Transfer Learning in Non-Stationary Environments

Minku, Leandro

Chapter 1
Transfer Learning in Non-Stationary Environments

Leandro L. Minku

Abstract The fields of transfer learning and learning in non-stationary environments are closely related. Both look into the problem of training and test data that come from different probability distributions. However, these two fields have evolved separately. Transfer learning enables knowledge to be transferred between different domains or tasks in order to improve predictive performance in a target domain and task. It has no notion of continuing time. Learning in non-stationary environments concerns with updating learning models over time in such a way to deal with changes that the underlying probability distribution of the problem may suffer. It assumes that training examples arrive in the form of data streams. Very little work has investigated the connections between these two fields. This chapter provides a discussion of such connections and explains two existing approaches that perform online transfer learning in non-stationary environments. A brief summary of the results achieved by these approaches in the literature is presented, highlighting the benefits of integrating these two fields. As the first work to provide a detailed discussion of the relationship between transfer learning and learning in non-stationary environments, this chapter opens up the path for future research in the emerging area of transfer learning in non-stationary environments.

Key words: Transfer learning, data streams, concept drift, learning in non-stationary environments, online learning

Department of Informatics, University of Leicester, University Road, Leicester LE1 7RH, UK, e-mail: `leandro.minku@leicester.ac.uk`

1.1 Introduction

Individuals, organisations and systems have been producing large amounts of data. Such data can be used to gain insights into various areas of interest, such as businesses and processes. In particular, machine learning can be used to create models able to give insights in the form of predictions. Examples of problems involving predictions include spam detection, credit card approval, network intrusion detection, speech recognition, among many others. This chapter concentrates on machine learning for building predictive models.

In supervised learning, predictive models are created based on existing training examples of the format $(\boldsymbol{x}_i, \boldsymbol{y}_i) \in \mathcal{X} \times \mathcal{Y}$, where $\boldsymbol{x}_i$ is example i's vector of input attributes, $\boldsymbol{y}_i$ is example i's vector of output attributes, $\mathcal{X}$ represents the input space and $\mathcal{Y}$ represents the output space. For example, in the problem of predicting the effort required to develop a software project, each example i could correspond to a software project described by $(\boldsymbol{x}_i, \boldsymbol{y}_i) \in \mathcal{X} \times \mathcal{Y}$, where $\mathcal{X}$ is the 4-dimensional space of all possible team expertises, programming languages, types of development and estimated sizes, and $\mathcal{Y}$ is the 1-dimensional space of all possible software required efforts in person-hours. In the problem of predicting whether a credit card customer will default their payments, each example i could correspond to a customer described by $(\boldsymbol{x}_i, \boldsymbol{y}_i) \in \mathcal{X} \times \mathcal{Y}$, where $\mathcal{X}$ is the 5-dimensional space of all possible ages, genders, salaries, types of bank accounts and numbers of years consecutively holding a bank account, and $\mathcal{Y}$ is the 1-dimensional space of payment categories (*pay* and *default*).[1] When $\mathcal{Y}$ is a one-dimensional space, $\boldsymbol{y}_i$ can be written as y_i. This work concentrates on one-dimensional output spaces, but the ideas discussed herein could be extended to multi-dimensional output spaces.

Most supervised learning algorithms are offline learning algorithms, defined as follows:

Definition 1 (Offline Learning) *Consider a fixed training set $\mathcal{S} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^m \sim_{iid} p(\boldsymbol{x}, y)$, where $(\boldsymbol{x}_i, y_i) \in \mathcal{X} \times \mathcal{Y}$, and $p(\boldsymbol{x}, y)$ is the joint probability distribution of the problem. Offline learning consists in using the pre-defined training set $\mathcal{S}$ to build a model $f : \mathcal{X} \rightarrow \mathcal{Y}$ able to generalise to unseen examples $(\boldsymbol{x}_i, y_i) \sim p(\boldsymbol{x}, y)$, $i > m$.*

Given definition 1 , a model f created based on offline learning is appropriate for predicting the output attributes of new instances from the same joint probability distribution $p(\boldsymbol{x}, y)$ as the one underlying the training set $\mathcal{S}$. However, many real world applications involve making predictions for a target task in a given domain based on examples coming from different source tasks or domains. In such cases, the probability distributions underlying the target and sources may differ.

[1] The software required effort and credit card payment problems will be used as illustrative examples for the concepts explained in this chapter.

The need for using examples from different sources arises from the high cost or even impossibility of collecting training examples from the target task and domain. For example, when building a model to predict the effort required to develop a software project in a given company, it is typically expensive to collect labelled examples describing projects from within this company [14]. Examples of projects developed by other companies are available in existing software project repositories [19, 13], but they may come from different underlying probability distributions.

Machine learning algorithms operating in this type of scenario must be able to tackle the different source and target probability distributions. Specifically, they need to transfer knowledge from the source task / domain to the target task / domain. Algorithms designed to achieve such knowledge transfer are referred to as *Transfer Learning* (TL) algorithms.

Many real world applications pose yet another challenge to machine learning algorithms. Instead of providing a fixed training set, they provide a potentially infinite sequence of training sets $\mathcal{S} = \langle \mathcal{S}_1, \mathcal{S}_2, \cdots \rangle$, where $\mathcal{S}_t = \{(\boldsymbol{x}_t^{(i)}, y_t^{(i)})\}_{i=1}^{m_t} \sim_{iid} p_t(\boldsymbol{x}, y)$; $t > 0$ is a time step, i.e., the sequential identifier of a time moment when a new training set was received; and $m_t > 0$ is the size of the training set. Such sequence is referred to as a *data stream.* Its unbounded nature provides a clear notion of continuing time. For example, in the problem of predicting whether a credit card customer will default their payments, new examples describing the behaviour of additional customers may be received over time [24].

Data streams may suffer changes in their underlying probability distributions over time, i.e., examples drawn at different time steps may belong to different probability distributions. Such changes are referred to as *concept drifts* [10]. They can be seen as the result of (1) actual changes in the underlying data generating process or (2) insufficient, unknown or unobservable input attributes, which result in the probability distributions underlying observable attributes to change, despite the true data generating process being stationary [7]. For example, customers' defaulting behaviour may be affected by the beginning of an economic crisis. If there are no input attributes describing the presence of a crisis, the beginning of the economic crisis would be perceived as a change in the joint probability distribution underlying the observable attributes.

Machine learning algorithms for Non-Stationary Environments (NSE) must be able to adapt predictive models to concept drift. They need to maintain up-to-date predictive models reflecting the current joint probability distribution, even considering that part of the past training examples may belong to a different joint probability distribution.

As we can see from the above, both TL and learning in NSE involve training and test data that potentially come from different probability distributions. However, these two fields have evolved separately, and little work has investigated the connections between them. A discussion of the relationship between these two fields could greatly benefit future work, as TL approaches could inspire new approaches to overcome problems in learning algorithms for NSE, and vice-versa.

This chapter provides a novel discussion of the relationship between the fields of TL and learning in NSE, and of the potential benefit of using one field to improve the other. It explains two existing approaches that combine the strengths of TL and learning in NSE: Diversity for Dealing with Drifts (DDD) [24] and Dynamic Cross-company Mapped Model Learning (Dycom) [22]. The results achieved by these two approaches in the literature are briefly explained, highlighting the benefits of transferring knowledge in NSE.

This chapter is further organised as follows. Sections 1.2 and 1.3 explain TL and learning in NSE, respectively. They are not intended to provide an extensive literature review of these two fields, which can be found elsewhere [29, 7, 17, 11]. Instead, they are intended to provide definitions and examples of representative algorithms to inform and enable the discussion of the relationship between these two fields, which is provided in section 1.4. After discussing this relationship, section 1.5 discusses the potential of combining these two types of approach to improve their weaknesses. It includes an explanation of the approaches DDD [24] and Dycom [22] for TL in NSE, and of the results they have achieved in the literature. Finally, section 1.6 concludes this work.

1.2 Transfer Learning (TL)

TL has been studied by the machine learning and data mining communities for many years [28]. When defining TL, several authors rely on the distinction between the terms "domain" and "task" [28]:

Definition 2 (Domain) *Domain is a pair* $\langle \mathcal{X}, p(\boldsymbol{x}) \rangle$*, where* $\mathcal{X}$ *is the input space;* $p(\boldsymbol{x})$ *is the unconditional probability distribution function (pdf); and* $\boldsymbol{x} \in \mathcal{X}$*.*

Definition 3 (Task) *Given a Domain* $= \langle \mathcal{X}, p(\boldsymbol{x}) \rangle$*, Task is a pair* $\langle \mathcal{Y}, p(y|\boldsymbol{x}) \rangle$*, where* $\mathcal{Y}$ *is the output space;* $p(y|\boldsymbol{x})$ *represents the posterior probabilities of the output attributes;* $\boldsymbol{x} \in \mathcal{X}$*; and* $y \in \mathcal{Y}$*.*

Given existing TL approaches [29], we can consider existing work to be adopting the following definition of TL:

Definition 4 (Transfer Learning) *Consider* N *source tasks,* $Task_i = \langle \mathcal{Y}_i, p_i(y|\boldsymbol{x}) \rangle$*,* $1 \leq i \leq N$*, and their corresponding source domains,* $Domain_i = \langle \mathcal{X}_i, p_i(\boldsymbol{x}) \rangle$*. Consider also a target task,* $Task' = \langle \mathcal{Y}', p'(y|\boldsymbol{x}) \rangle$*, and its corresponding target domain,* $Domain' = \langle \mathcal{X}', p'(\boldsymbol{x}) \rangle$*, where* $\forall i$*,* $[Domain_i \neq Domain'$ *or* $Task_i \neq Task']$*. The sources are associated to fixed (labelled or unlabelled) data sets* $\mathcal{S}_i$*. The target is associated to a fixed (labelled or unlabelled) data set* $\mathcal{S}'$*. TL consists in using both the source and target data sets* *to build a target model $f' : \mathcal{X}' \rightarrow \mathcal{Y}'$ able to generalise to unseen examples of $p'(\boldsymbol{x}, y) = p'(y|\boldsymbol{x})p'(\boldsymbol{x})$. Its aim is to improve learning in comparison with algorithms that use only S'.*

The use of the source data sets $\mathcal{S}_i$, $1 \leq i \leq N$, may be direct or indirect. In the former case, relevant source examples can be selected to train the target model f'. In the latter, source data sets can be used to learn parameters of models or representations, which are in turn used to help building f'.

TL is typically useful when there is not enough target data to produce a good target predictive model. The lack of target data may result from the high cost or even impossibility of collecting target training examples, as explained in section 1.1. However, it is worth noting that TL can only be beneficial if the sources and target share some similarities. If they are too dissimilar, the use of source examples could even be detrimental to the performance of the target predictive model, depending on the TL algorithm being used [31]. This phenomenon is called *negative transfer*.

Depending on whether sources and target differ in terms of their domains or tasks, TL approaches can be categorised into two types – transductive (section 1.2.1) and inductive (section 1.2.2).

1.2.1 Transductive TL

Given definition 4 (TL), transductive TL can be further defined as follows:

Definition 5 (Transductive Transfer Learning) *Transductive transfer learning consists in transferring knowledge between different domains that share the same task. More formally, $\forall i, [\mathcal{X}_i \neq \mathcal{X}'$ or $p_i(\boldsymbol{x}) \neq p'(\boldsymbol{x})]$ and $[\mathcal{Y}_i = \mathcal{Y}'$ and $p_i(y|\boldsymbol{x}) = p'(y|\boldsymbol{x})]$.*

Transductive TL is typically used when we do not have access to labelled target examples, but we do have one or more training sets containing labelled source examples. Let's take the software required effort problem introduced in section 1.1 as an example. Consider that three software development companies c_1, c_2 and c' can develop software projects whose estimated size varies from small to large. However, company c_1 is more often involved with large software projects, company c_2 is more often involved with medium software projects, and company c' is more often involved with small software projects. So, $p_1(\boldsymbol{x}) \neq p_2(\boldsymbol{x}) \neq p'(\boldsymbol{x})$. Consider also that the three companies adopt largely the same practices. So, it is likely that $p_1(y|\boldsymbol{x}) = p_2(y|\boldsymbol{x}) = p'(y|\boldsymbol{x})$. Companies c_1 and c_2 have collected several examples of their completed software projects, including information on their required efforts. Such data was donated to the International Software Benchmarking Standards Group [13]. Company c', on the other hand, collected only the input attributes of its projects, because required efforts are expensive to collect. Therefore, company c' may wish to use transductive TL to benefit from the data collected by companies c_1 and c_2 to build a model for predicting software required effort.

In transductive TL, as the source and target tasks are the same, labelled source examples can be used to learn the target task. However, as the source and target domains are different, the sources may not cover the same regions of the input space as the target, or may not share the same input space as the target. This needs to be tackled to avoid incorrectly biasing the target predictive model.

The differences between the source and target domains can be addressed by filtering or weighting source examples, so that the ones most relevant to the target domain are emphasised. For example, Turhan et al. [39] filter source examples based on their distance to the target examples in the input space. The source examples that are closest to target examples are used to buid the target predictive model. Huang et al. [12] learn weights for the source examples so as to minimise the difference between the means of the source and target examples in a kernel Hilbert space. These weights can then be used when learning a kernel-based model for the target domain based on the labelled source examples.

Another way to deal with the differences between source and target domains is to transfer parameters that compose models or feature representations. For example, Dai et al. [5] proposed a naïve Bayes transfer classification algorithm based on Expectation-Maximisation. The parameters of a naïve Bayes model are first estimated based on a labelled source data set. Expectation-Maximisation is then used based on an unlabelled target data set to gradually converge the source parameters to the target probability distribution. Pan et al. [29] proposed to learn a transformation of the source and target input attributes into a new space, called the latent space. This transformation is learned so as to minimise the distance between the transformed source and target domains. This idea shares some similarities with Huang et al. [12]'s approach. However, Huang et al. [12] learn weights to be used with the transformed examples, whereas Pan et al. [29] learns the transformation itself. Once the transformation is learnt, the target model can be learnt based on the transformed source examples and their corresponding output attributes.

1.2.2 Inductive TL

Given definition 4 (TL), inductive TL can be further defined as follows:

Definition 6 (Inductive Transfer Learning) *Inductive transfer learning consists in transferring knowledge between different tasks. The domains may or may not be different. More formally,* $\forall i, [\mathcal{Y}_i \neq \mathcal{Y}'$ *or* $p_i(y|\boldsymbol{x}) \neq p'(y|\boldsymbol{x})]$.

As the output space or the posterior probabilities of the output attributes are different, the source examples provide no information about the possible outputs or the relationship between inputs and outputs of the target task. Therefore, a few labelled target examples are necessary to learn such information. TL approaches operating in this scenario are thus useful when the cost of acquiring labelled target examples is high, but there are some labelled target examples available. The source examples may or may not need to be labelled, depending on the learning algorithm.

Let's take again the problem of software required effort introduced in section 1.1 as an example. Consider that a given software development company c_1 donated a data set containing several examples of completed software projects and their required efforts to the International Software Benchmarking Standards Group [13]. Company c', on the other hand, has collected only a few examples of completed software projects with their required efforts, due to the high cost of collecting required efforts. Companies c_1 and c' typically conduct the same type of software projects (i.e., $p_1(\boldsymbol{x}) = p'(\boldsymbol{x})$). However, the underlying function mapping $\mathcal{X}$ to $\mathcal{Y}$ may differ between these two companies (i.e., $p_1(y|\boldsymbol{x}) = p'(y|\boldsymbol{x})$) because they adopt different practices and such practices have not been collected as input attributes. In this case, company c' may wish to perform inductive TL based on c_1's projects in order to improve its software required effort predictions.

As with transductive TL, one way to perform inductive TL is to filter or weigh source examples based on how well they are believed to match the target probability distributions. Such examples can then be used to help training the target predictive model. A very popular approach in this category is TrAdaBoost [6]. This approach extends the well known AdaBoost [33] ensemble learning algorithm to perform inductive TL in scenarios where both domains and tasks may differ between sources and target. Base models of the ensemble are trained sequentially, as in the original Adaboost. Labelled target examples are weighted based on AdaBoost's original weighting rule, i.e., target training examples have their weights increased / decreased if they are incorrectly / correctly classified by former base models. In this way, misclassified target examples are emphasised, encouraging the ensemble to learn how to classify them correctly. For source training examples, which are also required to be labelled examples, this strategy is inverted. Source training examples correctly classified / misclassified by former base models have their weights increased / decreased, because they may match the target probability distributions better / worse.

Other inductive TL approaches transfer parameters that compose models or feature representations, which can then be used with the target model. For example, at the same time as training source and target predictive models, Argyriou et al. [1] learn a feature representation that is common to the source and target domains. Learning consists in concurrently determining (1) target predictive model parameters, (2) source predictive model parameters and (3) a transformation of the input space, so that the regularised error of the source and target predictive models is minimised. The error of the source / target predictive models is calculated based on the source / target training examples. For that, both source and target training examples need to be labelled. This approach works in scenarios where both tasks and domains may differ between sources and target, but a common feature space exists among them. Different from Argyriou et al. [1], Raina et al. [30] learns the feature representation and target model separately, so that unlabelled source examples can be used. Oquab et al. [26] use the internal layers of a convolutional neural network as a generic extractor of higher level features from the source domain. For that, it requires labelled source examples. The neural network parameters representing such higher level features are then reused by the target predictive model. Their corresponding internal layers are followed by an adaptation layer, which enables the target task to be learnt. This approach considers that both tasks and domains may differ between sources and target, but relies on domain-specific pre-processing of the source input attributes to produce a domain more similar to the target one.

Some inductive TL approaches also exist to transfer knowledge between relational domains, where data can be represented by multiple relationships, such as social networks [28]. Relational domains are out of the scope of this chapter.

1.3 Learning in Non-Stationary Environments (NSE)

NSE are environments where training examples arrive in the form of data streams which may suffer concept drift. Concept drift can be defined as follows [17]:

Definition 7 (Concept Drift) *Let $p_t(\boldsymbol{x}, y) = p_t(y|\boldsymbol{x})p_t(\boldsymbol{x})$ be the joint probability distribution (concept) underlying a machine learning problem at time step t. Concept drift occurs when the joint probability distribution changes over time. More formally, if, for any time steps t and $t + \Delta$, $p_t(\boldsymbol{x}, y) \neq p_{t+\Delta}(\boldsymbol{x}, y)$, then concept drift has occurred.*

Concept drifts can be either the result of changes in the actual data generating process, or simply perceived, rather than actual changes. As explained by Ditzer et al. [7], the latter case can be "caused by insufficient, unknown, or unobservable attributes, a phenomenon known as 'hidden context' [42]". In this case, there is a stationary data generating process, but it is hidden from the machine learning algorithm, which will perceive it as non-stationary. Therefore, this work will refer to learning in both cases as learning in NSE.

From definition 7, we can see that concept drift may involve changes in $p(y|\boldsymbol{x})$, $p(\boldsymbol{x})$ or both. This leads to the following widely used additional definitions:

Definition 8 (Real Concept Drift) *A concept drift is referred to as a real concept drift if it involves changes in $p(y|\boldsymbol{x})$. More formally, if, for any time steps t and $t+\Delta$, $p_t(y|\boldsymbol{x}) \neq p_{t+\Delta}(y|\boldsymbol{x})$, then a real concept drift has occurred.*

For example, in the credit card payment problem introduced in section 1.1, an economic crisis may cause customers that used to pay their bills in the past to start defaulting their payments, representing a change in $p(y|\boldsymbol{x})$.

Definition 9 (Virtual Concept Drift) *A concept drift is referred to as a virtual concept drift if it only involves changes in $p(\boldsymbol{x})$. More formally, let t and $t+\Delta$ be two time steps where $p_t(\boldsymbol{x}) \neq p_{t+\Delta}(\boldsymbol{x})$. If $p_t(y|\boldsymbol{x}) = p_{t+\Delta}(y|\boldsymbol{x})$, then the differences in the probability distributions between t and $t+\Delta$ represent a virtual concept drift.*

For example, in the credit card payment problem introduced in section 1.1, a given credit card company may start receiving and accepting more credit card applications from younger customers, leading to a change in the problem's $p(\boldsymbol{x})$.

Concept drifts are also frequently categorised with respect to their speed (number of time steps taken for a change to complete), severity (how large the changes in the probability distributions are), recurrence (whether the concept drift takes us to a previously seen concept) and periodicity (whether concept drifts occur periodically) [38, 23, 7, 17].

In particular, it is worth noting that $p(\boldsymbol{x}, y)$ may suffer several small changes between a number of consecutive time steps before becoming stable. Some authors refer to that as a single gradual concept drift [38], whereas others refer to that as a sequence of concept drifts of low severity [23]. This is distinguished from an abrupt or sudden concept drift, which is an isolated concept drift that takes a single time step to complete. The term "gradual concept drift" can also be used to describe a single concept drift that takes several time steps to complete because the old and new joint probability distributions are active concurrently for a given period of time [23, 2]. In this scenario, the chances of an example being drawn from the old / new joint probability distribution gradually reduce / increase, until the new joint probability distribution completely replaces the old one. Certain data streams may also continuously suffer concept drifts, i.e., they may not have any significant period of complete stability.

Based on existing work on learning in NSE [7, 17, 11], we can consider existing approaches for learning in NSE to be adopting the following definition:

Definition 10 (Learning Algorithms for Non-Stationary Environments) *Consider a process generating a data stream $\mathcal{S} = \langle \mathcal{S}_1, \mathcal{S}_2, \cdots \rangle$, where $\mathcal{S}_t = \{(\boldsymbol{x}_t^{(i)}, y_t^{(i)})\}_{i=1}^{m_t} \sim_{iid} p_t(\boldsymbol{x}, y)$; $m_t > 0$ is the size of the training set received at time step t; $(\boldsymbol{x}_t^{(i)}, y_t^{(i)}) \in \mathcal{X} \times \mathcal{Y}$; and $p_t(\boldsymbol{x}, y)$ is the joint probability distribution of the problem at time step t. Consider that at a current time step t, we are given access to a model $f_{t-1} : \mathcal{X} \to \mathcal{Y}$ created based on past examples from $\mathcal{S}$, a new training set $\mathcal{S}_t \in \mathcal{S}$ and possibly a set $\mathcal{S}_{past}$ containing* *a limited number of past examples from $\mathcal{S}$. Learning in non-stationary environments aims at creating an updated model $f_t : \mathcal{X} \rightarrow \mathcal{Y}$ able to generalise to unseen examples of $p_t(\boldsymbol{x}, y)$, based on the given information.*

The following observations must be made when using definition 10:

- The explicit index t in the probability distributions distinguishes these algorithms from algorithms for stationary environments [7], as it takes the possibility of concept drift into account.
- A model f_t may be an ensemble model, possibly composed of several predictive models created with data from previous training sets.
- The size of a training set can be one, i.e., the training set may consist of a single example.
- Many algorithms discard past training sets once they are processed. However, some approaches make use of a limited number of examples from past training sets, as will be explained in sections 1.3.1 and 1.3.2. When used, the number of past examples must be limited, given that data streams have potentially infinite size. The unbounded nature of data streams means that it is infeasible to always store all past examples for future access.

Learning algorithms for NSE may process data streams chunk-by-chunk (section 1.3.1) or example-by-example (section 1.3.2). Most of these algorithms are prepared to deal with changes that affect the suitability of the learnt decision boundaries. Different from real concept drifts, virtual concept drifts do not affect the true decision boundary of the problem. However, they may still affect the suitability of the learnt decision boundary [41]. Therefore, most existing algorithms are applicable to data streams with both types of concept drifts, despite having different strengths and weaknesses depending on the context.

It is also worth noting that, even though in theory each training example from a given data stream could potentially come from a completely different probability distribution, it would be impossible for learning algorithms to build well-performing predictive models in such scenario. In practice, it would be rather unlikely that a given learning problem continuously suffers very large changes. For instance, in the problem of predicting whether credit card customers will default their payments, it would be rather unlikely that the defaulting behaviour of customers erratically changes all the time. Therefore, most learning algorithms for NSE implicitly assume that there will be some periods of relative stability, or that there are very frequent / continuous changes, but such changes are frequently small.

1.3.1 Chunk-by-Chunk Approaches

Given definition 10, chunk-based learning algorithms for NSE can be defined as follows:

Definition 11 (Chunk-Based Learning for Non-Stationary Environments) *Chunk-based learning algorithms for non-stationary environments are algorithms that perform learning in non-stationary environments by processing the data stream chunk-by-chunk, where the chunk size is larger than 1. These algorithms need to wait for a whole chunk of examples to become available before learning it.*

Intuitively, a chunk would be equivalent to a training set $\mathcal{S}_t$ provided by the data stream. However, it is also possible to set the chunk size in such a way that they are not equivalent. For instance, a chunk may contain more than one training set within it, or a given training set may be broken down into different chunks. Even though the size of each chunk could potentially be very small, it is typically implicitly assumed that the size is set to be large enough for a predictive model trained only on it to be better than random guess.

Most chunk-based learning algorithms for NSE are ensemble learning algorithms whose predictions are the weighted average or weighted majority vote among the predictions of their base models. The weights enable the ensemble to emphasise the base models most appropriate to the current concept. These ensembles typically perform learning as follows [17, 40, 35, 9]:

1. Train an initial base model using the first chunk of training examples.
2. For each new chunk, do:
    a. use this chunk to evaluate each predictive model that composes the ensemble, based on a given performance measure;
    b. assign a weight to each predictive model based on its performance calculated above;
    c. create a new predictive model using this chunk;
    d. add the new predictive model to the ensemble if the maximum ensemble size has not been reached; otherwise, replace an existing predictive model by the new one;
    e. discard the current chunk.

One of the problems of such approaches is that they are sensitive to the chunk size. A too small chunk size means that there are not enough examples to learn a good predictive model. A too large chunk means that a single chunk may contain examples from different joint probability distributions, resulting in the inability to deal with concept drifts adequately.

Some chunk-based approaches try to reduce sensitivity to the chunk size. For example, Scholz and Klinkenberg [34] allow a new chunk to be used to update an existing predictive model, rather than always creating a new model for each new chunk. This enables chunk sizes to be small enough without necessarily hurting predictive performance. Some chunk-based approaches also enable a limited number of examples from past chunks to be reprocessed. For example, Chen and He [4] preserves certain minority class examples seen in past chunks in order to deal with class imbalanced problems.

1.3.2 Example-by-Example Approaches

Given definition 10, example-by-example learning algorithms for NSE can be defined as follows:

Definition 12 (Example-by-Example Learning for Non-Stationary Environments) *Example-by-example learning algorithms for non-stationary environments are algorithms that perform learning in non-stationary environments by processing the data stream example-by-example. These algorithms can update the predictive model whenever a new training example is received.*

The simplest type of example-by-example algorithms are algorithms that maintain a sliding window over the data stream [18]. They build a new classifier to replace the old one whenever the window slides, by making use of the examples within the window. Similar to most chunk-based algorithms, sliding window algorithms are also sensitive to the window size. They assume that the size must be large enough to produce a well-performing predictive model, but small enough not to delay adaptation to concept drifts due to examples belonging to past concepts being within the window.

Another type of example-by-example algorithms are *online learning algorithms*, which process each training example separately upon arrival and then immediately discard it. The fact that each training example is immediately discarded leads to significant differences between the mechanisms that typically underlie these algorithms and other example-by-example or chunk-by-chunk algorithms. Chunk-based or sliding window algorithms typically use offline learning algorithms to learn each new chunk or window of examples. In many cases, this involves training a new model using solely the examples within the new chunk or window [17]. This would lead to poor predictive performance if each chunk or window contained a single training example. Moreover, offline learning algorithms often requite iterating through training examples several times. Therefore, there is frequently an implicit assumption that training examples from a chunk or window can be re-processed several times before the chunk or window is discarded. Online learning algorithms are much stricter – each training example must be discarded before a new training example is used for training. They are therefore more suitable for applications with very tight time and/or memory constraints, such as applications where the rate of incoming data is very large or certain embedded systems.

Many of the online learning algorithms for NSE use concept drift detection methods to actively detect concept drift. This is typically done by monitoring an online learning model for stationary environments, e.g., naive Bayes [3] or Hoeffding tree [8]. An example of well-known concept drift detection method is Gama et al. [10]'s. This method tracks the error of an online learning model over time. If this error significantly increases, a concept drift detection is triggered. Other authors proposed different concept drift detection methods by monitoring different quantities. For instance, Baena-Garcia et al. [2]

monitor the distance between misclassifications over time, whereas Ross et al. [32] monitor the exponentially weighted average of the errors. A typical way to deal with concept drifts once they are detected is to reset the online learning model, so that it can start learning the new concept from scratch [10, 2, 32]. However, this strategy is sensitive to false positive drift detections (a.k.a., false alarms). Strategies such as creating a new online learning model upon concept drift detection, but maintaining old online learning models in case they turn out to be still useful, can help to improve robustness to false positive drift detections [24, 25].

Some online learning algorithms deal with concept drift passively, i.e., they do not use any concept drift detection method. A well known example is the Dynamic Weighted Majority (DWM) algorithm [16]. This algorithm maintains an ensemble of online learning models, each associated to a weight. For classification problems, the prediction given by the ensemble is the weighted majority vote among the predictions given by the base learners. When a new training example becomes available, each online learning model is asked to predict the output attribute of the example before learning it. If a given online learning model misclassifies the training example, its weight is reduced. In this way, the ensemble can automatically emphasise the online learning models most suitable to the current concept. Online learning models whose weight is below a certain threshold are deemed outdated and are thus eliminated. New online learning models can also be created when the ensemble as a whole misclassifies a training example. In this way, new models can be created to learn new concepts from scratch.

1.4 The Relationship Between TL and Learning in NSE

This section discusses the similarities (section 1.4.1) and differences (section 1.4.2) between TL and learning in NSE.

1.4.1 Similarities

As we can see from definitions 4 and 10, TL and learning in NSE both involve training and test data that potentially come from different probability distributions. In TL, training examples coming from different sources may have different domains and tasks than the target test data. In learning in NSE, past training examples may come from a different joint probability distribution from that underlying the current test data.

In particular, transductive TL is concerned with sources and targets that share the same task, but have different domains. This means that sources and target differ in terms of their unconditional pdf $p(\boldsymbol{x})$. This is similar to learning under virtual concept drifts, which also represent changes in $p(\boldsymbol{x})$. In inductive TL, sources and targets have different tasks, i.e., they differ in terms of the posterior probabilities of the classes $p(y|\boldsymbol{x})$. This is similar to learning under real concept drifts, which also represent changes in $p(y|\boldsymbol{x})$. Moreover, sources and targets may or may not differ in terms of $p(\boldsymbol{x})$ in inductive TL. This is similar to real concept drifts, which may or may not involve changes in $p(\boldsymbol{x})$.

The similarities above translate into similarities in the approaches proposed to perform TL and learning in NSE. As explained in sections 1.3.1 and 1.3.2, many approaches for learning in NSE are ensemble approaches that maintain weighted predictive models trained on data from different periods of time. So, each predictive model could potentially represent a different joint probability distribution. In TL terms, they could be seen as representing different sources. These approaches could arguably be seen as a form of inductive TL. They allow knowledge from different (source) joint probability distributions to possibly help making predictions for a given (target) concept. Moreover, the fact that predictive models are weighted based on how well they match the current concept has strong resemblance to approaches such as TrAdaBoost [6], which weigh examples from different sources based on how well they match the target joint probability distribution.

TL approaches could also potentially be seen as transferring knowledge from the past to the present. This is because different periods of time of a given data stream could be seen as different sources, which may have different domains and/or tasks from the present (target) data. Therefore, both TL and learning in NSE could be seen as trying to create good predictive models to a given present time.

Table 1.1 summarises the similarities between TL and learning in NSE.

Table 1.1 Similarities Between TL and Learning in NSE

TL	Learning in NSE
Test data come from a different joint probability distribution from that underlying (part of) the training data	Test data may come from a different joint probability distribution from that underlying (part of) the past training data
Transductive TL deals with changes in $p(\boldsymbol{x})$	Virtual concept drifts consist in changes in $p(\boldsymbol{x})$
Inductive TL deals with changes in $p(y\|\boldsymbol{x})$	Real concept drifts consist in changes in $p(y\|\boldsymbol{x})$
Inductive TL may deal with changes in $p(\boldsymbol{x})$	Real concept drifts may involve changes in $p(\boldsymbol{x})$
TL tries to use data from different sources to build a target model	NSE approaches that use past predictive models could be seen as using knowledge from different sources to build a target model
TL could be used to transfer knowledge from the past in order to perform well in the present	Learning in NSE consists in creating up-to-date predictive models that perform well in the present

1.4.2 Differences

Despite the strong similarities between TL and learning in NSE presented in section 1.4.1, there are also significant differences. The main difference is that TL has no notion of continuing time, as explained by Ditzler et al. [7] and elucidated by definition 4 (TL). In particular, TL approaches currently assume that there are pre-existing source and target data sets coming from fixed joint probability distributions. Even if the sources are used to represent data from different past periods of time and the target is used to represent a given present period of time, this would still capture only a fixed snapshot of the environment. The *continuing* nature of time captured by data streams, which is a fundamental aspect of learning in NSE (definition 10), is not considered.

The consequence of that is that TL approaches are not designed to process additional data over time. Therefore, they cannot automatically cope with concept drifts that may affect the present and cause the current target model to become obsolete. In order to transfer knowledge across time, these approaches require us to know beforehand which past and present periods of time represent a given source / target. Therefore, concept drifts resulting in a change in target (with the previous target becoming a source) would need to be manually identified and the whole TL approach re-run from scratch.

TL approaches also do not make provision for processing data with gradual concept drifts, where probability distributions slowly change until they become stable, or where there are two different probability distributions concurrently active before the concept drift completes. As each source and target should come from a fixed joint probability distribution in TL, it is unclear what to do with transitional periods when trying to transfer knowledge from the past to the present. Examples produced during such periods may need to be manually discarded. This issue becomes even more significant for real world data streams presenting continuous changes, i.e., whose underlying joint probability distribution is always changing from one time step to another. TL approaches are not prepared to deal with this type of problem.

Moreover, TL typically requires past sources to be re-processed several times. For example, TrAdaBoost [6] creates predictive models for its ensemble sequentially, and requires iterating over all source and target examples again for each new predictive model being created. The process is similar to AdaBoost [33], which is an offline ensemble learning algorithm. Argyriou et al. [1]'s feature learning requires iterating over all sources and target several times until a convergence criterion is reached. Given the unbounded nature of data streams (they are potentially infinite), this is infeasible for learning in NSE.

It is also worth mentioning that the time order between sources representing different past periods of time would not be taken into account by TL approaches, even if they were trying to transfer knowledge across time. This is because they do not distinguish between the moment in time where different past sources have been produced. However, this is a less significant issue in the context of NSE than the absent notion of continuing time itself. Even though some NSE approaches take the age of past predictive models into account (e.g., by eliminating older models), this is not necessarily a good strategy to learn in NSE [15].

Another difference between TL and learning in NSE is that TL is explicitly concerned with using different sources to create a better target model than one produced using only the target data. Even though the sources come from different probability distributions from that of the target, they are expected to be useful and are exploited. For example, many TL approaches try to transform the input space of the source and target into a feature space where they become more similar [1, 30, 29], as discussed in section 1.2. Others try to find out which source examples match the target probability distribution well, even though the source as a whole is known to follow a different probability distribution from that of the target [6].

Learning in NSE, on the other hand, is concerned with creating an appropriate predictive model for the current concept. Even though models representing the past can be used for that, attempting to use past knowledge to help learning a new concept is not a requirement. For example, many NSE approaches delete old predictive models once a concept drift is detected [10, 2, 32], without even trying to check whether such past models could be somehow helpful for building a new model. Even when models representing different periods of time are used by ensemble approaches for NSE [17, 40, 35, 9, 34], the ultimate goal of these approaches is to identify when the past models represent concepts that match the current concept well, so that they can be used in the present. Although past models representing somewhat different concepts may end up being used and result in possible benefits, this is different from deliberately trying to make use of source models / data when we know that they do come from different probability distributions, as done by TL approaches.

In addition, the sources used by TL do not necessarily need to come from the same data generating process. For instance, in the problem of software effort estimation, training examples may be acquired from different companies than the target one. Learning in NSE assumes that all training examples come from the same data generating process, even though there may be concept drift. Therefore, learning in NSE approaches are not prepared to benefit from different data generating process. In particular, they can process a single data stream over time.

Yet another difference is that TL explicitly considers different input and output spaces, as illustrated by definition 4. Learning in NSE could potentially involve changes in the input and output spaces, as they are intrinsically related to changes in $p(\boldsymbol{x})$ and $p(y)$, which compose the joint probability distribution of a problem. For instance, Sun et al. [36] proposed a NSE approach that explicitly takes class evolution (appearance, disappearance and re-occurrence of classes) into account. However, most existing NSE work does not explicitly tackle changes in the input and output space.

Table 1.2 summarises the differences between TL and learning in NSE.

Table 1.2 Differences Between TL and Learning in NSE

TL	Learning in NSE
No notion of continuing time	Continuing time is a fundamental aspect
Not designed to automatically process incoming data over time	Designed to automatically process incoming data over time
Unable to automatically cope with changes in the present	Designed to automatically cope with changes in the present
No provision for processing examples from transitional periods between different joint probability distributions	Can process examples from transitional periods between different joint probability distributions
Typically has to re-process examples several times, being unsuitable for potentially infinite data streams	Only requires repeated access to a limited number of past examples, being feasible for potentially infinite data streams
Unable to distinguish between the time order of different past sources	Can potentially take the age of different past models into account
Aims to use sources with different probability distributions to improve target model	Aims to create an up-to-date predictive model, without necessarily using past data or models to help learning a new concept
Sources may come from different data generating processes	All training examples come from the same data generating process
Explicitly considers sources and targets with different input and output spaces	Changes in the input and output spaces are usually not explicitly considered

1.5 The Potential of Transfer Learning in NSE

Sections 1.4.1 and 1.4.2 show that, even though there are strong similarities between current work on TL and learning in NSE, there are also significant differences. These differences lead to limitations that prevent these approaches to effectively deal with certain types of problem, or that cause these approaches to potentially miss useful knowledge that could lead to better predictive performance.

In particular, TL is not designed to automatically process incoming data over time and is typically unable to deal with potentially infinite data streams. It cannot automatically cope with changes to the present and has no provision to process examples from transitional periods between concepts. However, several real world problems that could potentially benefit from TL provide data streams rather than fixed training sets. Let's take the example of the software required effort problem mentioned in section 1.1. Software development companies develop additional software projects over time, which could be provided as training examples in the form of a data streams. Such data streams are likely to present concept drift, given that, e.g., the practices adopted by a software company and the type of projects that it develops may change over time. So, a TL approach able to deal with data streams would be desirable.

Meanwhile, NSE approaches are potentially wasting useful knowledge from past concepts that could be used to help learning a new concept. As concept drifts could lead to new concepts that share some similarities with respect to old concepts, it would be desirable to have NSE approaches able to transfer knowledge from old concepts to better learn new concepts. Moreover, NSE approaches cannot benefit from examples coming from different data generating processes. However, several learning problems that operate in NSE could benefit from data coming from different data generating processes. In particular, as explained in the previous paragraph, the software required effort problem introduced in section 1.1 is actually a problem that both operates in NSEs and could benefit from data coming from different data generating processes (i.e., different companies).

By combining TL and learning for NSE, the individual limitations of these approaches could be overcome. For instance, two approaches called Dynamic Cross-company Learning (DCL) [21] and Dynamic Cross-company Mapped Model Learning (Dycom) [22] make use of data from different source data generating processes in order to improve predictive performance in a target non-stationary data stream. The former can still only benefit from knowledge from a different data generating process when it matches the current target concept well. However, the latter can transform knowledge from sources with different tasks and domains into useful knowledge to learn a new concept in NSE. This approach enables TL to process data streams in NSE. It can be seen as using ideas from learning in NSE to make TL aware of continuing time.

Another online learning approach called Diversity for Dealing with Drifts (DDD) [24] attempts to use knowledge acquired from a past concept's $p(\boldsymbol{x})$ and $p(y|\boldsymbol{x})$ to aid the learning of a new concept. This helps it to improve predictive performance in the presence of gradual or not severe concept drifts. A recent chunk-based approach [37] called Diversity and Transfer based Ensemble Learning (DTEL) uses knowledge acquired from past concepts' $p(\boldsymbol{x})$ to aid the learning of a new concept. These approaches can be seen as getting inspiration from TL to improve learning in NSE.

The combination of TL and learning for NSE environments could lead to a whole new range of machine learning approaches, forming a new combined area of TL in NSE that overcomes the limitations of these areas in isolation. Sections 1.5.1 and 1.5.2 explain Dycom [22] and DDD [24], as well as the results achieved by these approaches in the literature [22, 24], highlighting the potential benefit of TL in NSE.

1.5.1 Dynamic Cross-company Mapped Model Learning (Dycom)

Dycom [22] is a regression approach for online inductive transfer learning in NSE where both tasks and domains may differ between the sources and target. It assumes that a good number of labelled source training examples are available from different data generating processes, but that the specific source generating a given example is unknown. It considers that collecting labelled training examples from the target data generating process is expensive, and that only very few of such examples can be acquired over time. Therefore, transferring knowledge from different sources may help to learn a better target predictive model. Such TL should be able to tackle concept drift, given that the target is non-stationary.

The approach is illustrated in figure 1.1. It separates the set of source training examples into M partitions according to their similarities. This could be done based on a clustering approach [20] or on prior knowledge [22]. Each of the M partitions is used to create a source predictive model (source models 1 to M in figure 1.1). In this version of Dycom, the source models are trained based on offline learning. However, this approach could be extended to use source data streams.

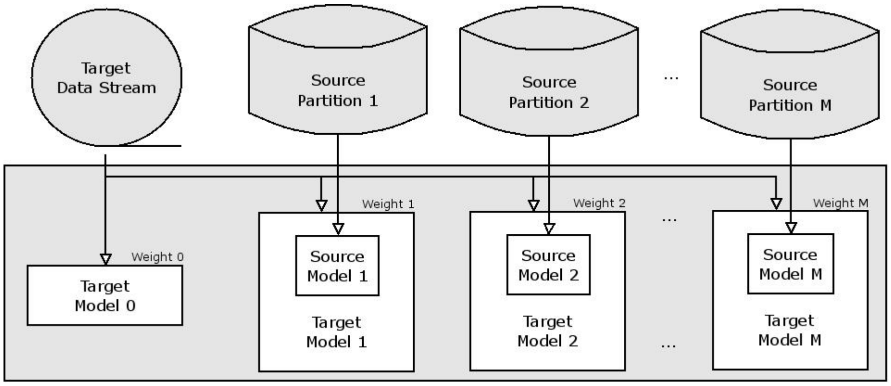

Fig. 1.1 Dycom approach for TL in NSE. Arrows represent flow of information.

A target data stream is used to train a target predictive model (target model 0 in figure 1.1) based on an online supervised learning algorithm. This predictive model is expected to be weak and perform poorly, due to the small number of labelled target training examples. However, depending on how long the periods of stability are, it may happen that, over time, it will be trained with enough examples to perform well. Therefore, it is worth building and maintaining this model.

The target data stream is also used to learn M functions $g_t^{(i)} : \mathcal{Y} \rightarrow \mathcal{Y}$ able to map the predictions made by each of the M source models i into the target concept at time t. These functions thus compose M target models (target models 1 to M in figure 1.1). Dycom assumes that, as the source training examples have been separated into similar partitions, there exists a reasonably linear relationship between each of the source models and the target concept, which can be learnt based on a few labelled target training examples. This relationship is described as follows:

$$g_t^{(i)}(f^{(i)}(\boldsymbol{x})) = f^{(i)}(\boldsymbol{x}) \cdot b_t^{(i)},$$

where $g_t^{(i)}$ is the function to map source model $f^{(i)}$'s predictions to the target concept, $b_t^{(i)}$ is a learnt parameter, and $\boldsymbol{x} \in \mathcal{X}$ are the input attributes of the target example being predicted.

Parameter $b_t^{(i)}$ is learnt in an online manner based on the following equation:

$$b_t^{(i)} = \begin{cases} 1, & \text{if no target training example} \\ & \text{has been received yet;} \\ \dfrac{y_t}{f^{(i)}(\boldsymbol{x}_t)}, & \text{if } (\boldsymbol{x}_t, y_t) \text{ is the first} \\ & \text{target training example;} \\ lr \cdot \dfrac{y_t}{f^{(i)}(\boldsymbol{x}_t)} + (1 - lr) \cdot b_{t-1}^{(i)}, & \text{otherwise,} \end{cases} \tag{1.1}$$

where $(\boldsymbol{x}_t, y_t)$ is the current target training example, and $lr \in (0, 1)$ is a pre-defined smoothing factor.

As explained in [22], the mapping function performs a direct mapping $b_t^{(i)} = 1$ if no target training example has been received yet. When the first target training example is received, $b_t^{(i)}$ is set to the value $y_t/f^{(i)}(\mathbf{x}_t)$. This gives a perfect mapping of $f^{(i)}(\boldsymbol{x}_t)$ to the target concept for the example being learnt, as $f^{(i)}(\mathbf{x}_t) \cdot b_t^{(i)} = f^{(i)}(\mathbf{x}_t) \cdot y_t/f^{(i)}(\mathbf{x}_t) = y_t$. For all other target training examples received, exponentially decayed weighted average with smoothing factor lr is used to set $b_t^{(i)}$. Higher lr will cause more emphasis on the most recent target training examples and higher adaptability to changing environments, whereas lower lr will lead to a more stable mapping function. So, the weighted average allows learning mapping functions that provide good mappings based on previous target training examples, while allowing adaptability to changes that may affect the target.

Dycom's predictions for new target examples are based on the weighted average of all target models. The weights are initialised to 1 and are updated whenever a new target training example becomes available as follows. The winner model is set to be the target model whose prediction for the current target training example is the most accurate. All other models are the looser models, and have their weights multiplied by a pre-defined factor $\beta \in (0, 1]$. All weights are then normalised so that they sum up to 1. The weights of the target models thus allow more emphasis to be placed on the models that are currently more accurate. In particular, if the target model that does not use source knowledge (target model 0) is inaccurate, its corresponding weight will be low, so that its predictions will not hinder Dycom's predictive performance.

This approach has been evaluated in the context of software effort estimation, where we are interested in predicting the effort for projects from a given target company. In this problem, we have access to training examples from other source companies, despite the fact that acquiring labelled training examples from within the target company is expensive. The problem of software effort estimation gave the name Dynamic *Cross-company* Mapped Model Learning to this approach, but Dycom is also applicable to other problems, as we can see from its description above.

Experiments on five databases containing software development projects [22] using regression trees as the base learner show that Dycom achieved similar or better predictive performance while requiring *10 times less target training examples* than a target model created using only target (no source) training examples. This highlights the benefits of using TL in NSE when the cost of acquiring labelled target training examples is high. Moreover, the experiments showed that Dycom's mapping functions can be visualised so that project managers can see how the relationship between the efforts required by their company and other companies changes over time. This could be potentially used to aid the development of strategies to improve a company's productivity. Therefore, the insights provided by TL in NSE could go beyond mere predictions.

1.5.2 Diversity for Dealing with Drifts (DDD)

DDD is a classification ensemble approach for online learning in NSE. It was not originally described as an approach for TL in NSE. However, this is essentially what it does. This approach is based on two key findings [24]:

- Knowledge from a past concept can be useful when learning in the presence of concept drifts that occur gradually or are not severe. When a gradual concept drift occurs, knowledge from the past concept remains useful for a certain period of time, until the drift completes. And, more importantly, if there is a non-severe concept drift (i.e., if the old and new concepts share some similarities), knowledge from the old concept could be useful to help learning the new concept. This is in line with the fact that TL approaches can be beneficial if the source and target share some similarities, as explained in section 1.2. On the other hand, if the concept drift occurs very fast and is very severe, the old and new concepts will not share enough similarities for knowledge from the old concept to help learning the new concept.

- Learning a given concept using very highly diverse ensembles will cause them to perform poorly on this concept. However, such weak learning enables these ensembles to quickly adapt to a new concept, if this new concept shares similarities with the given concept. In essence, this enables knowledge of a given concept to be transferred to a new concept in order to improve predictive performance in NSE.

Therefore, DDD maintains online learning ensembles with different diversity levels in order to achieve robustness to different data stream conditions (i.e., to different types of concept drift or periods of stability). Figure 1.2 illustrates its behaviour. Before a concept drift is detected, a low diversity ensemble is used both for learning incoming training examples and for making predictions. A very highly diverse ensemble is used for learning, but not for predictions. This is because this ensemble is expected to be weak and perform poorly in the current concept, but may become helpful if there are gradual or not severe concept drifts, based on the findings above.

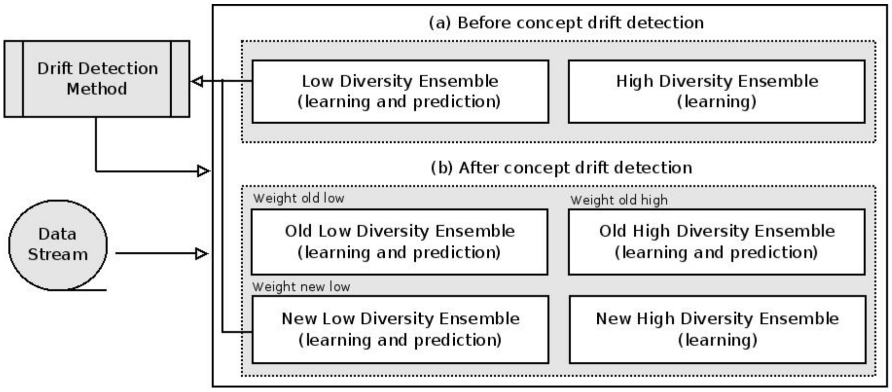
(a) Before concept drift detection
Drift Detection Method
Low Diversity Ensemble (learning and prediction)
High Diversity Ensemble (learning)
(b) After concept drift detection
Data Stream
Weight old low
Weight old high
Old Low Diversity Ensemble (learning and prediction)
Old High Diversity Ensemble (learning and prediction)
Weight new low
New Low Diversity Ensemble (learning and prediction)
New High Diversity Ensemble (learning)

Fig. 1.2 DDD approach for TL in NSE. Arrows represent flow of information.

The low diversity ensemble is monitored by a drift detection method. If a drift is detected, DDD switches to the mode “after concept drift detection”. In this mode, the previous low and high diversity ensembles are kept as old ensembles, and both are activated for learning and predictions. Low diversity is enforced into the learning procedure of the old high diversity ensemble, so that it can strongly learn the new concept while transferring knowledge from the old concept. These ensembles may be beneficial if the concept drift is gradual or not severe. They may also be beneficial in case the drift detection was a false positive drift detection (false alarm), in which case the concept did not change and the old ensembles remain representative of the current concept. A new low diversity ensemble is created to start learning the new concept from scratch. It may be useful if the concept drift is very fast and severe. This ensemble is therefore activated for predictions. It is also monitored by the drift detection method to detect new concept drifts. A new high diversity ensemble is created to weakly learn the new concept, and is not active for predictions. It may become useful if there is a new gradual or not severe concept drift.

The prediction given by the system in the mode “after concept drift detection” is the weighted majority vote of the predictions given by the ensembles that are active for predictions. The weight is the normalised accuracy of the corresponding ensemble since the last concept drift detection, and is calculated in an online way [24] based on incoming training examples. It allows the right ensembles to be emphasised for the given concept drift (or false alarm).

The approach switches back to the mode “before concept drift detection” once the accuracy of the new low diversity ensemble becomes higher than that of the old ensembles, or the accuracy of the old high diversity ensemble becomes higher than that of the low diversity ensembles, with a certain margin. The ensemble which became more accurate than the others and the new high diversity ensemble become the low and high diversity ensembles in the mode “before concept drift detection”.

Any online ensemble learning algorithm and method to encourage high or low diversity could potentially be used. Online bagging ensembles [27] were used in the paper that proposed DDD [24], and different levels of diversity were achieved by using different sampling rates based on the parameter lambda of the Poisson distribution used by online bagging. In particular, the lambda value was kept with the original value of one used by the online bagging algorithm in order to create low diversity ensembles. Lower lambda values (less than one) lead to higher diversity, and were used to produce the high diversity ensembles.

Experiments were performed to evaluate DDD based on several synthetic data streams containing different types of concept drift and on real world data streams from the areas of credit card approval, electricity price prediction and network intrusion detection [24]. The results show that DDD was usually able to maintain or improve predictive performance in comparison with other approaches for learning in NSE (DWM [16] and Baena-Garcia et al. [2]’s approach) and online bagging without mechanisms to deal with concept drifts. Predictive performance was improved specially for gradual and non-severe drifts, which are exactly the cases for which TL via the the old ensembles had been found to be helpful (see first bullet point in the beginning of this subsection).

Overall, DDD and its achieved results further illustrate the benefits of combining TL and learning in NSE. In particular, they show that knowledge from old concepts can be used to help learning the new concept, improving learning in NSE especially when the old and new concepts share some similarities.

1.6 Conclusions

This chapter provided background in the fields of TL and learning in NSE to enable understanding the relationship between these two fields. It gave definitions that existing work in these fields can be considered to be adopting, and examples of several representative approaches. Based on that, a discussion of the similarities and differences between these two fields was provided. This discussion reveals that approaches in these fields have limitations that could be overcome through a better integration between them. For instance, NSE approaches are potentially wasting useful knowledge from past concepts that could be helpful for learning a new concept. Moreover, they cannot benefit from examples coming from different data generating processes. TL, on the other hand, has no notion of continuing time. It is not designed to automatically process incoming data over time and is typically unable to deal with potentially infinite data streams. It cannot automatically cope with concept drifts affecting the present and has no provision to process examples from transitional periods between concepts.

Therefore, this chapter discussed the potential benefits of better integrating the fields of TL and learning in NSE. In particular, two existing approaches for TL in NSE called Dycom and DDD were explained. The positive results obtained by these approaches in the literature highlight the benefit of TL in NSE. Dycom shows how ideas from learning in NSE can be used to make TL aware of continuing time, enabling it to deal with data streams and automatically cope with concept drift. Conversely, DDD shows how ideas from TL can be used to inspire better algorithms for learning in NSE, by enabling knowledge from past concepts to aid the learning of new concepts.

As the first work to provide a detailed discussion of the relationship between TL and NSE, this chapter opens up the path for future research in the emerging area of TL in NSE. It encourages the research communities in these fields to work together, so that improved algorithms can be proposed to tackle challenging real world problems.

Acknowledgement

This work was partially funded by EPSRC Grant No. EP/R006660/1.

References

1. A. Argyriou, T. Evgeniou, and M. Pontil. Multi-task feature learning. In *Proceedings of the 19th Annual Conference on Neural Information Processing Systems*, pages 41–48, 2007.

2. M. Baena-García, J. Del Campo-Ávila, R. Fidalgo, and A. Bifet. Early drift detection method. In *Proceedings of the 4th International Workshop on Knowledge Discovery from DataStreams*, pages 77–86, Berlin, Germany, 2006.
3. C. Bishop. *Pattern Recognition and Machine Learning.* Springer, Singapore, 2006.
4. S. Chen and H. He. SERA: Selectively recursive approach towards nonstationary imbalanced stream data mining. In *Proceedings of the 2009 International Joint Conference on Neural Networks (IJCNN)*, pages 522–529, 2009.
5. W. Dai, G. X. andQ. Yang, and Y. Yu. Transferring naive bayes classifiers for text classification. In *Proceedings of the 22nd National Conference on Artificial Intelligence*, 2007.
6. W. Dai, Q. Yang, G. Xue, and Y. Yu. Boosting for transfer learning. In *Proceedings of the 24th International Conference on Machine Learning*, pages 193–200, 2007.
7. G. Ditzler, M. Roveri, C. Alippi, and R. Polikar. Learning in nonstationary environments: A survey. *IEEE Computational Intelligence Magazine*, 10(4):12–25, 2015.
8. P. Domingos and G. Hulten. Mining high-speed data streams. In *Proceedings of the 6th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD)*, pages 71–80, 2000.
9. R. Elwell and R. Polikar. Incremental learning of concept drift in nonstationary environments. *IEEE Transactions on Neural Networks and Learning Systems (TNNLS)*, 22(10):1517–1531, 2011.
10. J. Gama, P. Medas, G. Castillo, and P. Rodrigues. Learning with drift detection. In *Proceedings of the 7th Brazilian Symposium on Artificial Intelligence (SBIA) - Lecture Notes in Computer Science*, volume 3171, pages 286–295, São Luiz do Maranhão, Brazil, 2004. Springer.
11. J. Gama, I. Zliobaite, A. Bifet, M. Pechenizkiy, and A. Bouchachia. A survey on concept drift adaptation. *ACM Computing Surveys*, 46(4):44:1–44:44, 2015.
12. J. Huang, A. Smola, A. Gretton, K. Borgwardt, and B. Scholkopf. Correcting sample selection bias by unlabeled data. In *Proceedings of the 19th Annual Conference on Neural Information Processing Systems*, 2007.
13. ISBSG. The International Software Benchmarking Standards Group. `http://www.isbsg.org`, 2011.
14. B. Kitchenham, E. Mendes, and G. Travassos. Cross versus within-company cost estimation studies: A systematic review. *IEEE Transactions on Software Engineering (TSE)*, 33(5):316–329, 2007.
15. J. Z. Kolter and M. A. Maloof. Using additive expert ensembles to cope with concept drift. In *Proceedings of the 22nd International Conference on Machine Learning (ICML)*, pages 449–456, Bonn, Germany, 2005.
16. J. Z. Kolter and M. A. Maloof. Dynamic weighted majority: An ensemble method for drifting concepts. *Journal of Machine Learning Research (JMLR)*, 8:2755–2790, 2007.
17. B. Krawczyk, L. Minku, J. Gama, J. Stefanowski, and M. Wozniak. Ensemble learning for data stream analysis: a survey. *Information Fusion*, 37:132–156, 2017.
18. L. Kuncheva and I. Zliobaite. On the window size for classification in changing environments. *Intelligent Data Analysis (IDA)*, 13(6):861–872, 2009.
19. T. Menzies, B. Caglayan, Z. He, E. Kocaguneli, J. Krall, F. Peters, and B. Turhan. The promise repository of empirical software engineering data. http://promisedata.googlecode.com, 2012.
20. L. Minku and S. Hou. Clustering dycom: An online cross-company software effort estimation study. In *Proceedings of the 13th International Conference on Predictive Models and Data Analytics for Software Engineering (PROMISE)*, page 10p., 2017 (accepted).
21. L. Minku and X. Yao. Can cross-company data improve performance in software effort estimation? In *Proceedings of the 8th International Conference on Predictive Models in Software Engineering (PROMISE)*, pages 69–78, Lund, Sweden, 2012.

22. L. Minku and X. Yao. How to make best use of cross-company data in software effort estimation? In *Proceedings of the 36th International Conference on Software Engineering (ICSE)*, pages 446–456, 2014.
23. L. L. Minku, A. White, and X. Yao. The impact of diversity on on-line ensemble learning in the presence of concept drift. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 22(5):730–742, 2010.
24. L. L. Minku and X. Yao. DDD: A new ensemble approach for dealing with concept drift. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 24(4):619–633, 2012.
25. K. Nishida. *Learning and Detecting Concept Drift.* PhD thesis, Hokkaido University, Japan, 2008.
26. M. Oquab, L. Bottou, I. Laptev, and J. Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1717–1724, 2014.
27. N. C. Oza and S. Russell. Online bagging and boosting. In *Proceedings of the 2005 IEEE International Conference on Systems, Man and Cybernetics*, volume 3, pages 2340– 2345, New Jersey, 2005. Institute for Electrical and Electronics Engineers.
28. S. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 22(10):1345–1359, 2010.
29. S. J. Pan, I. W. Tsang, J. T. Kwok, and Q. Yang. Domain adaptation via transfer component analysis. *IEEE Transactions on Neural Networks (TNN)*, 22(2):199–210, 2011.
30. R. Raina, A. Battle, H. Lee, B. Packer, and A. Ng. Self-taught learning: Transfer learning from unlabeled data. In *Proceedings of the 24th International Conference on Machine Learning (ICML)*, pages 759–766, 2007.
31. M. Rosenstein, Z. Marx, and L. Kaelbling. To transfer or not to transfer. In *Proceedings of the Conference on Neural Information Processing Systems (NIPS) Workshop Inductive Transfer: 10 Years Later*, 2005.
32. G. Ross, N. Adams, D. Tasoulis, and D. Hand. Exponentially weighted moving average charts for detecting concept drift. *Pattern Recognition Letters*, 33:191–198, 2012.
33. R. E. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):297–336, 1999.
34. M. Scholz and R. Klinkenberg. Boosting classifiers for drifting concepts. *Intelligent Data Analysis (IDA)*, 11(1):3–28, 2007.
35. W. Street and Y. Kim. A streaming ensemble algorithm (SEA) for large-scale classification. In *Proceedings of the 7th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD)*, pages 377–382, 2001.
36. Y. Sun, K. Tang, L. L. Minku, S. Wang, and X. Yao. Online ensemble learning of data streams with gradually evolved classes. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 28(6):1532–1545, 2016.
37. Y. Sun, K. Tang, Z. Zhu, and X. Yao. Concept drift adaptation by exploiting historical knowledge. *ArXiv* *https://arxiv.org/abs/1702.03500*, 2017.
38. A. Tsymbal. The problem of concept drift: Definitions and related work. Technical Report 106, Computer Science Department, Trinity College, Dublin, Ireland, 2004.
39. B. Turhan, T. Menzies, A. Bener, and J. Di Stefano. On the relative value of cross-company and within-company data for defect prediction. *Empirical Software Engineering*, 14(5):540–578, 2009.
40. H. Wang, W. Fan, P. S. Yu, and J. Han. Mining concept-drifting data streams using ensemble classifiers. In *Proceedings of the 9th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD)*, pages 26–235, 2003.
41. S. Wang, L. Minku, and X. Yao. A systematic study of online class imbalance learning with concept drift. *ArXiv* *https://arxiv.org/abs/1703.06683*, 2017 (under review).
42. G. Widmer and M. Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23(1):69–101, 1996.