine Learning*, 109(3):533–568, 2020.

[21] Z.-H. Zhou. *Ensemble Methods: Foundations and Algorithms*. Chapman & Hall/CRC, Boca Raton, FL, 2012.

[22] Z.-H. Zhou. Learnware: On the future of machine learning. *Frontiers of Computer Science*, 10(4):589–590, 2016.

[23] Z.-H. Zhou. A brief introduction to weakly supervised learning. *National Science Review*, 5(1):44–53, 2018.

[24] Z.-H. Zhou. Open-environment machine learning. *National Science Review*, 9(8):nwac123, 2022.