Cost-Effective Transfer Learning for Data Streams

I. Theorem Proof

Theorem 1. *Given a data stream where data instances arrive in balanced classes. Let d_c, d_e and d_f be the construction complexities of the correct, error and full region. If $d_f < (d_c \vee d_e)$, $(d_c < d_e) \wedge (d_f < d_e)$ or $(d_c < d_e) \wedge (d_e < d_f < d_c + d_e)$, the transferred model has low adaptability.*

Proof. Suppose we have two arbitrary full tree classifiers T_c and T_e with maximum tree depth d_c and d_e respectively. Since these two trees are learnt from $R_f = \{R_c, R_e\}$, they can construct T_f with a maximum depth d_f, through three different types of construction including both T_c and T_e: T_c is a subtree of T_e, T_e is a subtree of T_c, T_c and T_e are independent of each other. We prove the theorem by exhausting all potential comparisons among d_c, d_e and d_f.

There are two cases. In the first case where $d_c > d_e$, we consider the possible relationships between d_e and d_f, and d_c and d_f. On one hand if $d_e > d_f$ and $d_f < d_c$ then T_f cannot be constructed by T_c and T_e without pruning their internal nodes. However since $d_e > d_f$ we have $\phi(T_e) > \phi(T_f)$. Therefore this case has low adaptability. On the other hand $d_e > d_f$ and $d_f > d_c$ is invalid. If $d_e < d_f$ and $d_f < d_c$ then T_e has to be either a subtree of or independent from T_c to have $d_e < d_f < d_c$. Therefore this case has high adaptability. Where $d_e < d_f$ and $d_f > d_c$, $d_f = [d_c, d_c + d_e]$ thus we have $\phi(T_f) > \phi(T_e)$. Therefore this case has high adaptability.

In the second case where $d_c < d_e$, we again consider the possible relationships between d_e and d_f, and d_c and d_f. Where $d_e > d_f$ and $d_f < d_c$ the result is the same as if $d_e > d_f$ and $d_f < d_c$ above, namely there is low adaptability. If $d_e > d_f$ and $d_f > d_c$ then T_c has to be either a subtree of or independent from T_c to have $d_c < d_f < d_e$. Since $\phi(T_e) > \phi(T_f)$ we have low adaptability. Where $d_e < d_f$ and $d_f < d_c$ is invalid. If $d_e < d_f$ and $d_f > d_c$ then $d_f = [d_e, d_c + d_e]$. Therefore T_e is a subtree of T_c or vice versa. Further, we need to consider two extra possibilities within this case, that is $d_f < d_c + d_e$ and $d_f > d_c + d_e$. $d_f < d_c + d_e$ implies that there is overlapping of features in T_c and T_e. This imposes higher learning difficulty on the error region classifier. Therefore this case has low adaptability if $d_f < d_c + d_e$. □

II. Reproducibility and Open Source

The experiments were performed on the NVIDIA DGX-2 Station (Version 4.0.7, 40 cores, Intel(R) Xeon(R) CPU E5-2698 v4 @ 2.20GHz, 252 GiB memory). We developed our transfer learning synthetic data generator based on Random Tree Generator from scikit-multiflow[1]. It is available at *data/random_tree_generator.py*. All the benchmarks and our OPERA framework were developed under MOA[2], which is an open-source stream mining framework. The Random Forest base model is MOA's built-in Adaptive Random Forest with its adaptation functions, (*i.e.*, background tree and weighted vote) turned off. We used grid search to tune the parameters. The parameters range are in the format of start:step:end, inclusive: $K = 1 : 2 : 23$, $\lambda_{conv} = 0.1 : 0.01 : 0.2$.

A. Pseudocode

We provide additional pseudocode in Algorithm 1, which includes all the control flows of the OPERA framework as an aid to time complexity evaluation.

B. Reproducibility

We developed our transfer learning synthetic data generator based on Random Tree Generator from scikit-multiflow[3]. All the benchmarks and our OPERA framework were developed under MOA[4], which is an open source framework for Big Data stream mining. The Random Forest base model is MOA's built-in Adaptive Random Forest with its adaptation functions (*, i.e.* background tree and weighted vote) turned off. The source codes for ECPF and AOTrAdaBoost are provided by their authors. Both were also developed under MOA. Table I lists the locations of the main source code files for all benchmarks in the supplementary materials.

The essential parameters for data generation and benchmarking are included in the main paper. Other specific parameters used for each dataset and benchmark pairs are available in supplemental materials under path *run/*.

C. Datasets

We assess the effectiveness of OPERA comprehensively on three different types of transfer learning datasets: synthetically generated, semi real-world, and real-world. The synthetic and semi real-world datasets are generated with varying degrees of similarities to validate OPERA's estimation of model adaptability in the target stream. We further generated these types of datasets with ten distinct seeds to assess the stability of our technique. The real-world datasets assess the practicability of our technique in real-world scenarios. The degrees of similarities are unknown in this type of dataset. Table II lists the characteristics of all the datasets, including the number of features, classes, instances from the source and the target stream. The *Config* column provides reference pointers to the specific configuration for the datasets.

[1]https://github.com/scikit-multiflow/scikit-multiflow

[2]https://github.com/Waikato/moa

[3]https://github.com/scikit-multiflow/scikit-multiflow

[4]https://github.com/Waikato/moa

Algorithm 1: The OPERA Framework

Data: An instance (x, y) from the target data stream T arriving at time t
Input : Transferred Model M_{SRC}

function `train((x, y))`:
  if *IsInObservationPeriod* **then**
    $R_f \leftarrow R_f \cup \{(x, y)\}$;
    if $M_{SRC}.predict(x) = y$ **then**
      $R_c \leftarrow R_c \cup \{(x, y)\}$;
    else
      $R_e \leftarrow R_f \cup \{(x, y)\}$;
    end
    if *DetectSufficientData* **then**
      `/* Init Incremental Patching */`
      Train an *ErrorRegionClassifier* E with R_f and R_c;
      Obtain d_f, d_c, d_e by constructing three Phantom Trees with R_f, R_c, R_f;
      Obtain adaptability based on d_f, d_c, d_e ;
      if *adaptability = high* **then**
        IsInObservationPeriod $\leftarrow$ False;
        init patch classifier P;
      else
        init a new classifier M_{TRT};
      end
    end
  else
    if *adaptability == high* **then**
      `/* Incremental Patching */`
      if $M_{SRC}.predict(x) \neq y$) **then**
        P.train(x, y);
      end
      M_{SRC}.train(x, y);
    else
      M_{TGT}.train(x, y);
    end
  end
end
function `predict((x))`:
  if $M_{TGT} == null$ **then**
    if *E.predict(x) = in error region* **then**
      P.predict(x) ;
    else
      M_{SRC}.predict(x) ;
    end
  else
    M_{TGT}.predict(x) ;
  end
end

TABLE I: Source Code Locations in Supplementary Materials

Benchmark	Location
RF	classifiers/meta/AdaptiveRandomForest.java
AOTrAdaBoost	classifiers/InstanceTransfer/TransForest.java
ECPF	classifiers/meta/ECPF.java
OPERA	classifiers/transfer/TransferFramework.java

Synthetic Datasets. For different transfer learning scenarios, we built a synthetic data generator based on the Random Tree generator [1] to control the degrees of similarities across different domains. The generator starts by building a full decision tree with depth d_0, by splitting on features and assigning classes on leaves at random. We further construct the target stream by pruning b number of branches at level d_1 from the tree, rebuilding b new full subtrees on the pruned branches with maximum depth at d_2. Data is generated by random walks from the root node to leaves in the two different trees, where the internal nodes and leaf nodes construct features and classes respectively. We generated three datasets denoted as High, Mid and Low, indicating the similarities between the source and the target data streams. We set $d_0 = 8$, $d_1 = 3$ for all the datasets. In addition, the High dataset is generated with a configuration of $d_2 = 6$, $b = 1$; the Mid dataset is of $d_2 = 12$, $b = 1$; and the Low is of $d_2 = 6, b = 3$.

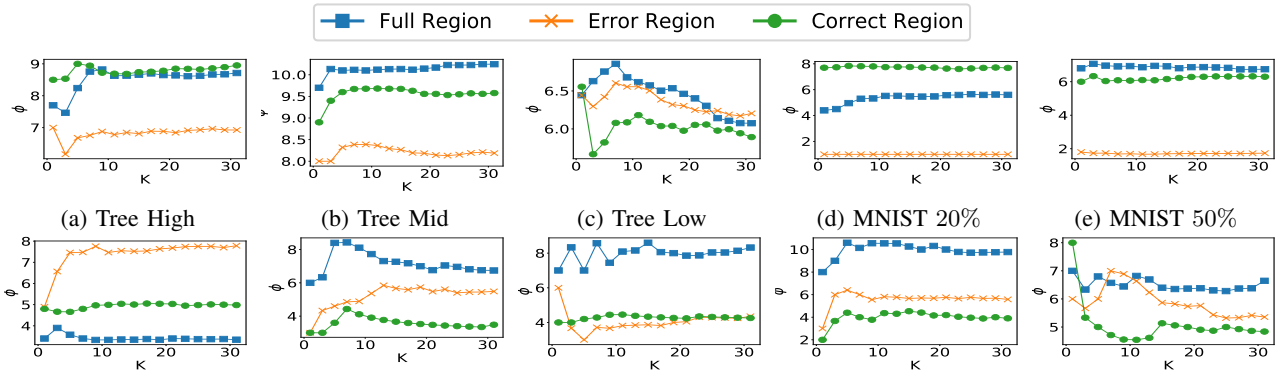
Semi Real-World Datasets. Inspired by the construction of transfer datasets [2], we use the original MNIST[5] and Fashion MNIST[6] dataset as the source stream, and generate the target stream with varying similarities by inverting all pixel values of 20%, 50%, 80% of the classes. We denote these configurations as High, Mid and Low for both MNIST and Fashion MNIST, indicating the similarity between the domains. In addition, we apply a 2×2 filter on both MNIST and Fashion MNIST datasets for reasonable performance outputs from the random forest base learners with limited tree models.

Real-World Datasets. We used two real world datasets, Covtype and Bike. Covtype[7] is a dataset commonly used to assess the classification performance of data stream classifiers. It contains recurrent concept drifts, meaning the dataset contains a set of different data distributions, and some of the data distributions recur at different points in time. Bike datasets [3] describe rental bikes which are in low or high demand at different times in different cities. We use the configuration of (i) weekdays from Washington D.C. as the source and weekends in London as the target, denoted as C0 and (ii) weekends from Washington D.C. as the source and weekdays in London as the target, denoted as C1. Due to the similarities between transfer learning and learning in non-stationary environments (*i.e.*, data stream environments containing concept drifts), we can break the dataset into chunks to construct transfer learning scenarios [4]. The whole dataset is divided into five chunks, based on the performance of a single Hoeffding tree, as a decline in accuracy means a concept drift has occurred. We first apply B-spline to smooth out the output performance of the Hoeffding tree, followed by finding the local minima to break the dataset into chunks. We then use every neighbouring block as the source and target streams. A total of four transfer learning scenarios are constructed with the five blocks.

III. Parameter Sensitivity Analysis

In this section, we investigate the parameter sensitivity of constructing phantom trees of the full, correct, and error regions with respect to the user-defined parameter K (in the Phantom Tree Algorithm), the number of phantom branches to construct. We examine one set of datasets from each synthetic, semi real-world and real-world dataset type for completeness. Fig. 1 shows with varying K, relationships of the construction complexity ϕ among the full, correct and error regions are relatively stable. A notable case is shown in Fig. 1(c), where the relationship between d_f and d_e inverts with a relatively higher K. However, the adaptability according to Theorem

[5]http://yann.lecun.com/exdb/mnist/
[6]https://github.com/zalandoresearch/fashion-mnist
[7]https://archive.ics.uci.edu/ml/datasets/covertype



(a) Tree High (b) Tree Mid (c) Tree Low (d) MNIST 20% (e) MNIST 50%

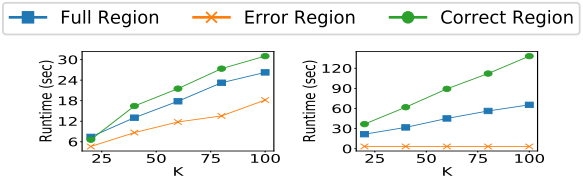
(f) MNIST 80% (g) Covtype 0 (h) Covtype 1 (i) Covtype 2 (j) Covtype 3

Fig. 1: Stability of model construction complexity ϕ with regards to phantom tree's parameter K.

TABLE II: Dataset Characteristics

Dataset	Config.	#Features	#Classes	#Samples Src.	#Samples Trt.
Tree	High	20	3	50000	50000
	Mid			50000	50000
	Low			50000	50000
MNIST	High	196	10	70000	70000
	Mid			70000	70000
	Low			70000	70000
Fashion MNIST	High	196	10	70000	70000
	Mid			70000	70000
	Low			70000	70000
Covtype	0	54	7	73501	48334
	1			48334	120836
	2			120836	72502
	3			72502	72501
Bike	0	4	2	5014	12060
	1			11865	4970

1 remains the same. Fig 2 shows that with varying K, the runtime of phantom tree constructions increases linearly. The number of features, features values and classes is fixed during the lifetime of a data stream, and the size of the observation period stays the same for each dataset given a user-defined γ_{conv}, the runtime complexity for constructing K phantom branches is $O(K)$. The MNIST datasets has a longer runtime compared to the Tree dataset, as it has a higher number of feature dimensions. We observed a similar runtime growth pattern across all the datasets.



(a) Tree High (b) MNIST 20%

Fig. 2: Runtime for constructing phantom trees for the three regions when varying the K parameter.

IV. Adaptability Estimation Evaluation

Fig. 3 captures the adaptation speed at the beginning of the target stream for each dataset. The observation is consistent with OPERA's adaptability estimation, where estimated higher adaptability yields faster adaptation.

V. Additional Experiment Results

Fig. 4 provides additional runtime results. Similarly to the results the runtime increases linearly as K increases. The same pattern is observed across all datasets.

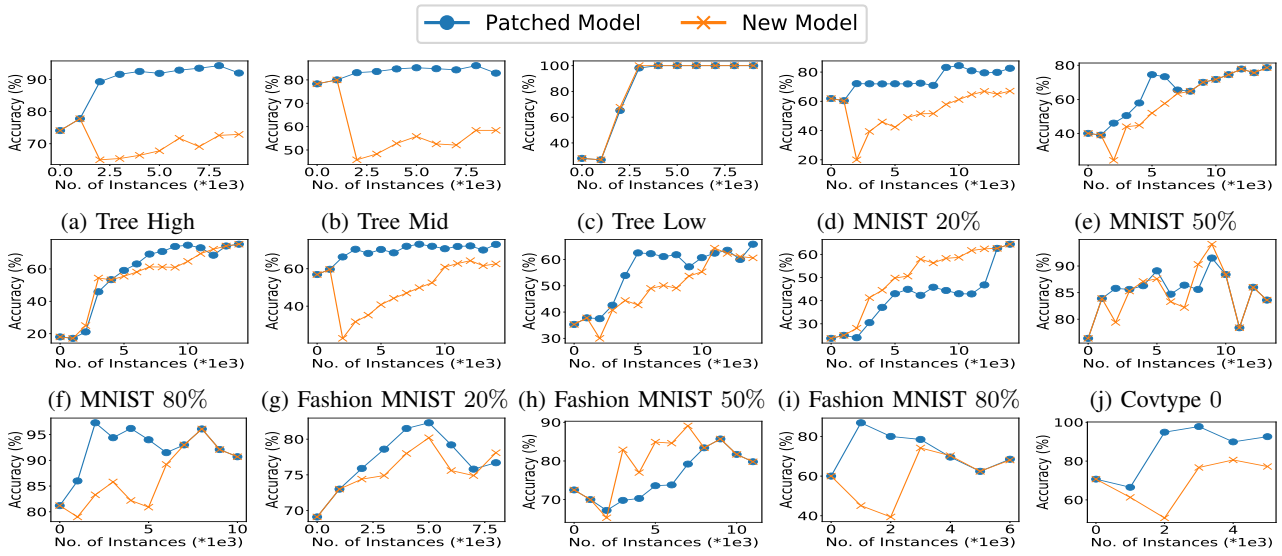
References

[1] P. Domingos and G. Hulten, "Mining high-speed data streams," in *SIGKDD*. Association for Computing Machinery, 2000, p. 71–80.

[2] S. Kauschke and J. Fürnkranz, "Batchwise patching of classifiers," *AAAI*, vol. 32, no. 1, Apr. 2018.

[3] H. Du, L. L. Minku, and H. Zhou, "Marline: Multi-source mapping transfer learning for non-stationary environments," in *ICDM*, 2020, pp. 122–131.

[4] L. L. Minku, *Transfer Learning in Non-stationary Environments*. Cham: Springer International Publishing, 2019, pp. 13–37.

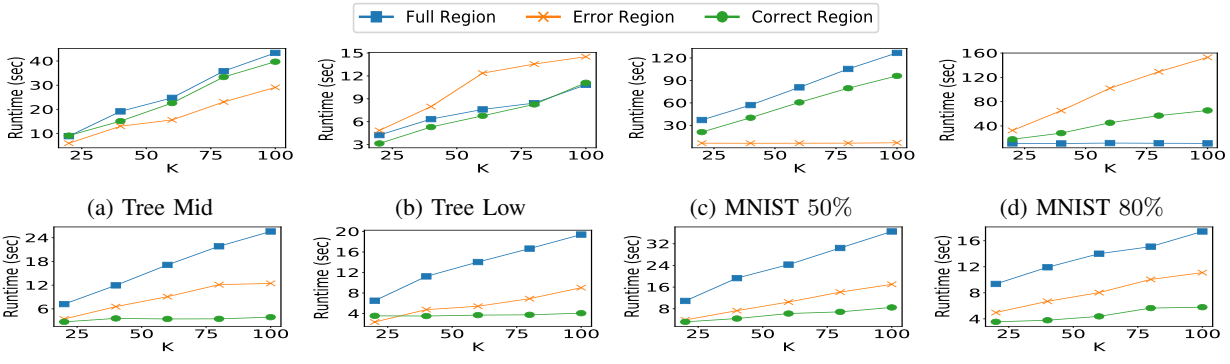


(a) Tree High (b) Tree Mid (c) Tree Low (d) MNIST 20% (e) MNIST 50%

(f) MNIST 80% (g) Fashion MNIST 20% (h) Fashion MNIST 50% (i) Fashion MNIST 80% (j) Covtype 0

(k) Covtype 1 (l) Covtype 2 (m) Covtype 3 (n) Bike C0 (o) Bike C1

Fig. 3: Comparison of accuracy performance at the beginning of the target stream between patching the transferred model and constructing a new model from scratch.



(a) Tree Mid (b) Tree Low (c) MNIST 50% (d) MNIST 80%

(e) Covtype 0 (f) Covtype 1 (g) Covtype 2 (h) Covtype 3

Fig. 4: Additional runtime for constructing phantom trees for the three regions when varying the K parameter.