Testing using Privileged Information by Adapting Features with Statistical Dependence

Kwang In Kim
UNIST

James Tompkin
Brown University

Abstract

Given an imperfect predictor, we exploit additional features at test time *to improve the predictions made, without retraining and without knowledge of the prediction function. This scenario arises if training labels or data are proprietary, restricted, or no longer available, or if training itself is prohibitively expensive. We assume that the additional features are useful if they exhibit strong statistical dependence to the underlying perfect predictor. Then, we empirically estimate and strengthen the statistical dependence between the initial noisy predictor and the additional features via manifold denoising. As an example, we show that this approach leads to improvement in real-world visual attribute ranking.*

1. Introduction

In supervised learning, sometimes we have additional information *at training time* but not at testing time. One example is human action recognition, where training images have additional skeletal features like bone lengths [17]. Another class of examples is tasks with crowdsourced labels, which have derived confidence values from multiple noisy human labelings [13]. In training, each input and output pair $(\mathbf{g}, y)$ has additional 'privileged' features $\mathbf{h}$ to help learn mapping function f. Vapnik and Vashist proposed to exploit these data with *learning using privileged information* [21]. Their support vector machine (SVM) training algorithm was later generalized to other learning algorithms.

We consider the complementary scenario where additional features $\mathbf{h}$ are only available *at testing time*. We call this testing using privileged information (TUPI). This is different from retraining with new features: Given a pre-trained f and test data $\mathbf{g}$ with *test time* features $\mathbf{h}$, TUPI does not assume access to the original large set of labeled training data $\{(\mathbf{g}_i, y_i)\}$. TUPI is useful when the predictor is trained on a single feature but multiple (perhaps heterogeneous) features are available when the predictor is deployed.

For instance, when a predictor was trained on a single lens camera but applied to multi-lens cameras (across smartphone models or robot build variations), or when a predictor was trained on RGB camera data yet run on RGB-D camera data. TUPI is also useful when a predictor is proprietary or built into a closed-source library and the training data are inaccessible, e.g., Microsoft Kinect pose estimator, but new features become available later on such as deep-learning-based features (we discuss more applications in Sec. 5).

Problem description. Suppose we have an estimation problem with an input feature space $\mathcal{G}$ and the corresponding output space $\mathcal{Y} \subset \mathbb{R}$. Traditional algorithms aim to identify the underlying function $f^* : \mathcal{G} \to \mathcal{Y}$ based on the input training features $G^{tr} = \{\mathbf{g}_1^{tr}, \ldots, \mathbf{g}_l^{tr}\} \subset \mathcal{G}$ and the corresponding task-specific labels $Y^{tr} \subset \mathcal{Y}$. Once an estimate f^I of f^* is constructed, we can apply it to unseen test data points $G = \{\mathbf{g}_1, \ldots, \mathbf{g}_n\}$ to construct the prediction $\mathbf{f}^I := f^I|_G = [f^I(\mathbf{g}_1), \ldots, f^I(\mathbf{g}_n)]^\top$.

In the TUPI scenario, we assume that additional feature sets $\{H^i\}_{i=1}^m$ are provided for the test set G such that each test instance $\mathbf{g}_k \in G$ is accompanied by m additional features $\{\mathbf{h}_k^1, \ldots, \mathbf{h}_k^m\}$. Our goal is to exploit $\{H^i\}$ to improve prediction $\mathbf{f}^I$. However, each new feature set H^i may or may not be related to the underlying function f^* and so may or may not be useful to improve $\mathbf{f}^I$. Further, we do not assume explicit forms for the feature extractors g or h^i (by which G and H^i are obtained), or for the learned function f (e.g. f could be a deep neural network (DNN) regressor or a rule-based classifier). Lastly, we do not assume access to a large set of labeled data points (G^{tr}, Y^{tr}) at the testing stage. Otherwise, the problem could be solved by adding $\{H^i\}_{i=1}^m$ to the feature set and applying traditional supervised feature selection algorithms [18]. In general, exploiting additional information during testing to improve a prediction is an ill-posed problem, and existing algorithms are not able to accomplish this without having access to the data or labels, or are only applicable to very specific test time features.

Denoising statistical dependence. Our insight is that if the test time feature sets are useful, then they will exhibit strong *statistical dependence* with the underlying perfect predictor f^*. For instance, in the extreme case of known perfect statistical dependence, knowing H^i would immediately identify f^*. Since we do not know in advance which feature sets (if

any) are actually useful, we must estimate these dependencies. We regard the initial predictor f^I as a noisy version of f^*, and we empirically estimate the pairwise dependencies between f^I and each feature set H^i. Then, we selectively strengthen—or *denoise*—the pairwise statistical dependencies to improve f^I. That is, we:

1. Embed f^I and $\{H^i\}_{i=1}^m$ into a model manifold M where the Hilbert-Schmidt independence criterion, a consistent measure of statistical dependence, constitutes a similarity measure [6].
2. Strengthen dependencies via denoising on M [7].

We demonstrate the TUPI scenario with visual attribute ranking [16], in which users provide pair-wise rank comparisons for attribute-based database ordering (our supplemental material contains more problem details). This problem is a good fit for TUPI because ranking applications commonly have multiple related feature representations and attributes. For this, we simply use the corresponding rank loss (Eq. 12). Through experiments across seven real-world datasets, we show that our approach can lead to significantly improved estimation accuracy over the initial predictors.

Semi-supervised adaptation. Given the initial prediction $\mathbf{f}^I$ and the test time features $\{H^i\}$, our algorithm improves $\mathbf{f}^I$ in an unsupervised manner. However, like many unsupervised approaches, our method has (two) hyperparameters that must be tuned for best-possible performance. This is a difficult issue that has no good approach in the unsupervised setting. In practical applications, we must rely on users to sample and evaluate hyperparameters assuming that their selections would be guided by experience on related problems.

For our method, user sampling is feasible as our method induces smoothness in accuracy over hyperparameters and is fast to execute (≈1 second on 50K items; see supplemental Fig. 1). This setting makes objective assessment challenging as the results are subject to user experience. Therefore, for comparison to other techniques, we use validation labels to tune these parameters. This renders the validation scenario of our adaptation algorithm semi-supervised. In our experiments, we further show that using such a small set of validation labels to retrain a new predictor is not competitive.

Related work. As TUPI uses small validation label sets for hyperparameter optimization, it might bring to mind semi-supervised learning and one might consider building a graph Laplacian from the test time features to help solve the task [25]. In our experiments, we demonstrate that TUPI provides a stronger alternative to this simple baseline. TUPI might also bring to mind multi-task learning (MTL), where features or functions learned for one problem are applicable to other problems [4, 14]. The closest related work in this setting is Kim et al.'s predictor combination algorithm [10] which regards predictions from potentially-related tasks as test time features. However, this algorithm cannot be applied to arbitrary multi-dimensional test time features (see Sec. 4).

The most closely-related work is the CoConut framework by Khamis and Lampert [9]. This regularizes the output label space during *co-classification* to prefer certain class proportions, without knowing anything about the label predictors, by enforcing smoothness via a graph Laplacian measured over neighboring points. These points can be defined by original features or by test time features. The latter case leads to a scenario similar to TUPI, and so we adapt CoConut to our relative attributes setting. We find that the two approaches are broadly complementary: CoConut exploits the *spatial smoothness* manifested via local neighborhood structure (i.e. via a graph Laplacian) while our algorithm exploits *statistical dependence* that is measured across the whole dataset. We show this explicitly in supplemental experiments which combine both algorithms (Fig. 3, supplemental).

2. Background

Hilbert-Schmidt independence criterion (HSIC). HSIC is a consistent measure of statistical (in-)dependence based on the reproducing kernel Hilbert space (RKHS) embeddings of probability distributions (see supplemental for details and related applications) [6]. Suppose we have two data spaces $\mathcal{V}$ and $\mathcal{W}$, equipped with joint probability distribution $\mathbb{P}_{\mathbf{vw}}$ and marginals $\mathbb{P}_{\mathbf{v}}$ and $\mathbb{P}_{\mathbf{w}}$, and we wish to estimate their statistical dependence. Unlike other popular dependence measures like mutual information, HSIC does not require estimating the underlying joint probability density. This is valuable because, in the TUPI scenario, this density may not even exist (see Sec. 3).

For $\mathcal{V}$, we define a separable RKHS of functions $\mathcal{K}_{\mathbf{v}}$ characterized by the feature map $\phi : \mathcal{V} \to \mathcal{K}_{\mathbf{v}}$ and the positive definite kernel function $k_{\mathbf{v}}(\mathbf{v}, \mathbf{v}') := \langle \phi(\mathbf{v}), \phi(\mathbf{v}') \rangle$. Similarly, for $\mathcal{W}$, we define $\mathcal{K}_{\mathbf{w}}$, the feature map ψ, and the corresponding kernel $k_{\mathbf{w}}$. The HSIC associated with $\mathcal{K}_{\mathbf{v}}$, $\mathcal{K}_{\mathbf{w}}$, and $\mathbb{P}_{\mathbf{vw}}$ is:

$$\begin{aligned}\text{HSIC}(\mathcal{K}_{\mathbf{v}},\mathcal{K}_{\mathbf{w}},\mathbb{P}_{\mathbf{vw}}) = \mathbb{E}_{\mathbf{vv'ww'}}[k_{\mathbf{v}}(\mathbf{v},\mathbf{v}')k_{\mathbf{w}}(\mathbf{w},\mathbf{w}')] \\ + \mathbb{E}_{\mathbf{vv'}}[k_{\mathbf{v}}(\mathbf{v},\mathbf{v}')]\mathbb{E}_{\mathbf{ww'}}[k_{\mathbf{w}}(\mathbf{w},\mathbf{w}')] \\ - 2\mathbb{E}_{\mathbf{vw}}\left[\mathbb{E}_{\mathbf{v}'}[k_{\mathbf{v}}(\mathbf{v},\mathbf{v}')]\mathbb{E}_{\mathbf{w}'}[k_{\mathbf{w}}(\mathbf{w},\mathbf{w}')]\right].\end{aligned}$$

For bounded and *universal* [20] kernels $k_{\mathbf{v}}$ and $k_{\mathbf{w}}$, such as Gaussian kernels (Eq. 5), HSIC is well-defined and is zero only when the two distributions $\mathbb{P}_{\mathbf{v}}$ and $\mathbb{P}_{\mathbf{w}}$ are independent: HSIC is the maximum mean discrepancy (MMD) between the joint probability measure $\mathbb{P}_{\mathbf{vw}}$ and the product of marginals $\mathbb{P}_{\mathbf{v}}\mathbb{P}_{\mathbf{w}}$ computed with the product kernel $k_{\mathbf{vw}} = K_{\mathbf{v}} \otimes K_{\mathbf{w}}$ [15]:

$$\begin{aligned}\text{HSIC}(\mathcal{K}_{\mathbf{v}},\mathcal{K}_{\mathbf{w}},\mathbb{P}_{\mathbf{vw}}) &= \text{MMD}^2(\mathbb{P}_{\mathbf{vw}},\mathbb{P}_{\mathbf{v}}\mathbb{P}_{\mathbf{w}}) \\ &= \|\mu_k[\mathbb{P}_{\mathbf{vw}}] - \mu_k[\mathbb{P}_{\mathbf{v}}\mathbb{P}_{\mathbf{w}}]\|_k^2, \quad (1)\end{aligned}$$

where $\|\cdot\|_k$ is the RKHS norm of $\mathcal{K}_k$ and $\mu_k[\mathbb{P}]$ is the *kernel mean embedding* of $\mathbb{P}$ based on k [15]. If the kernels $K_{\mathbf{x}}$

and $K_{\mathbf{y}}$ are universal, the MMD becomes a proper distance measure of probability distributions (i.e., $\text{MMD}(\mathbb{P}_A, \mathbb{P}_B) = 0$ only when $\mathbb{P}_A$ and $\mathbb{P}_B$ are identical), which applied to the distance between the joint and marginal distributions corresponds to the condition of independence.

In practice, we do not have access to the underlying probability distributions, but only to a sample $\{\mathbf{v}_i, \mathbf{w}_i\}_{i=1}^n$ drawn from $\mathbb{P}_{\mathbf{vw}}$. However, HSIC is applicable to sample-based estimates: empirical HSIC estimates have uniform convergence guarantees to the underlying true HSIC, which provides a reliable finite sample estimate. Thus, we construct a sample-based HSIC estimate [6]:

$$\widehat{\text{HSIC}} = \text{tr}[\mathbf{K_v C K_w C}],$$

where $[\mathbf{K_v}]_{ij} = k_{\mathbf{v}}(\mathbf{v}_i, \mathbf{v}_j)$, $[\mathbf{K_w}]_{ij} = k_{\mathbf{w}}(\mathbf{w}_i, \mathbf{w}_j)$, and $\mathbf{C} = I - \frac{1}{n}\mathbf{1}\mathbf{1}^\top$ with $\mathbf{1} = [1, \ldots, 1]^\top$.

Manifold denoising. This estimates the underlying manifold structure from noisy samples within the ambient space of a manifold [5, 23, 10, 7]. A set of data points $P = \{\mathbf{p}_0, \ldots, \mathbf{p}_m\} \subset \mathbb{R}^d$ are assumed to be noisy samples of 'true' data from an underlying embedded sub-manifold M of $\mathbb{R}^d$, i.e., $\mathbf{p}_i = \imath(\mathbf{q}_i) + \epsilon$ for $\mathbf{q}_i \in M$, with embedding $\imath : M \to \mathbb{R}^d$. Assuming i.i.d. Gaussian noise ϵ in the ambient space $\mathbb{R}^d$, Hein and Maier's manifold denoising algorithm suppresses noise in P—it pushes P towards M—without having access to M directly [7].

We begin by building the graph Laplacian matrix Δ from pairwise similarities of P in $\mathbb{R}^d$: $\Delta = I - D^{-1}W$:

$$\begin{aligned}[W]_{ij} &= \kappa(\|\mathbf{p}_i - \mathbf{p}_j\|^2, \sigma_P^2) \\ &:= \exp\left(-\frac{\|\mathbf{p}_i - \mathbf{p}_j\|^2}{\sigma_P^2}\right). \end{aligned} \tag{2}$$

The scale hyperparameter $\sigma_P^2 > 0$ and the diagonal matrix D perform the probabilistic normalization: $[D]_{ii} = \sum_j [W]_{ij}$. Using Δ as the generator of a diffusion process on P, the denoising algorithm iteratively improves the solution P by simulating the diffusion process governed by the differential equation with diffusion coefficient $\gamma > 0$.

$$\frac{\partial P}{\partial t} = -\gamma \Delta P \Rightarrow P(t+1) - P(t) = -\gamma \Delta P(t). \tag{3}$$

In our approach, we embed the predicted evaluations $\mathbf{f}^I$ and test time features $\{H^i\}$ as points on a manifold to facilitate denoising $\mathbf{f}^I$ (Sec. 3). In this case, only $\mathbf{f}$ is evolved while $\{H^i\}$ is fixed throughout diffusion. With this goal, at each discrete time step t, we construct P by placing $\mathbf{f}(t)$ as the zero-th element (i.e., $\mathbf{p}_0(t) = \mathbf{f}(t)$, $\mathbf{p}_0(0) = \mathbf{f}^I$), with the remaining elements in P corresponding to $\{H^i\}$, and formulate the denoising of a single point $\mathbf{p}_0$ given $P \setminus \{\mathbf{p}_0\}$ as minimizing the energy:

$$\mathcal{O}(\mathbf{p}) = \|\mathbf{p} - \mathbf{p}_0(t)\|^2 + \lambda \sum_{i=1}^m \alpha_i \|\mathbf{p} - \mathbf{p}_i\|^2, \tag{4}$$

where $\alpha_i = [W(t)]_{1i} / \sum_j [W(t)]_{1j}$ for $1 \leq i \leq m$ with $\lambda > 0$ being a hyperparameter. We obtain $\mathcal{O}$ based on implicit Euler time-discretization of the diffusion equation (Eq. 3), which is stable for any $\gamma > 0$. We fix γ at 1. The diffusion process is nonlinear: The weight matrix W and the corresponding Laplacian Δ evolve over t.

3. Feature adaptation at testing time

We are now prepared to estimate the statistical dependence between each test time feature set and the underlying function f^*, based on the sample evaluations $\{H^i\}$ and noisy $\mathbf{f}^I$. The estimated dependencies are noisy, too, as they are based on $\mathbf{f}^I$, and so we apply manifold denoising to suppress noise in both the estimated dependencies and in $\mathbf{f}^I$. To facilitate the HSIC-based denosing, we embed the feature extractors and the predictor function into a Riemannian manifold of covariance operators or kernels $\{k\}$.

Manifold of predictors and feature extractors. Suppose that $\mathcal{X}$ is the space of input data instances from which features are extracted, e.g., $\mathcal{X}$ is the space of pixel-valued images. Let us assume a feature extractor $g : \mathcal{X} \to \mathcal{G}$ from which we construct the estimated predictor f, plus an additional class of feature extractors $\{h^i : \mathcal{X} \to \mathcal{H}^i\}_{i=1}^m$. We assume that all feature extractors are measurable. While we do not have direct access to $\{g, f, h^1, \ldots, h^m\}$, we do have the empirical evaluations of f and $\{h^i\}$ on a sample $X \subset \mathcal{X}$: $\mathbf{f}^I = f|_{g(X)}$ and $h^i|_X = H^i$. Further, we assume that the original data instance space $\mathcal{X}$ is equipped with a probability distribution $\mathbb{P}_{\mathbf{x}}$ inducing the corresponding probability distributions $\mathbb{P}_g$ and $\mathbb{P}_{h^i}$ in $\mathcal{G}$ and $\mathcal{H}^i$, respectively. Then, f is given as an estimate constructed from the labeled data points (G^{tr}, Y^{tr}) sampled from the joint distribution $\mathbb{P}_{\mathbf{xy}}$.

Adopting the HSIC framework, we introduce a reproducing kernel Hilbert space (RKHS) on each element of the feature and predictor (evaluation) class: $\{\mathcal{Y}, \mathcal{H}^1, \ldots, \mathcal{H}^m\}$: For $f(\mathcal{G}) \subset \mathcal{Y}$, the RKHS $\mathcal{K}_f$ is defined with kernel $k_f : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$. Similarly, the RKHS $\mathcal{K}_i$ is defined with kernel $k_i : \mathcal{H}^i \times \mathcal{H}^i \to \mathbb{R}$. Using the fact that an RKHS is uniquely identified by its kernel, we can define new RKHSs of functions directly on the input space $\mathcal{X}$: A reproducing kernel $\overline{k}_f : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is induced from k_f by application of the feature map g and the predictor f: $\overline{k}_f(\mathbf{x}, \mathbf{x}') := k_f(f \circ g(\mathbf{x}), f \circ g(\mathbf{x}'))$ for $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$. The positive definiteness of $\overline{k}_f$ is guaranteed by the positive definiteness of k_f. Similarly, the RKHS $\overline{\mathcal{K}}_i$ on $\mathcal{X}$ corresponding to k_i is defined based on $\overline{k}_i(\mathbf{x}, \mathbf{x}') := k_i(h^i(\mathbf{x}), h^i(\mathbf{x}'))$. We use the Gaussian kernel k_f with the width parameter σ_f^2:

$$k_f(f(\mathbf{x}), f(\mathbf{x}')) = \kappa(\|f(\mathbf{x}) - f(\mathbf{x}')\|^2, \sigma_f^2). \tag{5}$$

At this point, our input space $\mathcal{X}$ is equipped with a data generating distribution $\mathbb{P}_{\mathbf{x}}$, itself connected to multiple

RKHSs, each constructed by a feature extractor (or predictor) and the corresponding RKHS. We will use this structure to characterize all feature extractors and predictors based on their respective induced kernels defined on $\mathcal{X}$, which enables us to compare them in a unified framework.

Manifold embedding. We embed predictor f and feature extractors $\{h^i\}$ into a space M of normalized kernels:

$$f \to \widetilde{k}_f := (\overline{k}_f - \mu_{\overline{k}_f})/(\|\overline{k}_f - \mu_{\overline{k}_f}\|_{\overline{k}_f}), \tag{6}$$

where $\mu_{\overline{k}_f}$ is the mean embedding of $\mathbb{P}_{\mathbf{x}}$ based on the kernel $\overline{k}_f$ [15], i.e., $\mu_{\overline{k}_f} = \mathbb{E}_{\mathbf{x}}[\overline{k}(\mathbf{x}, \cdot)]$, and $\|\overline{k}\|_{\overline{k}_f} = \mathbb{E}_{\mathbf{x}\mathbf{x}'}|k(\mathbf{x}, \mathbf{x}')|$, both of which are well-defined for bounded kernels k_f including Gaussian kernels (Eq. 5). Similarly, h^i is embedded into M based on $\widetilde{k}_i$.

The space M is a Hilbert submanifold of an ambient Hilbert space $\mathcal{M}$ with the inner product:[1]

$$\langle \overline{k}_f, \overline{k}_i \rangle_{\mathcal{M}} = \mathbb{E}_{\mathbf{x}\mathbf{x}'}[\overline{k}_f(\mathbf{x}, \mathbf{x}')\overline{k}_i(\mathbf{x}, \mathbf{x}')]. \tag{7}$$

Our M construction is motivated by two points. First, that the inner product $\langle \overline{k}_f, \overline{k}_i \rangle_{\mathcal{M}}$ between two centered kernels $\overline{k}_f - \mu_{\overline{k}_f}$ and $\overline{k}_i - \mu_{\overline{k}_i}$ is precisely the HSIC of the predictor f and the feature extractor h^i as random variables:

$$\begin{aligned}\langle \overline{k}_f, \overline{k}_i \rangle_{\mathcal{M}} &= \mathbb{E}_{\mathbf{x}\mathbf{x}'}\big[\,\big(\overline{k}_f(\mathbf{x}, \mathbf{x}') - \mathbb{E}_{\mathbf{x}''}[\overline{k}_f(\mathbf{x}, \mathbf{x}'')]\big) \cdot \\ &\quad \big(\overline{k}_i(\mathbf{x}, \mathbf{x}') - \mathbb{E}_{\mathbf{x}''}[\overline{k}_i(\mathbf{x}, \mathbf{x}'')]\big)\,\big] \\ &= \mathbb{E}_{ff'h^i(h^i)'}\big[\,\big(k_f(f, f') - \mathbb{E}_{f''}[k_f(f, f'')]\big) \cdot \\ &\quad \big(k_i(h^i, (h^i)') - \mathbb{E}_{(h^i)''}[k_i(h^i, (h^i)'')]\big)\,\big],\end{aligned}$$

where $f = f \circ g(\mathbf{x})$, $f' = f \circ g(\mathbf{x}')$, and $f'' = f \circ g(\mathbf{x}'')$.

Second, by noting that the scale normalization in the embedding (Eq. 6) is essential in our denoising application. In application scenarios like feature selection [18] or clustering [19], HSIC is used without normalization. However, in our denoising scenario, the prediction variables f are directly optimized based on how HSIC is influenced by the kernel evaluations $\overline{k}_f$. In this case, one could scale HSIC without influencing the resulting statistical dependencies. For instance, in the simple case of the standard dot-product kernel ($\overline{k}(\mathbf{x}, \mathbf{x}') = \mathbf{x}^\top \mathbf{x}'$), HSIC becomes the Frobenius norm of the standard cross-covariance matrix. This can be arbitrarily increased by multiplying f with a positive constant, but a constant scaling should not influence any reasonable measure of statistical dependence. With the normalization in Eq. 6, the inner product $\langle \widetilde{k}_f, \widetilde{k}_i \rangle_{\mathcal{M}}$ captures the same dependence information between f and h^i as HSIC, but with reduced influence of the scales of f and h^i's.

[1] M can have a (semi)-Riemannian structure if we identify the local neighborhood $N(p)$ of each point $p \in M$ with its tangent space $T_p(M)$. As we will only use the ambient space distance $\mathcal{M}$, an explicit construction of the metric in M is unnecessary.

The two random variables f and h^i have joint functional dependence on $\mathbf{x}$, and so their joint probability density may not exist. Using HSIC enables estimating the statistical dependence even in this case, as it is estimated entirely based on kernel evaluations. This is not directly possible for some other dependence measures, e.g., mutual information.

In practice, we have sample evaluations $\mathbf{f}^I = f|_G$ (of size n) and corresponding features $\{H^i\}$. From this, we obtain a finite dimensional manifold $\widehat{M}$ with point-embedding $\mathbf{f} \to \widetilde{\mathbf{K}}_{\mathbf{f}} := \mathbf{K}_{\mathbf{f}}\mathbf{C}/\|\mathbf{K}_{\mathbf{f}}\mathbf{C}\|_F$ where $[\mathbf{K}_{\mathbf{f}}]_{kl} = k_f([\mathbf{f}]_k, [\mathbf{f}]_l)$ and $\|\cdot\|_F$ is the Frobenius norm. The inner-product (Eq. 7) on $\widehat{M}$ becomes the HSIC estimate $\widehat{\text{HSIC}}$.

Exploiting test time information by denoising statistical dependence. With the manifold structure $\widehat{M}$, we are now ready to apply the denoising algorithm (Eq. 4). $\widehat{M}$ is a sub-manifold of a matrix Hilbert space $\widehat{\mathcal{M}}$ where the inner-product between two points $\mathbf{K}_A$ and $\mathbf{K}_B$ can be calculated as $\langle \mathbf{K}_A, \mathbf{K}_B \rangle_{\widehat{\mathcal{M}}} = \text{tr}[\mathbf{K}_A\mathbf{K}_B]$ (cf. Eq. 7). Then, we iteratively minimize an energy functional, replacing the Euclidean distance in $\mathcal{O}$ (Eq. 4) with the ambient metric (restricted to $\widehat{M}$) $d^2_{\widehat{\mathcal{M}}}(\widetilde{\mathbf{K}}_{\mathbf{f}}, \widetilde{\mathbf{K}}_i) := 1 - \langle \widetilde{\mathbf{K}}_{\mathbf{f}}, \widetilde{\mathbf{K}}_i \rangle_{\widehat{\mathcal{M}}}$:

$$\mathcal{O}(\mathbf{f}) = d^2_{\widehat{\mathcal{M}}}(\widetilde{\mathbf{K}}_{\mathbf{f}}, \widetilde{\mathbf{K}}_{\mathbf{f}}(t)) + \lambda \sum_{i=1}^{m} w^i(t) d^2_{\widehat{\mathcal{M}}}(\widetilde{\mathbf{K}}_{\mathbf{f}}, \widetilde{\mathbf{K}}_i), \tag{8}$$

$$w^i(t) = \frac{\kappa(d^2_{\widehat{\mathcal{M}}}(\widetilde{\mathbf{K}}_{\mathbf{f}}(t), \widetilde{\mathbf{K}}_i), \sigma_w^2)}{\sum_{j=1}^{m} \kappa(d^2_{\widehat{\mathcal{M}}}(\widetilde{\mathbf{K}}_{\mathbf{f}}(t), \widetilde{\mathbf{K}}_j), \sigma_w^2)}. \tag{9}$$

Note that $\mathbf{f}$ denotes the variable to be optimized (based on $\widetilde{\mathbf{K}}_{\mathbf{f}}$ which is a function of $\mathbf{f}$) while $\widetilde{\mathbf{K}}_{\mathbf{f}}(t)$ represents the results obtained from the previous time step t. In general, when k is non-linear (as for Gaussian kernels in Eq. 5), the optimization problem is non-convex. We optimize $\mathcal{O}$ via gradient descent with $\mathbf{f}(0)$ obtained as the initial prediction $\mathbf{f}^I$. The time and memory complexities of optimizing $\mathcal{O}$ are $O(mn^3)$ and $O(mn^2)$, respectively. Algorithm 1 summarizes this proposed TUPI process. With this form, we see that our approach does not need to know the underlying function f, nor its feature extractor g, nor the function underlying the test time information h. This fulfills the TUPI scenario.

Algorithm interpretation. We solve the minimization problem at iteration t by trading the deviation from the solution of iteration $t-1$ (first term in Eq. 8) with the statistical dependence of f and the feature extractors weighted by $\{w^i(t)\}$ (in Eq. 8; see Eq. 4). Each weight $w^i(t)$ is an increasing function of the estimated dependence strength at t-th step, which disregards outliers. The uniformity of the weights is controlled by the hyperparameter σ_w^2:

- As $\sigma_w^2 \to \infty$, all features contribute equally to the minimization, which might include outliers.

Algorithm 1 TUPI algorithm

Input: Initial predictor evaluations $\mathbf{f}^I$; class of test time features $\{H^i\}_{i=1}^m$; hyperparameters λ and σ_w^2 (Eq. 8); (maximum iteration number T; see Sec. 4);
Output: Denoised evaluations $\mathbf{f}^O$;
$t = 0$; $\mathbf{f}(t) = \mathbf{f}^I$;
repeat
  Calculate weights $\{w^i(t)\}$ based on Eq. 9;
  Update $\mathbf{f}(t)$ by minimizing $\mathcal{O}$ (Eq. 8);
  $t = t+1$;
until termination condition is met (e.g. if $t \geq T$);

- As $\sigma_w^2 \to 0$, the single most relevant (statistically dependent) feature influences the construction, which might neglect other less relevant but still beneficial features.

Due to kernelization (Eq. 5) and normalization ($\widetilde{\mathbf{K}}$) in point-embedding, our algorithm applies when the absolute scale of predictions is irrelevant, e.g., for ranking. For classification and regression, we would store the scales and means, normalize and denoise, then restore the scales and means.

Large-scale problems. When the time $O(mn^3)$ and memory $O(mn^2)$ complexities of optimizing $\mathcal{O}$ are limiting, we adopt the Nyström approximation of $\mathbf{K}_\mathbf{f}$:

$$\mathbf{K}_\mathbf{f} \approx \mathbf{K}_{\mathbf{f}B}\mathbf{K}_{BB}^{-1}\mathbf{K}_{\mathbf{f}B}^\top, \tag{10}$$

where $[\mathbf{K}_{\mathbf{f}B}]_{kl} = k_f(b_k, b_l)$ for the basis set $B = \{b_1, \ldots, b_K\}$ and $[\mathbf{K}_{\mathbf{f}B}]_{kl} = k_f([\mathbf{f}]_k, b_l)$. The rank K of the approximation is prescribed based on the computational and memory capacity limits. Similarly, each $\mathbf{K}_i$ is approximated based on the corresponding basis set ($\mathbf{K}_i \approx \mathbf{K}_{iB}[\mathbf{K}_{BB}^i]^{-1}\mathbf{K}_{iB}^\top$). For example, the second (unnormalized) trace term in Eq. 8 and its derivative with respect to $\mathbf{f}$ are written as:

$$\begin{aligned} &\mathrm{tr}[\mathbf{K}_\mathbf{f}\mathbf{C}\mathbf{K}_i\mathbf{C}] \approx \mathcal{C}(\mathbf{f}) = \mathrm{tr}[\mathbf{K}_{\mathbf{f}B}\mathbf{S}_{\mathbf{f}i}] \\ &\frac{\partial \mathcal{C}(\mathbf{f})}{\partial [\mathbf{f}]_k} = 2[\partial \mathbf{K}_{\mathbf{f}B}]_{(k,:)}[\mathbf{S}_{\mathbf{f}i}]_{(:,k)}, \end{aligned} \tag{11}$$

where $\mathbf{S}_{\mathbf{f}i} = \mathbf{K}_{BB}^{-1}\mathbf{K}_{\mathbf{f}B}^\top\mathbf{C}\mathbf{K}_{iB}[\mathbf{K}_{BB}^i]^{-1}\mathbf{K}_{iB}^\top\mathbf{C}$, $[A]_{(k,:)}$ denotes the k-th row of A and $[\partial\mathbf{K}_{\mathbf{f}B}]_{kl}$ corresponds to the derivative of $k_f([\mathbf{f}]_k, b_l)$ (see Eq. 5). The computational bottleneck in the gradient evaluation is the multiplication $\mathbf{K}_{\mathbf{f}B}^\top\mathbf{C}\mathbf{K}_{iB}$ for each $i = 1, \ldots, m$, which takes $O(mnK^2)$ time. Thus, complexity is linear in the number of data points n and number of test time features m.

Convergence. While the trajectory of the solution during iteration depends on the initial solution and the mean curvature of the manifold [7], in the limit case (as $t \to \infty$), the embedding $\widetilde{\mathbf{K}}_\mathbf{f}(t)$ of the solution $\mathbf{f}(t)$ becomes a weighted average of feature kernels; i.e., the solution becomes independent of the initial predictions $\widetilde{\mathbf{K}}_\mathbf{f}(0)$, which is not useful. This is analogous to conventional diffusion where, when all points are evolved, the solution converges towards a constant as $t \to \infty$ [7]. In our case, we evolve only the predictor embedding $\widetilde{\mathbf{K}}_\mathbf{f}(t)$ and hold the remaining test time feature embeddings $\{\widetilde{\mathbf{K}}_i\}$ fixed; hence, $\widetilde{\mathbf{K}}_\mathbf{f}(t)$ becomes the weighted average of $\{\widetilde{\mathbf{K}}_i\}$. Therefore, we must terminate the iteration before convergence (see Sec. 4).

4. Experimental results

Setting. We test our approach on the *Relative Attributes* rank setting [16]. Our algorithm receives the initial rank evaluation $\mathbf{f}^I$ and the test time feature set $\{H^i\}$, and outputs an improved rank estimate $\mathbf{f}^O$. Throughout all experiments, the initialization $\mathbf{f}^I$ is predicted from 200 data points with pairwise comparison labels, by either a deep neural network (DNN) or a rank support vector machine (RSVM) [2]—whichever gave the higher validation accuracy. Hyperparameters were optimized on the validation set: RSVM regularization parameter, DNN training epochs, DNN MLP layers (2–8) and neurons per layer (5–160). To optimize the DNN, we use a standard mini-batch gradient descent with batch normalization. For both DNN and RSVM, we use the soft hinge loss l_H: an ordered training pair (q, r) implies that the ranking of $\mathbf{g}_q$ should be higher than $\mathbf{g}_r$:

$$l_H((\mathbf{g}_q, \mathbf{g}_r); f) = \max\left(0, 1 - (f(\mathbf{g}_q) - f(\mathbf{g}_r))\right)^2. \tag{12}$$

For all datasets, we ran experiments 10 times with different training and validation sets and averaged the results. Accuracy is measured as the ratio of correct pairwise rank comparisons with respect to all possible pairs.

TUPI parameters and effect discussion. Our algorithm requires setting values for the large-scale factorization rank K (Eq. 10), kernel scale σ_f^2 (Eq. 5), weight uniformity σ_w^2 (Eq. 8), number of iterations T, and regularization λ.

We fix the approximation rank K at 50 (Eq. 10). For multi-dimensional features, the basis points $\{b_k\}$ are constructed as their cluster centers. For one-dimensional features and for rank predictors $\mathbf{f}$, we obtain the basis points as the endpoints of linearly sampled intervals in the respective ranges. While we expected the sample-based HSIC to increase in accuracy as K increases, the performance for $K = 200$ was not significantly higher than for $K = 50$.

For kernel scale parameter σ_f^2, we use the standard heuristic and set it to twice the standard deviations of the pairwise distances of elements of $\mathbf{f}$. We set the scale parameter for each feature similarly.

We tune the remaining hyperparameters σ_w^2 and λ based on 50 validation data points, with T set implicitly: We set the maximum T value at 50 and monitored the progress of validation accuracy: we terminate iteration immediately

whenever the validation accuracy did not increase from the previous iteration. The influences of σ_w^2 and T are complementary: Larger σ_w^2 and T values set the focus on more strongly-dependent features, which leads to similar results as with smaller σ_w^2 and T values.

Baselines. Since we are not aware of any existing algorithms for rank testing with test time information, we form comparisons with existing methods that apply to less general or alternative settings. We aim to show that naïvely applying existing algorithms to TUPI is challenging.

Our baselines are: 1) the initial rank prediction $\mathbf{f}^I$, for which all results are shown as *relative difference* from this; 2) retraining a DNN or RSVM (whichever is better) on $\mathbf{h}$ with the validation labels (as we assume that a small validation set is available); 3) using semi-supervised learning (SSL) to build a graph Laplacian with the test time features and validation labels [25], 4) Khamis and Lampert's CoConut algorithm [9] adapted to TUPI in our rank prediction setting (in principle, their method is complementary to ours), and 5) Kim et al.'s predictor combination algorithm [10]. The hyperparameters of all baseline algorithms were tuned based on validation sets.

Adapting CoConut [9]. We minimize the energy

$$\mathcal{O}'(\mathbf{v}) = \|\mathbf{v} - \mathbf{f}^I\|^2 + \frac{\lambda^C}{k^C}\mathbf{v}^\top L\mathbf{v}, \tag{13}$$

where L is the graph Laplacian calculated based on local k-nearest neighbors (with $k = k^C$) in the test time feature space, and λ^C and k^C are hyperparameters. The first term in $\mathcal{O}'$ ensures that the final solution does not deviate significantly from $\mathbf{f}^I$ while the second term contributes to improving the final solution by enforcing its spatial smoothness measured via the Laplacian L. The accompanying supplemental material provides the details of this adaptation and the construction of the Laplacian L.

Adapting Kim et al. [10]. This method forms predictive *distributions* from reference tasks, then penalizes their pairwise KL-divergence from the target distribution. To adapt their method to our setting, if we let their reference task predictions be new features $\{H^i\}$, then this approach works when the (probability) space of each feature coincides with the space of predictions $\mathcal{Y}$. This makes their algorithm applicable only to datasets where one-dimensional test time features are provided as the predictions made for potentially-related tasks (e.g., the *PubFig* and *Shoes* datasets discussed shortly). We demonstrate that our general multi-dimensional test time feature algorithm is a strong alternative to Kim et al.'s approach even in this special setting.

4.1. Results

MFeat. This contains 6 different feature sets (F1–F6) of 2,000 handwritten digits, with rank outputs obtained from digit class labels. We use each feature set as the baseline features $\mathbf{g}$, with the remaining features used as test time features $\{H^i\}_{i=1}^5$, creating 6 different experimental settings.

Table 1. *MFeat* dataset. Ranking algorithm mean accuracy percent, plus standard deviation in parenthesis, given the F1–F6 features.
$\mathbf{f}^I$: The initial predictions.
$\mathbf{f}^O$: TUPI with other F-feature sets as test time information.
$\mathbf{f}^R$: $\mathbf{f}^O$ with an additional 10 random features.
$\mathbf{f}^{S_1,S_3}$: $\mathbf{f}^O$ with only randomly selected 1 and 3 F-feature sets.
$\mathbf{f}^{G_1-G_3}$: $\mathbf{f}^O$ with ground-truth target variables as test time features (of decreasing noise standard deviations $\{1, 0.2, 0\}$).

	$\mathbf{f}^I$	$\mathbf{f}^O$	$\mathbf{f}^R$	$\mathbf{f}^{S_1}$	$\mathbf{f}^{S_3}$	$\mathbf{f}^{G_1}$	$\mathbf{f}^{G_2}$	$\mathbf{f}^{G_3}$
F1	77.85 (2.26)	81.97 (2.95)	82.14 (2.86)	79.88 (1.49)	82.15 (2.37)	78.84 (2.35)	87.84 (0.84)	99.56 (0.34)
F2	79.28 (1.23)	81.45 (1.69)	81.61 (1.60)	80.40 (1.37)	81.37 (1.54)	80.00 (1.22)	88.16 (0.28)	99.61 (0.16)
F3	75.70 (2.38)	78.31 (3.13)	78.25 (3.16)	77.12 (1.79)	78.32 (3.02)	76.77 (2.20)	87.13 (0.82)	99.35 (0.42)
F4	70.88 (1.23)	74.33 (5.00)	74.36 (5.03)	70.80 (1.44)	74.29 (5.02)	72.00 (1.38)	86.22 (0.48)	99.53 (0.48)
F5	76.05 (2.66)	78.19 (3.29)	78.05 (3.26)	77.62 (3.28)	78.06 (3.72)	77.22 (2.62)	87.35 (0.72)	99.30 (0.44)
F6	77.10 (1.60)	82.25 (2.09)	82.44 (1.90)	79.30 (2.39)	80.71 (2.58)	78.07 (1.37)	86.93 (0.70)	99.44 (0.28)

Our approach consistently improves performance ($\mathbf{f}^O$ in the second column of Table 1). TUPI utility is demonstrated by results for F1, F2, and F6, with higher accuracy than the highest individual feature (F2). Further, we verify our algorithm's ability to *only pick useful features* by adding 10 additional random features to the test time information ($\{H^i\}_{i=1}^5$), with dimensions varying from 2 to 20. The results ($\mathbf{f}^R$) are very similar to that of $\mathbf{f}^O$ without the spurious random features. In addition, we measured the sensitivity of our algorithm against the number of test time feature sets by randomly selecting only one and three feature sets out of five. The results (Table 1: $\mathbf{f}^{S_1,S_3}$) indicate that the performance of our algorithm gracefully degrades.

Lastly, we verify the correct operation of TUPI in the ideal case by using ground-truth target ranks as test time information (Table 1: $\mathbf{f}^{G_1-G_3}$): The target rank variables are globally scaled to $[0, 1]$ and contaminated with zero-mean Gaussian noise with standard deviations of $\{1, 0.2, 0\}$. This leads to average peak signal-to-noise ratios of $\{-0.13, 14.08, \infty\}$-dB, respectively. When the noise-level is zero, our algorithm was able to fully exploit the test time information and achieve almost perfect rankings. Performance degrades gracefully as the noise-level increases.

***PubFig*, *Shoes*, and *OSR*.** These contain 772 images of 8 classes with 11 attributes [16], 14,658 images of 10 categories with 10 attributes [11], and 2,688 images of 6 attributes from 8 categories [11], respectively. The goal is to estimate rankings for each target attribute. The labels are provided as category-wise comparisons, i.e., each category has a stronger or weaker presence of certain attributes than other categories. For *PubFig* and *Shoes*, we construct the initial

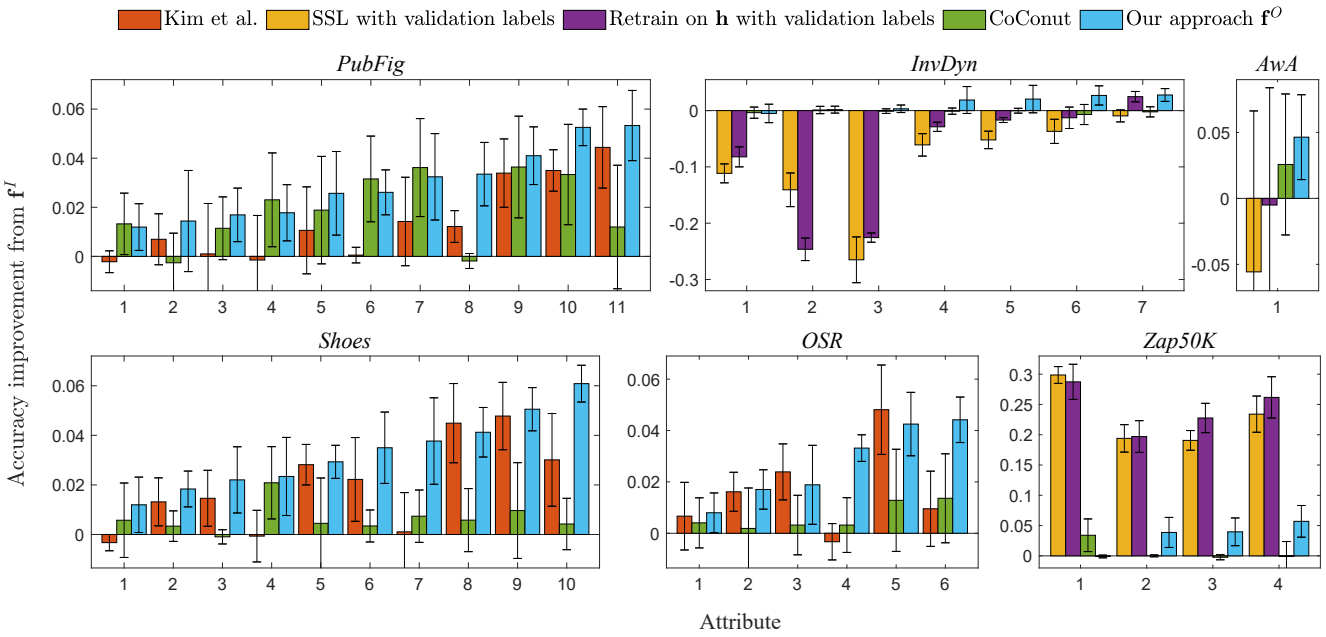

Figure 1. Accuracy improvements over $\mathbf{f}^I$ for six datasets. Higher is better; error bars are one standard deviation above and below the mean. Kim et al. cannot apply to *AwA*, *Zap50K*, or *InvDyn*. Attributes are sorted in ascending order of $\mathbf{f}^O$ accuracy values for improved readability. We include absolute accuracy values in our supplemental material.

rankings $\mathbf{f}^I$ using the GIST features and color histograms provided by Parikh and Grauman [16]. Similarly, for *OSR*, $\mathbf{f}^I$ was constructed using GIST features provided by the authors of [11]. We train a rank predictor from each target attribute. Then, for each attribute rank predictor, we use all other attribute rank predictions as test time information.

This setting has been explored by Kim et al. [10] as each test time feature set is only one-dimensional, though this method is not applicable for general test time features. For *PubFig*, Kim et al.'s predictor combination algorithm largely improves performance over the baseline $\mathbf{f}^I$ (Fig. 1), with our algorithm ($\mathbf{f}^O$) making further improvements to attributes 7–11. CoConut also improves the performance from $\mathbf{f}^I$ and it generates the best results in attributes 1, 4, 6, and 7; in this respect CoConut and our algorithm are complementary. In the supplemental material, we demonstrate that by combining these two approaches, we can obtain even better combination algorithms. For *Shoes*, Kim et al.'s algorithm provides similar performance, with our algorithm further improving attributes 3, 4, 6, 7, and 10. Our algorithm constantly outperformed CoConut. For *OSR*, our algorithm and Kim et al.'s algorithm are complementary: For attributes 3 and 5, Kim et al.'s algorithm is the best; for the remaining features, our algorithm further significantly improves the performance. CoConut achieves moderate accuracy gain. The accompanying supplemental provides tests of statistical significance of the results of all algorithms.

***InvDyn*.** This contains 45,000 data points collected from robot movement tasks. Each point is a 21-dimensional feature vector containing 7 joint positions, velocities, and accelerations [22]. The goal is to estimate the 7 torque variables constituting 7-different ranking problems. For each target output, only a subset of 15 input measurements are provided as the baseline features $\mathbf{g}$, with the remaining measurements provided as test time features $\{H^i\}$. Using only the test time features $\mathbf{h}$ and SSL severely degrades the ranking performance overall. This indicates that the test time information is complementary to the baseline features, which our TUPI approach exploits to increase accuracy.

In our automatic evaluation, we use 50 data points to select the hyperparameters of our algorithm. Thus, for comparison, we also show the results of training a new ranker and the graph Laplacian-based semi-supervised learning (SSL) ranker on 50 data points (denoted as $\mathbf{h}$ and SSL in Fig. 1). Even with only 50 labeled data points, these rankers can show higher accuracies than the baseline $\mathbf{f}^I$, especially for attribute 6; however, they are considerably worse than $\mathbf{f}^I$ for attribute 1. It is rare for our approach to degrade performance, which helps demonstrate that our algorithm can exploit new features appropriately when a large label set is not available. CoConut did not show any noticeable improvement over $\mathbf{f}^I$.

Animals with Attributes dataset (*AwA*). This contains 30,475 images of 50 animals class. Our goal is to rank images according to class labels. We use the features extracted by a pre-trained DeCAF network [3] as the baseline features G, and adopt SURF, PHOG, and VGG19 as the test time features $\{H^i\}_{i=1}^3$. These are provided by Lampert et al. [12].

Again, CoConut improved upon the baseline rankers $\mathbf{f}^I$

while our approach further significantly improved performance. We observe that powerful VGG19 features provide more distinctive descriptions of features and lead to improvements: performance of TUPI using only VGG19 test time features is almost the same as using three test time features. Throughout the denoising process, our algorithm successfully *selected* these VGG19 features.

Limitation—*Zap50K* dataset. When the test time information is significantly more powerful than the baseline features, retraining with even a small number of new labels can provide better results. This is demonstrated with the *Zap50K* dataset, which contains 50,025 images of shoes with 4 attributes. Attribute labels are collected by instance-level pairwise comparison via *Mechanical Turk* [24].

We use the 30-dimensional color histogram features and 960-dimensional GIST features as G and H, respectively, as provided by Yu and Grauman [24]. Across attributes, we use training and validation sets of around 300–400 pairs, and use test sets of around 300 pairs. $\mathbf{f}^I$ and $\mathbf{h}$ are trained on G and H with training and validation labels of the same size, respectively. In this case, GIST features H lead to much higher accuracy than color histogram features G. Even after our algorithm's improvements, a large performance gap remains—a potential upper limit of what our algorithm can achieved in TUPI on this dataset.

5. Conclusion

We have considered the problem of improving a predictor by exploiting additional features that are not available during predictor training. Our thesis is that these features can provide additional prediction information if they exhibit strong statistical dependence to the underlying task predictor. This might not be true in general, e.g., there is trivially no gain if the additional features are identical to the initial predictions (our supplemental material provides a toy example of such simple failure cases). However, to test this in practice, we introduce a new algorithm that estimates and strengthens statistical dependence. Over seven real-world relative attribute ranking experiments, on average, our algorithm often improves performance over the baseline predictors (43/45 attributes) and, more importantly, only very rarely degrades performance (2/45 attributes). This provide evidence that our thesis holds in practical applications, even when the feature adaptation scenario allows us no assumptions on the predictor or feature extractor forms, or any known existing statistical dependence.

Our experiments focused on two application scenarios using standard databases: 1) when predictors are trained on classical features and later tested with more powerful features; 2) when predictors are trained on a single feature but applied to multiple complementary features that are not necessary stronger than the original feature. Future work should assess TUPI in more real-world application scenarios arising in this context. Additional examples include:

1. When training requires specialized hardware unavailable to researchers or practitioners, e.g., an FPGA or massive 'tech giant'-scale cloud resources [8].

2. When training data are 'transient' and deleted after predictor creation, e.g., due to large storage requirements from scientific instruments like particle accelerators (CERN's petabyte-scale data [1]), but where new data are later available from new experiments to potentially exploit. A contrary scenario related to Vapnik and Vashist's framework [21] would be when privileged data existed only when training the original predictor.

3. Issues due to privacy concerns and data protection laws. For example, GDPR Right to be Forgotten/Erasure, where predictors trained on deleted data may still be kept, but where users re-using a service would provide new test time features (such as item ratings/recommendations).

Future work. Providing performance bounds to supporting our empirical findings would require future work to develop new theoretical analysis techniques. The challenges are that 1) our algorithm builds upon statistical dependence via HSIC, rather than via a common probability distribution distance (e.g., KL-divergence). Even if we assume that the test time features contain ground-truth labels, the analysis of convergence towards the ground truth is not straightforward as most existing techniques are developed based on probability distribution distances (e.g., PAC Bayesian bounds); 2) we use f^I as a surrogate to f^* when estimating the statistical dependence between f^* and $\{H^i\}_{i=1}^m$ and therefore, the deviations between f^I and f^* need to be quantified.

Acknowledgments

This work was supported by the National Research Foundation of Korea (NRF) grant (No. 2021R1A2C2012195), Institute of Information and communications Technology Planning and evaluation (IITP) grant (2021–0–00537, Visual Common Sense Through Self-supervised Learning for Restoration of Invisible Parts in Images), and IITP grant (2020–0–01336, Artificial Intelligence Graduate School Program, UNIST), funded by the Korea government (MSIT). This material is based on research sponsored by Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory (AFRL) under agreement number FA8750-19-2-1006. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory (AFRL) or the U.S. Government.

References

[1] K. Anthony. Are you up for the TrackML challenge?, 2018. https://home.cern/about/updates/2018/05/are-you-trackml-challenge. 8

[2] O. Chapelle and S. S. Keerthi. Efficient algorithms for ranking with SVMs. *Information Retrieval*, 13(3):201–215, 2010. 5

[3] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. DeCAF: a deep convolutional activation feature for generic visual recognition. In *ICML*, pages 647–655, 2014. 7

[4] T. Evgeniou and M. Pontil. Regularized multi–task learning. In *KDD*, pages 109–117, 2004. 2

[5] D. Gong, F. Sha, and G. Medioni. Locally linear denoising on image manifolds. *JMLR*, 16:265–272, 2010. 3

[6] A. Gretton, O. Bousquet, A. Smola, and B. Schölkopf. Measuring statistical dependence with Hilbert-Schmidt norms. In *ALT*, pages 63–77, 2005. 2, 3

[7] M. Hein and M. Maier. Manifold denoising. In *NIPS*, pages 561–568, 2007. 2, 3, 5

[8] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. In *NIPS Deep Learning Workshop*, 2014. 8

[9] S. Khamis and C. Lampert. CoConut: Co-classification with output space regularization. In *BMVC*, pages 1–11, 2014. 2, 6

[10] K. I. Kim, J. Tompkin, and C. Richardt. Predictor combination at test time. In *ICCV*, pages 3553–3561, 2017. 2, 3, 6, 7

[11] A. Kovashka, D. Parikh, and K. Grauman. Whittlesearch: Image search with relative attribute feedback. In *CVPR*, pages 2973–2980, 2012. 6, 7

[12] C. H. Lampert, H. Nickisch, and S. Harmeling. Learning to detect unseen object classes by between-class attribute transfer. In *CVPR*, pages 951–958, 2009. 7

[13] M. Lapin, M. Hein, and B. Schiele. Learning using privileged information: SVM+ and weighted SVM. *Neural Networks*, 53:95–108, 2014. 1

[14] G. Lee, E. Yang, and S. J. Hwang. Asymmetric multi-task learning based on task relatedness and loss. In *ICML*, pages 230–238, 2016. 2

[15] K. Muandet, K. Fukumizu, B. Sriperumbudur, and B. Schölkopf. Kernel mean embedding of distributions: a review and beyond. *Foundations and Trends in Machine Learning*, 10(1–2):1–141, 2017. 2, 4

[16] D. Parikh and K. Grauman. Relative attributes. In *ICCV*, pages 503–510, 2011. 2, 5, 6, 7

[17] Z. Shi and T.-K. Kim. Learning and refining of privileged information-based RNNs for action recognition from depth sequences. In *CVPR*, pages 3461–3470, 2017. 1

[18] L. Song, A. Smola, A. Gretton, J. Bedo, and K. Borgwardt. Feature selection via dependence maximization. *JMLR*, 13:1393–1434, 2012. 1, 4

[19] L. Song, A. Smola, A. Gretton, and K. M. Borgwardt. A dependence maximization view of clustering. In *ICML*, pages 815–822, 2007. 4

[20] I. Steinwart. On the influence of the kernel on the consistency of support vector machines. *JMLR*, 2:67–93, 2001. 2

[21] V. Vapnik and R. Izmailov. Learning using privileged information: similarity control and knowledge transfer. *JMLR*, 16:2023–2049, 2015. 1, 8

[22] S. Vijayakumar and S. Schaal. Locally weighted projection regression: An $O(n)$ algorithm for incremental real time learning in high dimensional space. In *ICML*, pages 1079–1086, 2000. 7

[23] B. Wang and Z. Tu. Sparse subspace denoising for image manifolds. In *CVPR*, pages 468–475, 2013. 3

[24] A. Yu and K. Grauman. Fine-grained visual comparisons with local learning. In *CVPR*, pages 192–199, 2014. 8

[25] D. Zhou, J. Weston, A. Gretton, O. Bousquet, and B. Schölkopf. Ranking on data manifolds. In *NIPS*, pages 169–176, 2004. 2, 6