10.24 Paper Presentation:

Deep Forest and Kernel

Yibin Xiong

October 24, 2021

Table of Contents

Motivation

- Why **deep** models are successful?
  - layer-by-layer processing
  - in-model feature transformation
  - sufficient model complexity

  Deep neural net $\Rightarrow$ Deep (random) forest
- Ensemble: combine *accurate* (bias) and *diverse* (variance) base learners

Motivation

▷ Why **deep** models are successful?

- layer-by-layer processing
- in-model feature transformation
- sufficient model complexity

Deep neural net $\Rightarrow$ Deep (random) forest

▷ Ensemble: combine *accurate* (bias) and *diverse* (variance) base learners

$\sim$ Aside: Types of random forests

1. completely random forest (CRF)

Features used for splits and cutoff values are randomly chosen. Grow the tree until each leaf is "pure" (i.e. contain only instances from 1 class).

2. random forest

Randomly pick $\sqrt{d}$ features as candidates for splits. Each time select the one that minimizes Gini index for split.

Multi-grained Scanning [5]

- Create more converted instances by sliding a window on raw inputs; enrich the input
- introduce the inductive bias of "locality", like CNN
- converted instances have the same label as their origins

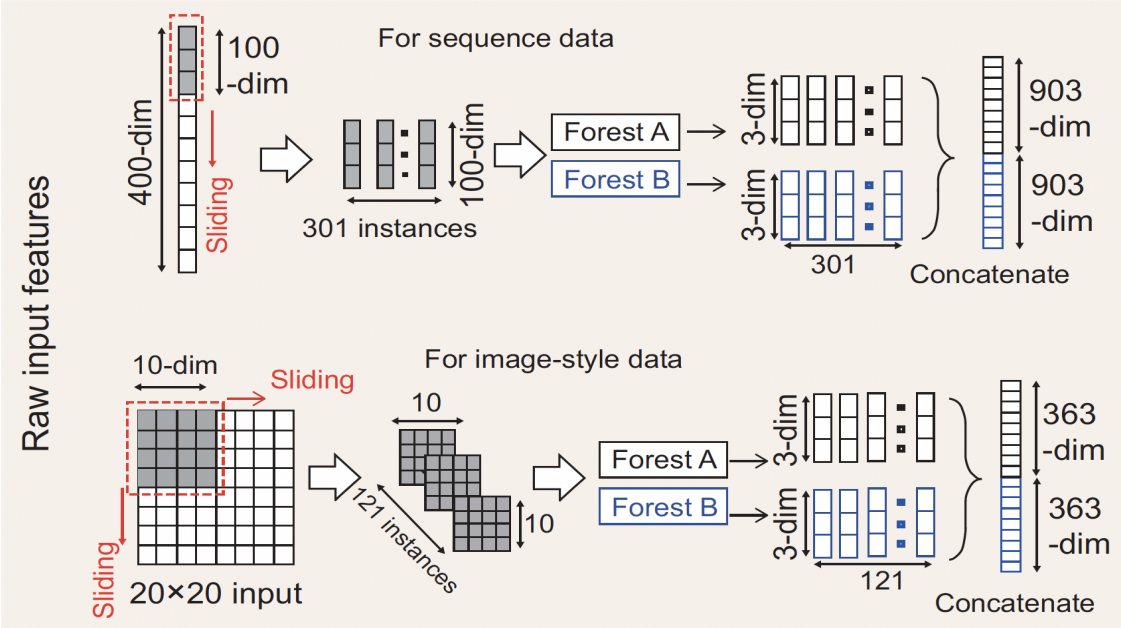


Figure 4. Illustration of feature re-representation using sliding window scanning. Suppose that there are three classes, the raw features are 400-dim, and the sliding window is 100-dim.

Question: Does the "window" have weights? I guess no.

Cascade Forest Structure [5]

- Utilizes *layer-by-layer* processing
- Adopts the idea of skip connection in ResNet

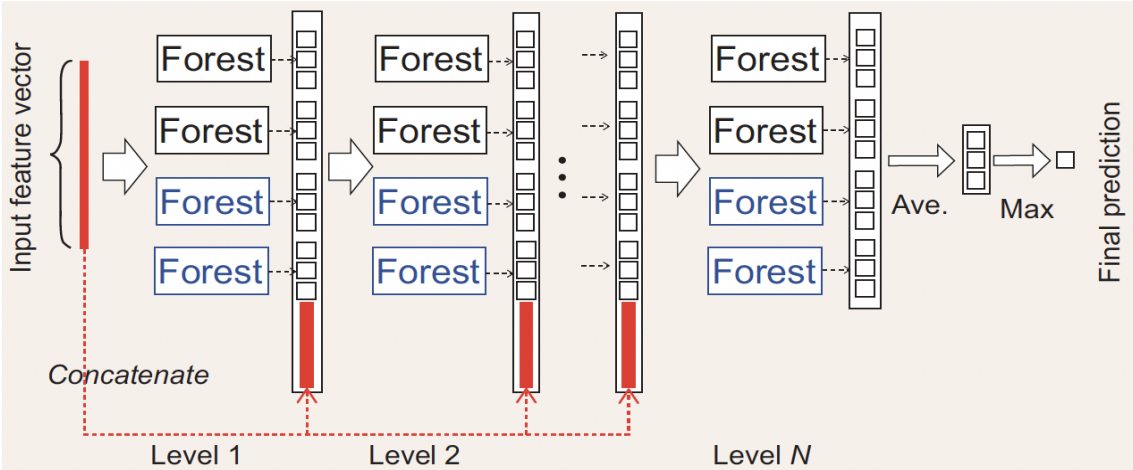


Figure 2. Illustration of the cascade forest structure. Suppose that each level of the cascade consists of two random forests (black) and two completely random forests (blue). Suppose that there are three classes to predict; thus, each forest will output a 3D class vector, which is then concatenated for re-representation of the input.

- How to determine depth?

– "If growing a new level does not improve the performance, the growth of the cascade terminates and the estimated number of levels is obtained."

gcForest

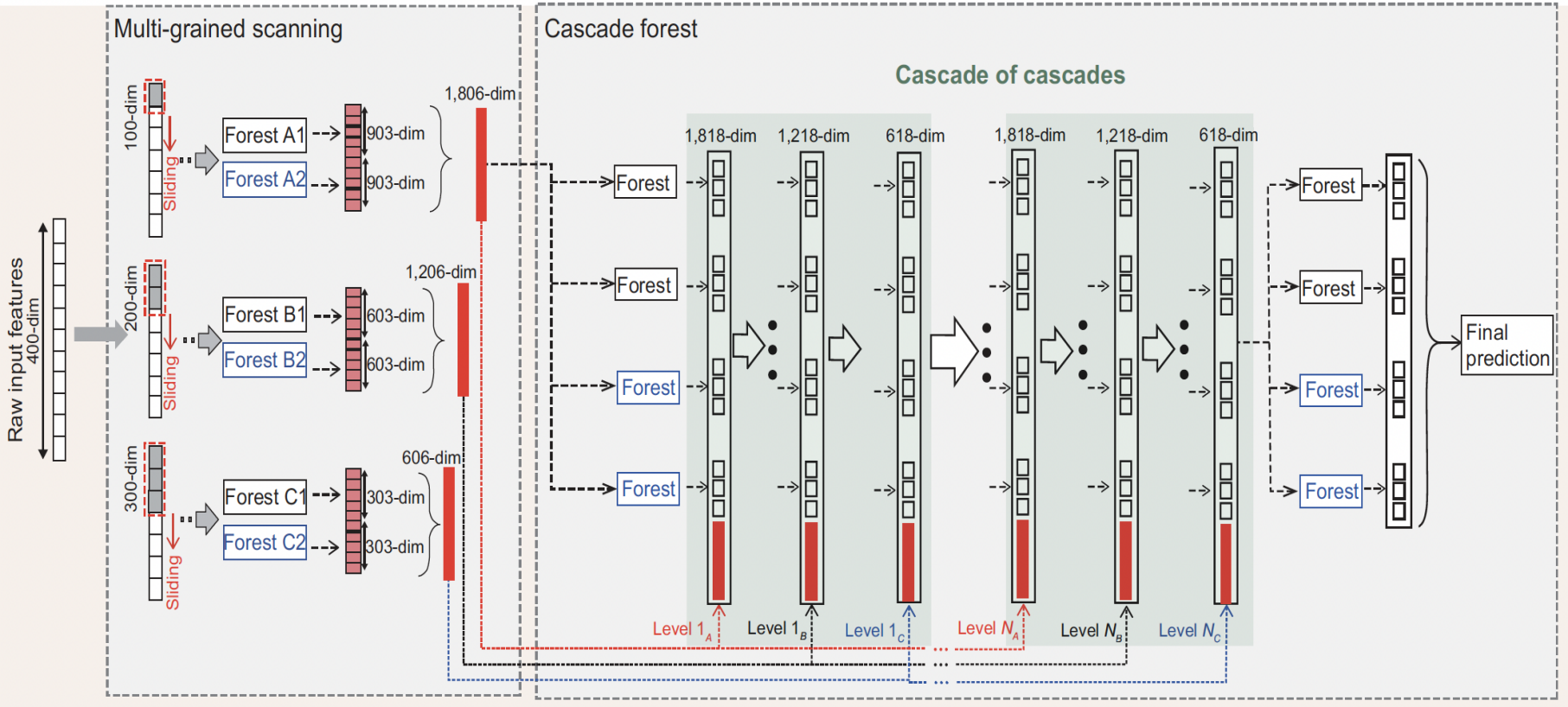


Figure 5. Overall procedure of gcForest. Suppose that there are three classes, the raw features are 400-dim, and three sizes of sliding windows are used.

DNN V.S. Deep Forest

DNN

- has too many hyperparameters to tune
- is good ONLY when we have sufficient data, bad for small-scale training data
- network complexity is predetermined and not adaptable according to sample size $\Rightarrow$ overfitting
- lack interpretability
- not as good as tree-based model for dealing with tabular data
- has difficulty handling non-differentiable real-world problems

Deep Forest (gcForest)

- has *supervised* feature extraction (in multi-grained scanning)
- output of intermediate layers can be used as final output (*Are probabilities good "features" to learn*)
- is NOT finetunable, so it cannot be used in unsupervised or semi-supervised settings
- Its performance is more robust to hyperparameter settings

Table of Contents

Problems & Improvements [1]

Problems: high time cost and memory requirements

- Number of *instances*: Many newly generated instance in multi-grained scanning
- Number of *features*: The original input (with high dimension) is passed into every cascade

Problems & Improvements [1]

Problems: high time cost and memory requirements

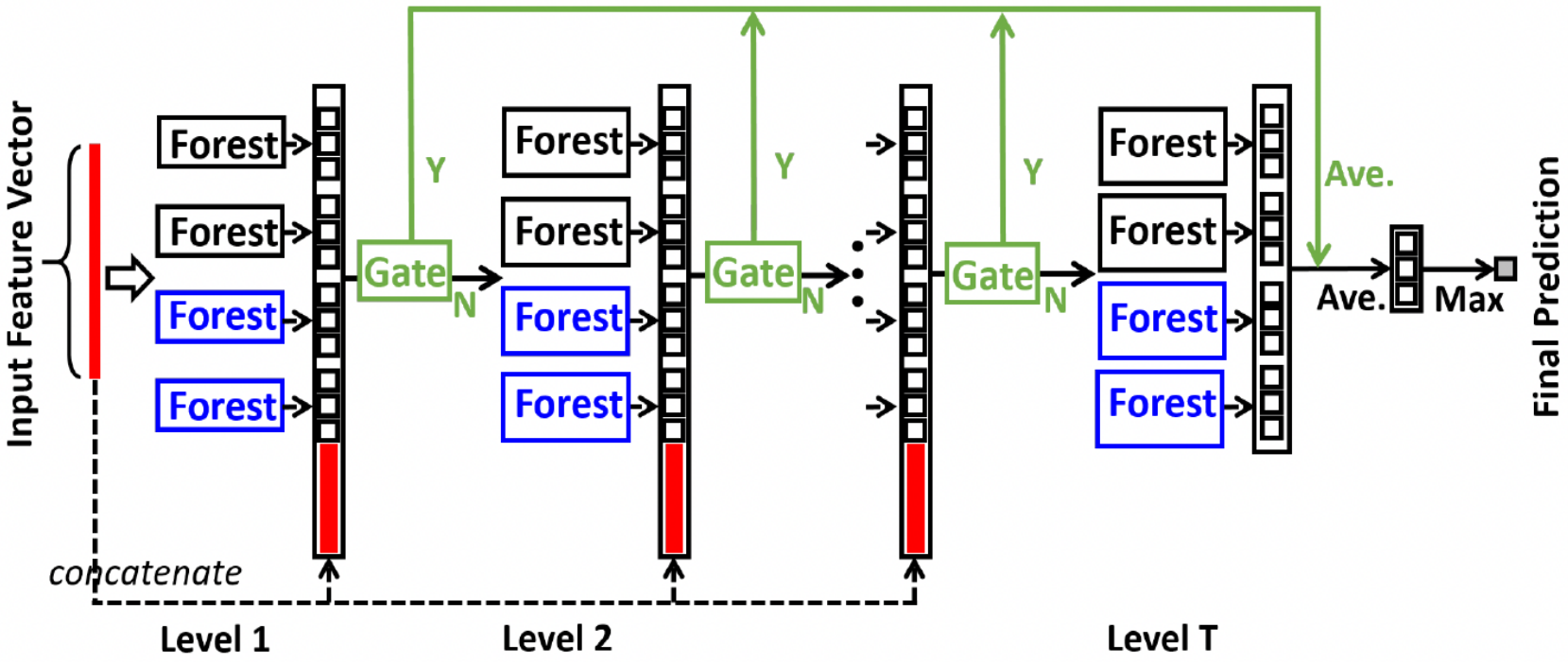
- Number of *instances*: Many newly generated instance in multi-grained scanning
- Number of *features*: The original input (with high dimension) is passed into every cascade

Improvements:

- Confidence screening
  If an instance is easy to predict (namely, the model has high confidence in prediction), then its prediction is produced at the *current* level. Otherwise, it needs to go through more levels.
- Feature screening
  If a feature is useful for improving the performance (in further levels), then it is passed to the next level. Other features are reserved for the current level.
- Unsupervised feature transformation using completely random forest
  - Replaces *supervised* multi-grained scanning
  - linear runtime

Instance-based Improvement

1. Confidence screening



(a) Cascade forest with *confidence screening*

As the level increases, the forests being used are more complex for "hard" instances.

Instance-based Improvement

How to find the threshold η_t at level t?

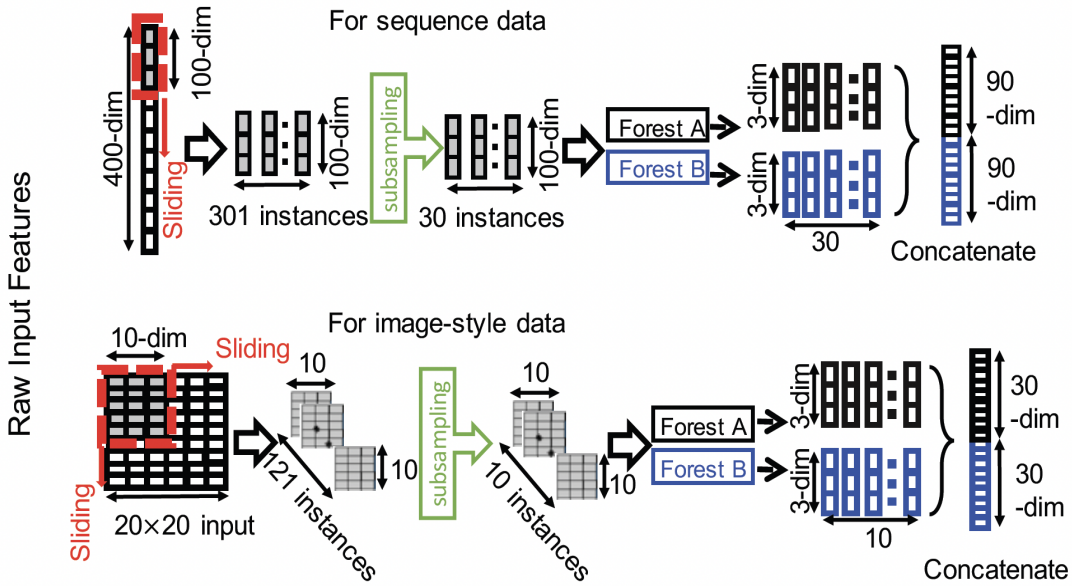
– η_t is automatically determined based on *cross-validation error rate* ϵ_t.

- Arrange the instances x_i by prediction confidence c_i in descending order
- Let $L(x_1, \ldots, x_k) = \frac{1}{k} \sum_{i=1}^{k} \mathbb{1}\{g_t(x_i) \neq y_i\}$ be the average error rate of k instances with the largest confidence
- $\eta_t := \min\{c_k \mid L(x_1, \ldots, x_k) < a\epsilon_t, \text{for some } k \in \{1, \ldots, m\}\}$
- $a \in (0, 1)$ is a hyperparameter chosen in advance
- *Question: do we choose the smallest possible k that satisfies the inequality?*

Instance-based Improvement

2. Subsampling in multi-grained scanning

- Randomly sample a proportion of instances from the newly generated instances



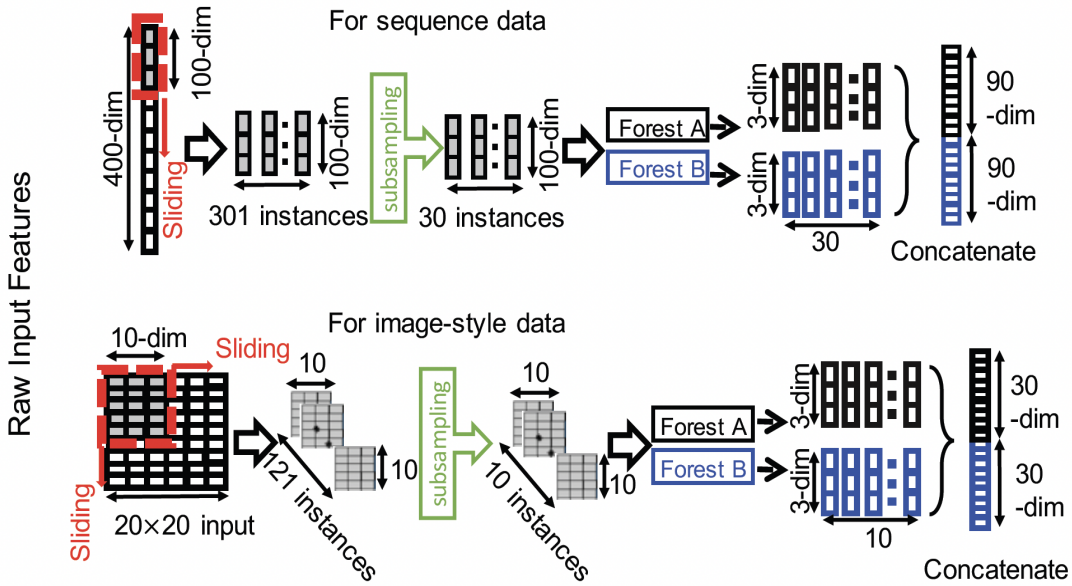
(b) Feature re-representation using sliding window scanning with *subsampling*

- Also reduces the number of *features* in future levels

Instance-based Improvement

2. Subsampling in multi-grained scanning

◦ Randomly sample a proportion of instances from the newly generated instances



(b) Feature re-representation using sliding window scanning with *subsampling*

◦ Also reduces the number of *features* in future levels

Question: For example in image classification, what if some important parts of an image are NOT sampled? Which ones are safe to discard?

Feature-based Improvement

1. **Feature screening**: Use a selected set of *important* features rather than all features (like "hard-version" attention mechanism)
   - Reduce dimension for better computational efficiency
   - Screen out "irrelevant" features
   - Even if all original features are relevant, tree models tend to (frequently) use only a subset of them (*augmented features*). Feature screening helps balance between augmented features and the others.

Feature-based Improvement

1. **Feature screening**: Use a selected set of *important* features rather than all features (like "hard-version" attention mechanism)
   - Reduce dimension for better computational efficiency
   - Screen out "irrelevant" features
   - Even if all original features are relevant, tree models tend to (frequently) use only a subset of them (*augmented features*). Feature screening helps balance between augmented features and the others.

▷ How do we define the "importance" of the features? (FS-rank)
For a single decision tree DT_n, the importance of feature X_j is defined as

$$\mathcal{I}_n^j = \sum_{\ell=1}^{L} \hat{\delta}_\ell \mathbb{1}\{X_{v_\ell} = X_j\}$$

where L is the number of *internal* nodes, X_{v_ℓ} is the feature used to split at node ℓ, and $\hat{\delta}_\ell$ is the estimated improvement of impurity after splitting.

Feature-based Improvement

For a forest, the importance of X_j is $\mathcal{I}^j = \sum_{n=1}^{N} \frac{1}{N} \mathcal{I}_n^j$. With the importance measures of all features, we select those with importance greater than a threshold τ.

Feature-based Improvement

For a forest, the importance of X_j is $\mathcal{I}^j = \sum_{n=1}^{N} \frac{1}{N} \mathcal{I}_n^j$. With the importance measures of all features, we select those with importance greater than a threshold τ.

$\triangleright$ Problems:

i) Cross validation for choosing τ requires high computational cost since we need to refit the model using a new set of features.

ii) The greedy selection strategy does not consider the *redundancy* of features.
"Given a selection size budget, selecting *redundant* features means informative features with smaller importances are discarded which may degrade the prediction performance."

Feature-based Improvement

▷ FS-reg: an improved version

$$\hat{\beta} = \arg\min_{\beta} \left\| y - \sum_{j=1}^{d} \mathbf{x}_j \beta_j \right\|_2^2 + \lambda \sum_{j=1}^{d} (\mathcal{I}^{\max} - \mathcal{I}^j)|\beta_j|, \quad (7)$$

where $\mathcal{I}^{\max}$ is the maximum value of $\{\mathcal{I}^1, \mathcal{I}^2, \ldots, \mathcal{I}^d\}$ and $\lambda \geq 0$. The objective function aims to minimize the squared loss with a weighted l_1 regularization.

- This is more like attention, i.e. soft-alignment
- Solve this optimization problem using *least angle regression*

Feature-based Improvement

▷ FS-reg: an improved version

$$\hat{\beta} = \arg\min_{\beta} \left\| y - \sum_{j=1}^{d} \mathbf{x}_j \beta_j \right\|_2^2 + \lambda \sum_{j=1}^{d} (\mathcal{I}^{\max} - \mathcal{I}^j) |\beta_j|, \quad (7)$$

where $\mathcal{I}^{\max}$ is the maximum value of $\{\mathcal{I}^1, \mathcal{I}^2, \ldots, \mathcal{I}^d\}$ and $\lambda \geq 0$. The objective function aims to minimize the squared loss with a weighted l_1 regularization.

◦ This is more like attention, i.e. soft-alignment

◦ Solve this optimization problem using *least angle regression*

◦ "At the first level, feature importance is calculated by classification trees where the learning target is the labels; at each of the following levels, feature importance is calculated by regression trees based on the remaining samples of this level where the learning target is the residual of the last level."

Feature-based Improvement

2. **Completely-Random Forest Transformation**

- Given the newly generated instances $\{z_i^1, \ldots, z_i^r\}$ associated with x_i in multi-grained scanning
- Build a completely-random forest and get classification results

* Since splits are randomly chosen, we don't need labels to build the CRF

- Define the transformation function $q(z_j, \ell^k) = \|z^j - c_k\|_2$, where c_k is the average of all instances in node k.
- The transformed feature $D_{j,k} = q(z_j, \ell^k)$. We arrange these numbers into a matrix $D \in \mathbb{R}^{r \times u}$
- Further improvement for runtime efficiency:
  r is large, so we group the converted instances $\{z_i^1, \ldots, z_i^r\}$ into $\{\mathcal{G}_1, \ldots, \mathcal{G}_{r_g}\}$, where each group contains r/r_g consecutive instances. Then $D'_{j,k} = \sum_{i \in \mathcal{G}_j} \|z_i - c_k\|_2$. $D' \in \mathbb{R}^{r_g \times u}$
- Flatten D or D' out into a transformed feature vector

Table of Contents

Multi-Label Learning

Multi-label Learning Problem

- Given $\mathcal{D} = \{(x_1, y_1), \ldots (x_m, y_m)\}$, where $x_i \in \mathbb{R}^d$, $y_i \in \{0, 1\}^\ell$, find a classifier $H : \mathcal{X} \to \{0, 1\}^\ell$ that optimizes a chosen performance metric.
- Traditionally, treat this problem as ℓ independent *binary* classification problem.
  e.g. SVM, logistic regression, LDA, decision trees
- However, some classes are **related**. Information on one class may be helpful for predicting whether an instance is in some other classes.
- Capturing such **correlations** is the key in multi-label learning.
  e.g. deep neural network, deep forest

Performance Measures for Multi-label Learning

Measure	Formulation
hamming loss $\downarrow$	$\frac{1}{ml}\sum_{i=1}^{m}\sum_{j=1}^{l}\mathbb{I}[h_{ij} \neq y_{ij}]$
one-error $\downarrow$	$\frac{1}{m}\sum_{i=1}^{m}\mathbb{I}\left[\arg\max F(\boldsymbol{x}_i) \notin Y_{i\cdot}^{+}\right]$
coverage $\downarrow$	$\frac{1}{ml}\sum_{i=1}^{m}\mathbb{I}\left[\max_{j\in Y_{i\cdot}^{+}} \text{rank}_F(\boldsymbol{x}_i, j) - 1\right]$
ranking loss $\downarrow$	$\frac{1}{m}\sum_{i=1}^{m}\frac{\lvert\mathcal{S}_{\text{rank}}^{i}\rvert}{\lvert Y_{i\cdot}^{+}\rvert\lvert Y_{i\cdot}^{-}\rvert}$
average precision $\uparrow$	$\frac{1}{m}\sum_{i=1}^{m}\frac{1}{\lvert Y_{i\cdot}^{+}\rvert}\sum_{j\in Y_{i\cdot}^{+}}\frac{\lvert\mathcal{S}_{\text{precision}}^{ij}\rvert}{\text{rank}_F(\boldsymbol{x}_i, j)}$
macro-AUC $\uparrow$	$\frac{1}{l}\sum_{j=1}^{l}\frac{\lvert\mathcal{S}_{\text{macro}}^{j}\rvert}{\lvert Y_{\cdot j}^{+}\rvert\lvert Y_{\cdot j}^{-}\rvert}$

$\mathcal{S}_{\text{rank}}^{i} = \{(u, v) \in Y_i^{+} \times Y_i^{-} | f_u(\boldsymbol{x}_i) \leq f_v(\boldsymbol{x}_i)\}$

$\mathcal{S}_{\text{precision}}^{ij} = \{k \in Y_i^{+} | \text{rank}_F(\boldsymbol{x}_i, k) \leq \text{rank}_F(\boldsymbol{x}_i, j)\}$

$\mathcal{S}_{\text{macro}}^{j} = \left\{(a, b) \in Y_{\cdot j}^{+} \times Y_{\cdot j}^{-} | f_j(\boldsymbol{x}_a) \geq f_j(\boldsymbol{x}_b)\right\}$

***Hamming loss**: How many times an instance-label pair is misclassified

One-error: What fraction of instances are misclassified

Coverage: On average of all instances, how far we need to go down the ranked list to cover all ground-truth labels

Ranking loss: The average fraction of label pairs that are reversely ordered for the instance

Average precision: The average fraction of predicted labels that are ground truth.

***Macro-AUC**: The average AUC value over all classes

Measure-aware Feature Reuse [4]

If the *confidence* of a feature in current level is lower than some threshold, we reuse the counterpart from the previous level.

Algorithm 1 Measure-aware feature reuse

Input: measure M, forests' output $\mathbf{H}^t$, previous $\mathbf{G}^{t-1}$.
Output: new representation $\mathbf{G}^t$.
Procedure:

Initialize matrix $\mathbf{G}^t = \mathbf{H}^t$.
if Measure M is label-based **then**
  for $j = 1$ **to** l **do**
    compute confidence α_j^t on $\mathbf{H}_{\cdot j}^t$
    Update $\mathbf{G}_{\cdot j}^t$ to $\mathbf{G}_{\cdot j}^{t-1}$ when $\alpha_j^t < \theta_t$.
  end for
end if
if Measure M is instance-based **then**
  for $i = 1$ **to** m **do**
    compute confidence α_i^t on $\mathbf{H}_{i\cdot}^t$
    Update $\mathbf{G}_{i\cdot}^t$ to $\mathbf{G}_{i\cdot}^{t-1}$ when $\alpha_i^t < \theta_t$.
  end for
end if

Confidence w.r.t. the Measures

Measure	Confidence
hamming loss	$\frac{1}{m}\sum_{i=1}^{m} p_{ij}\mathbb{I}[p_{ij} > 0.5] + (1-p_{ij})\mathbb{I}[p_{ij} \leq 0.5]$
one-error	$\max_{i=1,\ldots,m} p_{ij}$
coverage	$1 - \frac{1}{l}\sum_{j=0}^{l}\left[j \cdot p_{ij} \prod_{k=j+1}^{l}(1-p_{ik})\right]$
ranking loss	$\sum_{j=0}^{l}\prod_{k=1}^{j} p_{ik} \prod_{k=j+1}^{l}(1-p_{ik})$
average precision	$\sum_{j=0}^{l}\prod_{k=1}^{j} p_{ik} \prod_{k=j+1}^{l}(1-p_{ik})$
macro-AUC	$\sum_{i=0}^{m}\prod_{k=1}^{i} p_{kj} \prod_{k=i+1}^{m}\left(1-p_{kj}\right)$

Determine the Threshold

Algorithm 2 Determine threshold

Input: measure M, forests' output $\mathbf{H}^t$, ground-truth $\mathbf{Y}$, previous performance on layer_{t-1}.
Output: threshold θ_t.
Procedure:

Initialize confidence set $\mathcal{S} = \emptyset$.
if Measure M is label-based **then**
  for $j = 1$ **to** l **do**
    compute confidence α_j^t on $\mathbf{H}_{\cdot j}^t$.
    compute measure m_j^t on $(\mathbf{H}_{\cdot j}^t, \mathbf{Y}_{\cdot j})$.
    $\mathcal{S} = \mathcal{S} \cup \{\alpha_j^t\}$ when m_j^t is worse than m_j^{t-1}.
  end for
end if
if Measure M is instance-based **then**
  for $i = 1$ **to** m **do**
    compute confidence α_i^t on $\mathbf{H}_{i\cdot}^t$.
    compute measure m_i^t on $(\mathbf{H}_{i\cdot}^t, \mathbf{Y}_{i\cdot})$.
    $\mathcal{S} = \mathcal{S} \cup \{\alpha_i^t\}$ when m_j^t is worse than m_j^{t-1}.
  end for
end if
θ_t = Compute threshold on $\mathcal{S}$.

Measure-aware Layer Growth [4]

If the recent 3 layers do not improve the measure, stop growing the tree.

Algorithm 3 Measure-aware layer growth

Input: maximal depth T, measure M, training data $\{\mathbf{X}, \mathbf{Y}\}$.
Output: model set S, threshold set Θ and final layer index L.
Procedure:

Initialize parameters:
  performance in each layer $\boldsymbol{p}$[1:T],
  best performance on train set p_{best},
  the initial threshold $\theta_1 = 0$,
  the best performance layer index $L = 1$,
  the model set $\mathcal{S} = \emptyset$.
for $t = 1$ **to** T **do**
  Train forests in layer$_t$ and get classifier h_t.
  Predict $\mathbf{H}_t = h_t([\mathbf{X}, \mathbf{G}_{t-1}])$.
  θ_t = Determine Threshold (Algorithm 2) when $t > 2$.
  $\mathbf{G}^t$ = measure-aware feature reuse (Algorithm 1).
  Compute performance $\boldsymbol{p}[t]$ on measure M with $\mathbf{G}^t$.
  if $\boldsymbol{p}[t]$ is better than p_{best} **then**
    Update best performance $p_{\text{best}} = \boldsymbol{p}[t]$.
    Update the layer index of best performance $L = t$.
  else if p_{best} is not updated in recent 3 layers **then**
    break
  end if
  Add layer$_t$ to model set: $\mathcal{S} = \mathcal{S} \cup \text{layer}_t$.
end for
Keep $\{\text{layer}_1, \ldots, \text{layer}_L\}$ in model set S and drop others $\{\text{layer}_{L+1}, \ldots\}$.

Table of Contents

Anomaly Detection

An unsupervised task that aims to identify the outliers.

- Traditional models: neural network autoencoders
- New model: completely random forest (RecForest)

Idea of **reconstruction**-based anomaly detectors: Anomalies are much harder to reconstruct, which contributes to large reconstruction error.

Advantages of RecForest [3]

- Higher training efficiency and significantly fewer hyperparameters than NN-based models $\Rightarrow$ Faster runtime
- Can find outlying attributes and handle irrelevant attributes, while other forest-based models cannot
- Fast runtime for finding outlying attributes

Reconstruction: Bounding Box

A **bounding box** for a sample x is a rectangle box (k-cell) in the feature space that contain x.

◦ In the context of a *tree* model, a bounding box for x is the rectangle box that corresponds to a leaf node that x belongs to.
◦ In the context of a *forest* model, a bounding box for x is the *intersection* of the bounding boxes associated with each tree model.

The reconstruction of x is defined as the *center* of the bounding box for x.

The reconstruction error is defined as $\|x - x_{rec}\|_2^2$

RecForest: Anomaly Detection

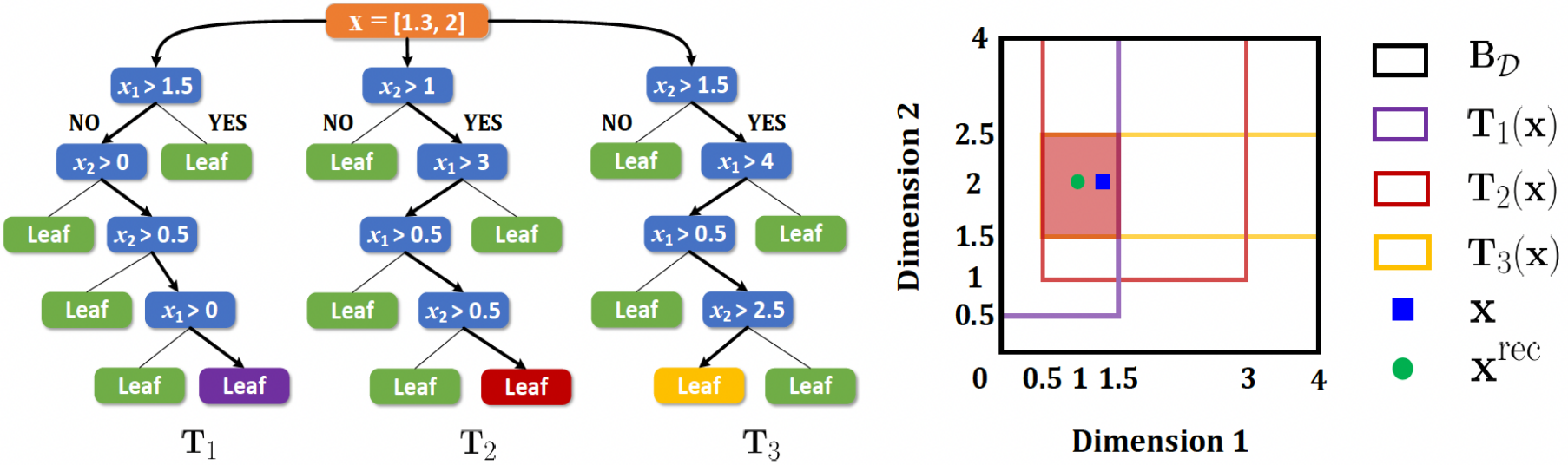
Train: Given the dataset $\mathcal{D}$,

- build the bounding box $B_{\mathcal{D}}$ by finding $L_i^{B_{\mathcal{D}}} = \min\{x_i \mid \forall x \in \mathcal{D}\}$ and $U_i^{B_{\mathcal{D}}} = \max\{x_i \mid \forall x \in \mathcal{D}\}$ for each attribute i.
- build the completely random forest with m trees

Test: Given a test set $\{x_i\}$,

- put it in each tree and identify the leaves it belongs to
- find its bounding box, which is the intersection of $B_{\mathcal{D}}$ and all rectangles corresponding to the leaves
- find x_{rec} and the reconstruction error
- determine whether an instance is anomaly by a threshold in terms of ranking of the reconstruction error. Samples rank on the top are more likely to be anomalies.

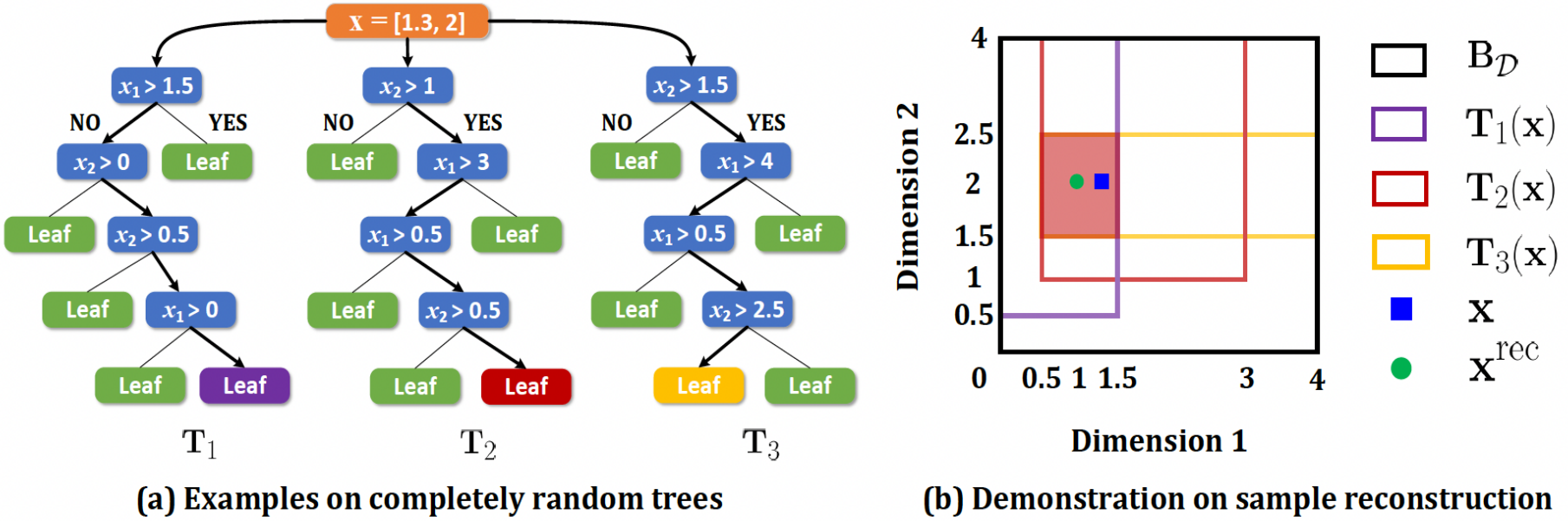
RecForest: Anomaly Detection



(a) Examples on completely random trees **(b) Demonstration on sample reconstruction**

Figure 1: Demonstration of sample reconstruction using a completely random forest. (a) The structure of completely random trees in the forest. Decision paths of a given test sample $\mathbf{x}$ are bolded. (b) Bounding boxes from trees $\{\mathbf{T}_1(\mathbf{x}), \mathbf{T}_2(\mathbf{x}), \mathbf{T}_3(\mathbf{x})\}$, and from the training data $\mathbf{B}_{\mathcal{D}}$. The intersection area is colored red.

RecForest: Anomaly Detection



(a) Examples on completely random trees **(b) Demonstration on sample reconstruction**

Figure 1: Demonstration of sample reconstruction using a completely random forest. (a) The structure of completely random trees in the forest. Decision paths of a given test sample $\mathbf{x}$ are bolded. (b) Bounding boxes from trees $\{\mathbf{T}_1(\mathbf{x}), \mathbf{T}_2(\mathbf{x}), \mathbf{T}_3(\mathbf{x})\}$, and from the training data $\mathbf{B}_{\mathcal{D}}$. The intersection area is colored red.

Runtime: Let the size of training/testing set be n and t, respectively. Let h be the maximum tree height and m be the number of trees in a forest.

Training: $\mathcal{O}(nmh)$; Testing: $\mathcal{O}(tmh)$

RecForest: Outlying Attributes Mining

For a suspected anomaly point, we want to know on which attribute(s) its value(s) deviate much from the normal range.

The outlying score for attribute i and an example x is

$$O_i = \frac{\exp\{(x_i - x_i^{rec})^2\}}{\sum_{i=1}^{d} \exp\{(x_i - x_i^{rec})^2\}}$$

Attributes with a higher score tend to be more outlying.

*This helps people to explain why certain examples are anomalies.

Table of Contents

Problem Formulation

▷ Domain adaption

- Suppose there are c providers. User i builds his learnware (model) on his own task T_i using a *private* local dataset $S_i = \{(x_{i,n}, y_{i,n})\}_{n=1}^{N}$.
- T_i is represented by (P_i, f), where P_i is the distribution on input space $\mathcal{X}$ and f is a *shared global* optimal rule function $f : \mathcal{X} \to \mathcal{Y}$ s.t. $\forall i \in [c] : \forall (x, y) \in S_i : f(x) = y$.
  * $\mathcal{X}$ and f are the same; P_i's are different.
- Assume all models are trained so that they have small true error $\mathcal{L}(P_i, f, \hat{f}_i)$
- Find models that are useful for current task T_t by the *specifications* we create that are associated with the models

▷ Two kinds of assumptions to solve this problem

- **Tesk-recurrent assumption**: $\exists i \in [c]$ s.t. $P_t = P_i$.
- **Instance-recurrent assumption**: $P_t = \sum_{i=1}^{c} w_i P_i$, where $w_i \in [0, 1]$ sum up to 1.

How to Create the Specification: KME

Idea: We want the specification to represent the true distribution P_i.

$\triangleright$ Kernel Mean Embeddings (KME): the mean of a $\mathcal{H}_k$-valued random variable that maps probability distributions [1] to an element in $\mathcal{H}_k$ associated with kernel k

Let a kernel $k(\cdot, \cdot)$ be given. Let P be an $\mathcal{X}$-valued random varible. It KME is defined as

$$\mu_k(P) := \int_{\mathcal{X}} k(x, \cdot) dP(x)$$

- Assume k is bounded, i.e. $\int_{\mathcal{X}} \sqrt{k(x,x)} dP(x) < \infty$
- By reproducing property, $\forall f \in \mathcal{H}_k : < f, \mu_k(P) >_{\mathcal{H}_k} = E_P[f(X)]$
- If k is a characteristic kernel, by definition the KME is injective w.r.t. P. Then $\|\mu_k(P) - \mu_k(Q)\| = 0 \Leftrightarrow P = Q$. This makes KME capable to represent a distribution.

[1] I may prefer to say "probability measure" because of the definition of random variable

How to Create the Specification: KME [2]

◦ Empirical approximation:

Since we have no access to the true distribution P_i, we can only draw i.i.d samples $X = \{x_n\}_{n=1}^{N} \sim P^N$. Denote empirical distribution as P_X.

$$\mu_k(P_X) = \frac{1}{N} \sum_{n=1}^{N} k(x_n, \cdot)$$

$\mu_k(P_X)$ *converges to* $\mu_k(P)$ at rate $\mathcal{O}(1/\sqrt{N})$

How to Create the Specification: KME [2]

◦ Empirical approximation:

Since we have no access to the true distribution P_i, we can only draw i.i.d samples $X = \{x_n\}_{n=1}^N \sim P^N$. Denote empirical distribution as P_X.

$$\mu_k(P_X) = \frac{1}{N}\sum_{n=1}^{N} k(x_n, \cdot)$$

$\mu_k(P_X)$ *converges to* $\mu_k(P)$ at rate $\mathcal{O}(1/\sqrt{N})$

▷ Reduced Set Construction

◦ What if the necessary $|X|$ is too large? A method to speed up.

◦ Approximate P_X by P_R, where $R = (B, Z) = \{(\beta_m, z_m)\}_{m=1}^M$ is a *weighted* set of points. $M << N$.

◦ Minimize the distance

$$\|\mu_k(P_X) - \mu_k(P_R)\|_{\mathcal{H}_k}^2 = \left\| \sum_{n=1}^{N} \frac{1}{N} k(x_n, \cdot) - \sum_{m=1}^{M} \beta_m k(z_m, \cdot) \right\|_{\mathcal{H}_k}^2$$

How to Create the Specification: KME

▷ Kernel Herding

- An algorithm that learns to approximate a distribution with a collection of samples
- Iteratively draw an example that *greedily* reduces

$$\|\mu_k(P_X) - \mu_k(P)\|^2_{\mathcal{H}_k}$$

- With kernel herding, this quantity decreases at a rate of $\mathcal{O}(1/T)$

$$x_{T+1} = \begin{cases} \arg\max_{x \in \mathcal{X}} \Phi(x), & \text{if } T = 0 \\ \arg\max_{x \in \mathcal{X}} \Phi(x) - \frac{1}{T+1} \sum_{t=1}^{T} k(x_t, x), & \text{if } T \geq 1. \end{cases} \tag{13}$$

where x_{T+1} is the next example we want to sample from P when $\{x_t\}_{t=1}^{T}$ have been already sampled. And Proposition 4.8 in [7] shows the following error $\mathcal{E}_T$ will decrease as $O(1/T)$:

$$\mathcal{E}_T = \left\| \frac{1}{T} \sum_{t=1}^{T} k(x_t, \cdot) - \Phi \right\|^2_{\mathcal{H}_k}.$$

Upload Phase: Compute the Specifications

Given a local dataset $S_i = \{(x_{i,n}, y_{i,n})\}_{n=1}^{N}$, we first build the empirical KME $\mu_k(P_X)$. Then we need to find a *reduced set* that minimizes

$$\|\mu_k(P_X) - \mu_k(P_R)\|_{\mathcal{H}_k}^2 = \sum_{n=1}^{N}\sum_{m=1}^{N}\frac{1}{N^2}k(x_n, x_m) + \sum_{n=1}^{M}\sum_{m=1}^{M}\beta_n\beta_m k(z_n, z_m) - 2\sum_{n=1}^{N}\sum_{m=1}^{M}\frac{\beta_m}{N}k(x_n, z_m)$$

We suppose this function is differentiable w.r.t. $B = (\beta_1, \ldots \beta_M)$ and $Z = \{z_1, \ldots, z_m\}$. Then we do alternating optimization (like coordinate descent).

Algorithm: Alternating Update

Suppose F is differentiable, we adopt the alternating optimization to minimize (8).

Fix Z update β. Suppose vectors in Z are fixed, setting $\frac{\partial F(\boldsymbol{\beta},Z)}{\partial \boldsymbol{\beta}} = 0$ obtains the closed-form solution of $\boldsymbol{\beta}$ using pseudo-inverse of K:

$$\boldsymbol{\beta} = K^{\dagger}C, \tag{9}$$

where

$$K \in \mathbb{R}^{N\times M}, K_{nm} = k(z_n, z_m),$$
$$C \in \mathbb{R}^{N}, C_n = \frac{1}{N}\sum_{m=1}^{M} k(z_n, x_m).$$

Fix β update Z. When $\boldsymbol{\beta}$ is fixed, $\{z_1, \cdots, z_M\}$ in Z are independent in (8), therefore we can iteratively run gradient descent on each z_m as

$$z_m^{(t)} = z_m^{(t-1)} - \eta \frac{\partial F(\boldsymbol{\beta}, Z)}{\partial z_m}. \tag{10}$$

Algorithm: Alternating Update

Suppose F is differentiable, we adopt the alternating optimization to minimize (8).

Fix Z update $\boldsymbol{\beta}$. Suppose vectors in Z are fixed, setting $\frac{\partial F(\boldsymbol{\beta},Z)}{\partial \boldsymbol{\beta}} = 0$ obtains the closed-form solution of $\boldsymbol{\beta}$ using pseudo-inverse of K:

$$\boldsymbol{\beta} = K^{\dagger} C, \tag{9}$$

where

$$K \in \mathbb{R}^{N\times M}, K_{nm} = k(z_n, z_m),$$
$$C \in \mathbb{R}^{N}, C_n = \frac{1}{N}\sum_{m=1}^{M} k(z_n, x_m).$$

Fix $\boldsymbol{\beta}$ update Z. When $\boldsymbol{\beta}$ is fixed, $\{z_1, \cdots, z_M\}$ in Z are independent in (8), therefore we can iteratively run gradient descent on each z_m as

$$z_m^{(t)} = z_m^{(t-1)} - \eta \frac{\partial F(\boldsymbol{\beta}, Z)}{\partial z_m}. \tag{10}$$

Algorithm 1 Reduced KME Construction

input:

Local dataset $X = \{x_n\}_{n=1}^{N}$, kernel function k, size of reduced set M, learning rate η, and iteration T

output:

Reduced KME $\Phi(\cdot) = \sum_{m=1}^{M} \beta_m k(z_m, \cdot)$

procedure:

1: Initialize $z_m^{(0)}$ by running k-means on X, where the number of clusters is set to M.
2: **for** $t = 1 : T$ **do**
3: Update $\boldsymbol{\beta}$ by (9)
4: Update each $z_m^{(t)}$ by (10)
5: **end for**

Each model is uploaded with the RKME specification Φ_i, represented by B and Z.

Deployment Phase: Use RKME to identify useful models

Algorithm 2 Deployment Procedure

input:

Current task test data $X_t = \{x_{t,n}\}_{n=1}^{N}$, a learnware pool of pre-trained models $\{\widehat{f}_i\}_{i=1}^{c}$ and RKMEs $\{\Phi_i\}_{i=1}^{c}$.

output:

Prediction $Y_t = \{y_{t,n}\}_{n=1}^{N}$.

procedure:

if task-recurrent assumption **then**
  $\Phi_t = \sum_{n=1}^{N} \frac{1}{N} k(x_{t,n}, \cdot)$
  $i^* = \arg\min_i \left\| \Phi_t - \Phi_i \right\|_{\mathcal{H}_k}^2$
  $y_{t,n} = \widehat{f}_{i^*}(x_{t,n}), n \in [N]$
end if
if instance-recurrent assumption **then**
  Estimate $\widehat{\boldsymbol{w}}$ as (12)
  Initialize the mimic sample set $S = \emptyset$
  while $|S| < cN$ **do**
    Sample a provider index i by weight $\widehat{w}_i$
    Sample an example x by kernel herding as (13)
    $S = S \cup \{(x, i)\}$
  end while
  Train a selector g on mimic sample S
  for $n = 1 : N$ **do**
    $i^* = g(x_{t,n})$
    $y_{t,n} = \widehat{f}_{i^*}(x_{t,n})$
  end for
end if

References

[1] Ming Pang et al. "Improving Deep Forest by Screening". In: *IEEE Transactions on Knowledge and Data Engineering* (2020).

[2] Xi-Zhu Wu et al. "Model reuse with reduced kernel mean embedding specification". In: *IEEE Transactions on Knowledge and Data Engineering* (2021).

[3] Yi-Xuan Xu et al. "Reconstruction-based Anomaly Detection with Completely Random Forest". In: *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*. SIAM. 2021, pp. 127–135.

[4] Liang Yang et al. "Multi-label learning with deep forest". In: *arXiv preprint arXiv:1911.06557* (2019).

[5] Zhi-Hua Zhou and Ji Feng. "Deep forest". In: *arXiv preprint arXiv:1702.08835* (2017).