Overlapping Domain Cover for Scalable and Accurate Regression Kernel Machines

Mohamed Elhoseiny
m.elhoseiny@cs.rutgers.edu
Ahmed Elgammal
elgammal@cs.rutgers.edu

Computer Science Department
Rutgers University
New Jersey, USA

Abstract

In this paper, we present the Overlapping Domain Cover (ODC) notion for kernel machines, as a set of overlapping subsets of the data that covers the entire training set and optimized to be spatially cohesive as possible. We propose an efficient ODC framework, which is applicable to various regression models and in particular reduces the complexity of Twin Gaussian Processes (TGP) regression from cubic to quadratic. We also theoretically justified the idea behind our method. We validated and analyzed our method on three human pose estimation datasets and interesting findings are discussed.

1 Introduction

Estimation of a continuous real-valued or a structured-output function from input features is one of the critical problems that appears in many machine learning applications. Recent advances in structure regression encouraged researchers to adopt it for formulating various problems with high-dimensional output spaces, such as segmentation, detection, and image reconstruction, as regression problems. However, the computational complexity of the state-of-the-art regression algorithms limits their applicability for big data. In particular, kernel-based regression algorithms such as Ridge Regression [12], Gaussian Process Regression (GPR) [17], and the Twin Gaussian Processes (TGP) [2] require inversion of kernel matrices ($O(N^3)$, where N is the number of the training points), which limits their applicability for big data. We refer to these non-scalable versions of GPR and TGP as full-GPR and full-TGP, respectively.

Khandekar et. al. [13] discussed properties and benefits of overlapping clusters for minimizing the conductance from spectral perspective. These properties of overlapping clusters also motivate studying scalable local prediction based on overlapping kernel machines; see figure 1. *In summary, the main question, we address in this paper, is how local kernel machines with overlapping training data could help speedup the computations and gain accurate predictions. We achieved considerable speedup and good performance on GPR, TGP, and IWTGP (Importance*

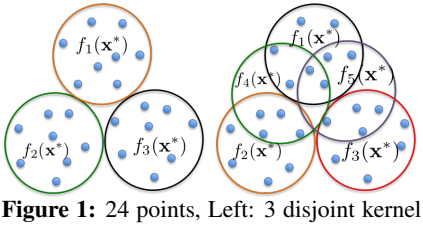


Figure 1: 24 points, Left: 3 disjoint kernel machines of 8 points, Right: 5 Overlapping kernel machines of 8 points. $f_i(\mathbf{x}^*)$ is the i^{th} kernel machine prediction for $\mathbf{x}^*$ test point.

Pages 94.1-94.12
DOI: https://dx.doi.org/10.5244/C.29.94

Weighted TGP) applied to 3D pose estimation datasets. To the best of our knowledge, our framework is the first to achieve quadratic prediction complexity for TGP. The ODC concept is also novel in the context of kernel machines and is shown here to be successfully applicable to multiple kernel-machines. We focus here on GPR and TGP due to space, IWTGP case (a third model) is attached in the Supplementary Materials (SM). The remainder of this paper is organized as follows: Section 2 and 3 presents some motivating kernel machines and the related work. Section 4 presents our approach and a theoretical justification for our ODC concept. Section 5 and 6 presents our experimental validation and conclusion.

2 Background on Full GPR and TGP Models

In this section, we show example kernel machines that motivated us to propose the ODC framework to improve their performance and scalability. Specifically, we review GPR for single output regression, and TGP for structured output regression. We selected GPR and TGP kernel machines for their increasing interest and impact. However, our framework is not restricted to them.

GPR [17] assumes a linear model in the kernel space with Gaussian noise in a single-valued output, i.e., $y = f(\mathbf{x}) + \mathcal{N}(0, \sigma_n^2)$, where $\mathbf{x} \in \mathbb{R}^{d_X}$ and $y \in \mathbb{R}$. Given a training set $\{\mathbf{x}_i, y_i, i = 1:N\}$, the posterior distribution of y given a test point $\mathbf{x}_*$ is:

$$p(y|\mathbf{x}_*) = \mathcal{N}(\mu_y = \mathbf{k}(\mathbf{x}_*)^\top (\mathbf{K} + \sigma_n^2 \mathbf{I})^{-1} \mathbf{f}, \sigma_y^2 = k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}(\mathbf{x}_*)^\top (\mathbf{K} + \sigma_n^2 \mathbf{I})^{-1} \mathbf{k}(\mathbf{x}_*)) \quad (1)$$

where $k(\mathbf{x}, \mathbf{x}')$ is kernel defined in the input space, $\mathbf{K}$ is an $N \times N$ matrix, such that $\mathbf{K}(l,m) = k(\mathbf{x}_l, \mathbf{x}_m)$, $\mathbf{k}(\mathbf{x}_*) = [k(\mathbf{x}_*, \mathbf{x}_1), ,..., k(\mathbf{x}_*, \mathbf{x}_N)]^\top$, $\mathbf{I}$ is an identity matrix of size N, σ_n is the variance of the measurement noise, $\mathbf{f} = [y_1, \cdots, y_N]^\top$. GPR could predict structured output $\mathbf{y} \in \mathbb{R}^{d_Y}$ by training a GPR model for each dimension. However, this indicates that GPR does not capture dependency between output dimensions which limit its performance.

TGP [2] encodes the relation between both inputs and outputs using GP priors. This was achieved by minimizing the Kullback-Leibler divergence between the marginal GP of outputs (e.g., poses) and observations (e.g., features). Hence, TGP prediction is given by:

$$\begin{aligned} \hat{\mathbf{y}}(\mathbf{x}_*) = & \underset{\mathbf{y}}{\operatorname{argmin}} [k_Y(\mathbf{y}, \mathbf{y}) - 2\mathbf{k}_Y(\mathbf{y})^\top (\mathbf{K}_X + \lambda_X \mathbf{I})^{-1} \mathbf{k}_X(\mathbf{x}_*) \\ & - \eta log(k_Y(\mathbf{y}, \mathbf{y}) - \mathbf{k}_Y(y)^\top (\mathbf{K}_Y + \lambda_Y \mathbf{I})^{-1} \mathbf{k}_Y(\mathbf{y}))] \end{aligned} \quad (2)$$

where $\eta = k_X(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}_X(\mathbf{x}_*)^\top (\mathbf{K}_X + \lambda_X \mathbf{I})^{-1} \mathbf{k}_X(\mathbf{x}_*)$, $k_X(\mathbf{x}, \mathbf{x}') = exp(\frac{-\|\mathbf{x}-\mathbf{x}'\|}{2\rho_x^2})$and $k_Y(\mathbf{y}, \mathbf{y}') = exp(\frac{-\|\mathbf{y}-\mathbf{y}'\|}{2\rho_y^2})$ are Gaussian kernel functions for input feature $\mathbf{x}$ and output vector $\mathbf{y}$, ρ_x and ρ_y are the kernel bandwidths for the input and the output . $\mathbf{k}_Y(\mathbf{y}) = [k_Y(\mathbf{y}, \mathbf{y}_1), ..., k_Y(\mathbf{y}, \mathbf{y}_N)]^\top$, where N is the number of the training examples. $\mathbf{k}_X(\mathbf{x}_*) = [k_X(\mathbf{x}, \mathbf{x}_1), , ..., k_X(\mathbf{x}, \mathbf{x}_N)]^\top$, and λ_X and λ_Y are regularization parameters to avoid overfitting. This optimization problem can be solved using a quasi-Newton optimizer with cubic polynomial line search [2]; we denote the number of steps to convergence as l_2. Table 1 shows the training an testing complexity of full GPR and TGP models, where d_Y is the dimensionality of the output. Table 1 also summarizes the computational complexity of the related approximation methods, discussed in the following section, and our method.

Table 1: Comparison of computational Complexity of training and testing for each of Full, NN (Nearest Neighbor), FITC, Local-RPC, and our ODC. Training is the time include all computations that does not depend on test data, which includes clustering in some of these methods. Testing includes computations only needed for prediction

	Training for GPR and TGP			Testing for each point		
	Ekmeans Clustering	RPC Clustering	Model training	GPR-Y	GPR-Var	TGP-Y
Full	-	-	$O(N^3+N^2 d_X)$	$O(N\cdot(d_X+d_Y)$	$O(N^2\cdot d_Y)$	$O(l_2\cdot N^2\cdot d_Y)$
NN [8]	-	-	-	$O(M^3\cdot d_Y)$	$O(M^3\cdot d_Y)$	$O(M^3+l_2\cdot M^2\cdot d_Y)$
FIC (GPR only, $d_Y=1$ [25])	-	-	$O(M^2\cdot(N+d_X))$	$O(M\cdot d_X)$	$O(M^2)$	-
Local-RPC (only GPR, $d_Y=1$ [8])	-	$N\cdot log(\frac{N}{M})$	$O(M^2\cdot(N+d_X))$	$O(M\cdot d_X)$	$O(M^2)$	-
ODC (our framework)	$O(N\cdot\frac{N}{(1-p)M}\cdot d_X\cdot l_1)$	$O(N\cdot log(\frac{N}{(1-p)M})\cdot d_X)$	$O(M^2\cdot(\frac{N}{1-p}+d_X))$	$O(K'\cdot M\cdot(d_X+d_Y))$	$O(K'\cdot M^2\cdot d_Y)$	$O(l_2\cdot K'\cdot M^2\cdot d_Y)$

3 Related Work on Approximation Methods

Various approximation approaches have been presented to reduce the computational complexity in the context of GPR. As detailed in [15], approximation methods on Gaussian Processes may be categorized into three trends: matrix approximation, likelihood approximation, and localized regression. The matrix approximation trend is inspired by the observation that the kernel matrix inversion is the major part of the expensive computation, and thus, approximating the matrix by a lower rank version, $M \ll N$ (e.g., Nyström Method [24]). While this approach reduces the computational complexity from $O(N^3)$ to $O(NM^2)$ for training, there is no guarantee on the non-negativity of the predictive variance [17]. In the second trend, likelihood approximation is performed on testing and training examples, given M artificial examples known as inducing inputs, selected from the training set (e.g. Deterministic Training Conditional (DTC) [18], Full Independent conditional (FIC) [21], Partial Independent Conditional (PIC) [20]). The drawback of this trend is the dilemma of selecting M inducing points, which might be distant from the test point, resulting in a performance decay; see Table 1 for the complexity of FIC.

A third trend, localized regression, is based on the belief that distant observations are almost unrelated. The prediction of a test point is achieved through its M nearest points. One technique to implement this notion is through decomposing the training points into disjoint clusters during training, where prediction functions are learned for each of them [15]. At test time, the prediction function of the closest cluster is used to predict the corresponding output. While this method is efficient, it introduces discontinuity problems on boundaries of the subdomains. Another way to implement local regression is through Mixture of Experts (MoE) as an Ensemble method to make prediction based on computing the final output by combining outputs of local predictors called experts (see a study on MoE methods [26]). Examples include Bayesian committee machine (BCM [22]), local probabilistic regression (LPR [23]), mixture of Tree of Gaussian Processes (GPs) [9], and Mixture of GPs [17]. While these approaches overcome the discontinuity problem by the combination mechanism, they suffer from intensive complexity at test time, which limits its applicability in large-scale setting, e.g., Tree of GPs and Mixture of GPs, involve complicated integration, approximated by computationally expensive sampling or Monte Carlo simulation.

Park etal. [15] proposed a large-scale approach for GPR by domain decomposition on up to 2D grid on input, where a local regression function is inferred for each subdomain such that they are consistent on boundaries. This approach obviously lacks a solution to high-dimensional input data because the size of the grid increases exponentially with the dimensions, which limits its applicability. More recently, [8] proposed a Recursive Partitioning Scheme (RPC) to decompose the data into non-overlapping equal-size clusters, and they built a GPR on each cluster. They showed that this local scheme gives better performance than FIC [21] and other methods. However, this partitioning scheme obviously lacks

consistency on the boundaries of the partitions and it was restricted to single-output GPR. Table 1 shows the complexity of this scheme denoted by local-RPC for GPR.

Beyond GPR, we found that local regression was adopted differently in structured regression models like Twin Gaussian Processes (TGP) [2], and also an data bias version of it, denoted by IWTGP [25]. TGP and IWTGP outperform not only GPR in this task, but also various regression models including Hilbert Schmidt Independence Criterion (HSIC) [10], Kernel Target Alignment (KTA) [6], and Weighted-KNN [17]. Both TGP and IWTGP have no closed-form expression for prediction. Hence, the prediction is made by gradient descent on a function that needs to compute the inverse of both the input and output kernel matrices, $O(N^3)$ complexity. Practically, both approaches have been applied by finding the $M \ll N$ Nearest-Neighbors (NN) of each test point in [2] and [25]. The prediction of a test point is $O(M^3)$ due to the inversion of $M \times M$ input and output kernel Matrices. However, NN scheme has three drawbacks: (1) A regression model is computed for each test point, which results in a scalability problems in prediction (i.e., Matrix inversions on the NN of each each test point), (2) Number of neighbors might not be large enough to create an accurate prediction model since it is constrained by the first drawback, (3) It is inefficient compared with the other schemes used for GPR. Table 1 shows the complexity of this NN scheme.

4 ODC Framework

The problems of the existing approaches, presented above, motivated us to develop an approach that satisfies the properties listed in table 2. The table also shows which of these properties are satisfied for the relevant methods. In order to satisfy all the properties, we present the Overlapping Domain Cover (ODC) notion. We define the ODC as a collection of overlapping subsets of the training points, denoted by subdomains, such that they are as spatially coherent as possible. During training, an ODC is computed such that each subdomain overlaps with the neighboring subdomains. Then, a local prediction model (kernel machine) is created for each subdomain and the computations that does not depend on the test data are factored out and precomputed (e.g. inversion of matrices). The nature of the ODC generation makes these kernel machines consistent in the overlapped regions, which are the boundaries since we constraint the subdomains to be coherent. This is motivated by the notion that data lives on a manifold with local properties and consistent connections between its neighboring regions. On prediction, the output is calculated as a reduction function of the predictions on the closed subdomain(s). Table 1 (the last row) shows the complexity for our generalized ODC framework, detailed in Sec 4.1 and 4.2. In contrast to the prior work, our ODC framework is designed to cover structured regression setting, $d_Y > 1$ and to be applicable to GPR, TGP, and many other models.

Table 2: Contrast against most relevant methods

	[■]	FIC/PIC [■]	NN [■]	ODC
Accurate	No for high input dimension	No	Yes	Yes
Efficient	No	Yes	No	Yes
Scalable to arbitrary input dimension	No (2D)	Yes	Yes	Yes
Consistent on Boundaries	Yes	No	Yes	Yes
supported kernel machines	GPR	GPR	TGP	GPR, TGP, IWTGP and others
Easy to parallelize	No	No	Yes	Yes

Notations. Given a set of input data $X = \{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$, our prediction framework firstly generates a set of non-overlapping equal-size partitions, $C = \{C_1, \cdots, C_K\}$, such that $\cup_i C_i = X$, $|C_i| = N/K$. Then, the ODC is defined based on them as $\mathcal{D} = \{D_1, \cdots, D_K\}$, such that $|D_i| = M \forall i$, $D_i = C_i \cup O_i, \forall i$. O_i a the set of points that overlaps with the other partitions, i.e., $O_i = \{x : x \in \{\cup_{j \neq i} C_j\}\}$, such that $|O_i| = p \cdot M$, $|C_i| = (1-p) \cdot M$, $0 \leq p \leq 1$ is the ratio of points in each overlapping subdomain, D_i, that belongs to/overlaps with partitions, other

than its own, C_i.

It is important to note that, the ODC could be specified by two parameters, M and p, which are the number of points in each subdomain and the ratio of overlap respectively; this is since $K = N/(1-p)M$. This parameterization of ODC generation is reasonable for the following reasons. First, M defines the number of points that are used to train each local kernel machine, which controls the performance of the local prediction. Second, given M and that $K = N/(1-p)M$, p defines how coarse/fine the distribution of kernel machines are. It is not hard to see that as p goes to 0, the generated ODC reduces to the set of non-overlapping clusters. Similarly, as p approaches $1-1/M$, the ODC reduces to generating a cluster at each point with maximum overlap with other clusters, i.e., $K = N$, $|C_i| = 1$, and $|O_i| = M-1$. *Our main claim is two fold. First, precomputing local kernel machines (e.g. GPR, TGP, IWTGP) during training on the ODC significantly increase the speedup on prediction time. Second, given a fixed M and N, as p increases, local prediction performance increases, theoretically supported by Lemma 4.1*

Lemma 4.1. Under ODC notion, as the overlap p increases, the closer the nearest model to an arbitrary test point and the more likely that model get trained on a big neighborhood of the test point; see the proof in the Supplementary Materials (SM).

4.1 Training

There are several overlapping clustering methods that include (e.g. [16] and [3]), which looks relevant for our framework. However these methods does not fit our purpose both equal-size constraints for the local kernel machines. We also found them very slow in practice because their complexity varies from cubic to quadratic (with a big constant factor) on the training-set. These problems motivated us to propose a practical method that builds overlapping local kernel-machines with spatial and equal-size constraints. These constraints are critical for our purpose since the number of points in each kernel-machine determine its local performance. Hence, our training phase is two steps: (1) the training data is split into $K = N/(1-p)M$ equal-sized clusters of $(1-p)M$ points. (2) an ODC with K overlapping subdomains is generated by augmenting each cluster with $p \cdot M$ points from the neighboring clusters.

4.1.1 Equal-size Clustering

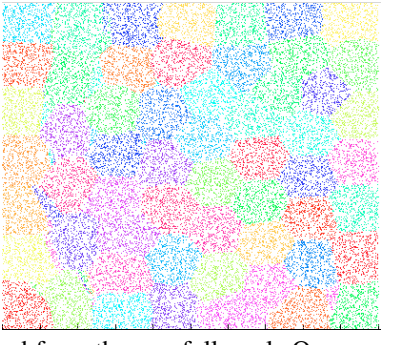
There are recent algorithms that deal with size constraints in clustering. For example, [27] formulated the problem of clustering with size constraints as a linear programming problem. However such algorithms are not computationally efficient, especially for large scale datasets (e.g., Human3.6M). We study two efficient ways to generate equal size clusters; see Table 1 (last row) for their ODC-complexity.

Recursive Projection Clustering (RPC) [5]. In this method, the training data is partitioned to perform GPR prediction. Initially all data points are put in one cluster. Then, two points are chosen randomly and orthogonal projection of all the data onto the line connecting them is computed. Depending on the median value of the projections, The data is then split into two equal size subsets. The same process is then applied to each cluster to generate 2^l clusters after l repetitions. The iterations stops once $2^l > K$. As indicated, the number of clusters in this method has to be a power of two and it might produce long thin clusters.

Equal-Size K-means (EKmeans). We propose a variant of k-means clustering [11] to generate equal-size clusters. The goal is to obtain disjoint partitioning of X into clusters $C = \{C_1, \cdots, C_K\}$, similar to the k-means objective, minimizing the within-cluster sum of squared Euclidean distances, $C = \arg_C J(C) = \min \sum_{j=1}^{K} \sum_{\mathbf{x_i} \in C_j} d(\mathbf{x_i}, \mu_j)$, where μ_i is the mean of cluster C_i, and $d(\cdot,\cdot)$ is the squared distance. Optimizing this objective is NP-hard and k-means iterates between the assignment and update steps as a heuristic to achieve a solution; l_1 denotes number of iterations of kmeans. We add equal-size constraints $\forall (1 \leq i \leq K), |C_i| = N/K = (1-p)M$.

In order to achieve this partitioning, we propose an efficient heuristic algorithm, denoted by *Assign and Balance (AB) EKmeans*. It mainly modifies the assignment step of the k-means to bound the size of the resulting clusters. We first assign the points to their closest see center as typically done in the assignment step of k-means. We use $C(\mathbf{x}_p)$ to denote the cluster assignment of a given point $\mathbf{x}_p$. This results in three types of clusters: balanced, overfull, and underfull clusters. Then some of the points in the overfull clusters are redistributed to the underfull clusters by assigning each of these points to the closest underfull cluster. This is achieved by initializing a pool of overfull points defined as $\tilde{X} = \{\mathbf{x}_p : \mathbf{x}_p \in C_l, |C_l| > N/K\}$; see Figure 2.

Figure 2: AB-EKmeans on 300,000 2D points, K= 57



Let us denote the set of underfull clusters by $\tilde{C} = \{C_p : |C_p| < N/K\}$. We compute the distances $d(\mathbf{x}_i, \mu_j), \forall \mathbf{x_i} \in \tilde{X}$ and $C_i \in \tilde{C}$. Iteratively, we pick the minimum distance pair $(\mathbf{x}_p, \mu_l)$ and assign $\mathbf{x}_p$ to cluster C_l instead of cluster $C(\mathbf{x}_p)$. The point is then removed from the overfull pool. Once an underfull cluster becomes full it is removed from the underfull pool, once an overfull cluster is balanced, the remaining points of that cluster are removed from overfull pool. The intuition behind this algorithms is that, the cost associated with the initial optimal assignment (given the computed means) is minimally increased by each swap since we pick the minimum distance pair in each iteration. Hence the cost is kept as low as possible while balancing the clusters.

4.1.2 Overlapping Domain Cover(ODC) Model

Having generated the disjoint equal size clusters, we generate the ODC subdomains based on the overlapping ratio p, such that $p \cdot M$ points are selected from the neighboring clusters. Let's assume that we select only the closest r clusters to each cluster, C_i is closer to C_j than C_k if $\|\mu_i - \mu_j\| < \|\mu_i - \mu_k\|$. It is important to note that r must be greater than $p/(1-p)$ in order to supply the required $p \cdot M$ points; this is since number of points in each cluster is $(1-p)M$. Hence, the minimum value for r is $\lceil (p \cdot M)/((1-p) \cdot M) \rceil = \lceil p/(1-p) \rceil$ clusters. Hence, we parametrize r as $r = \lceil t \cdot p/(1-p) \rceil, t \geq 1$. We study the effect of t in the experimental results section. Having computed r from p and t, each subdomain D_i is then created by merging the points in the cluster C_i with $p \cdot M$ points, retrieved from the r neighboring clusters. Specifically, the points are selected by sorting the points in each of r clusters by the distance to μ_i. The number of points retrieved for each of the r neighboring clusters is inversely proportional to the distance of its center to μ_i. If a subset of the r clusters are requested to retrieve more than its capacity (i.e., $(1-p)M$), the set of the extra points are requested from the remaining clusters giving priority to the closer clusters (i.e., starting

from the nearest neighboring cluster to the cluster on which the subdomain is created). As $t = 1$ and p increases, all points that belong to the r clusters tends to be merged with C_i. In our framework, we used FLANN [14] for fast NN-retrieval; see pseudo-code of ODC generation in SM.

After the ODC is generated, we compute the the sample normal distribution using the points that belong to each subdomain. Then, a local kernel machine is trained for each of the overlapping subdomains. We denote the point set normal distribution of the subdomains as $p(\mathbf{x}|D_i) = \mathcal{N}(\mu'_i \in R^{d_X}, \Sigma'_i \in R^{d_X \times d_X})$; ${\Sigma'}_i^{-1}$ is precomputed during the training for later use during the prediction. Finally, we factor out all the computations that does not depend on the test point (for GPR, TGP, IWTGP) and store them with each sub domain as its local kernel machine. We denote the training model for subdomain i as $\mathcal{M}^i$, which is computed as follows for GPR and TGP respectively; see SM for IWTGP.

GPR. Firstly, we precompute $(\mathbf{K}^i_j + \sigma^2_{n^i_j}\mathbf{I})^{-1}$, where $\mathbf{K}^i_j$ is an $M \times M$ kernel matrix, defined on the input points in D_i. Each dimension j in the output could have its own hyper-parameters, which results in a different kernel matrix for each dimension $\mathbf{K}^i_j$. We also precompute $(\mathbf{K}^i_j + \sigma^2_{n^i_j}\mathbf{I})^{-1}\mathbf{y}_j$ for each dimension. Hence $\mathcal{M}^i_{GPR} = \{(\mathbf{K}^i_j + \sigma^2_{n^i_j}\mathbf{I})^{-1}, (\mathbf{K}^i_j + \sigma^2_{n^i_j}\mathbf{I})^{-1}\mathbf{y}_j)\ , j = 1 : d_Y\}$.

TGP. The local kernel machine for each subdomain in TGP case is defined as $\mathcal{M}^i_{TGP} = \{(\mathbf{K}^i_X + \lambda^i_X\mathbf{I})^{-1}, (\mathbf{K}^i_Y + \lambda^i_Y\mathbf{I})^{-1}\}$, where $\mathbf{K}^i_X$ and $\mathbf{K}^i_Y$ are $M \times M$ kernel matrices defined on the input points and the corresponding output points respectively, which belong to domain i.

4.2 Prediction

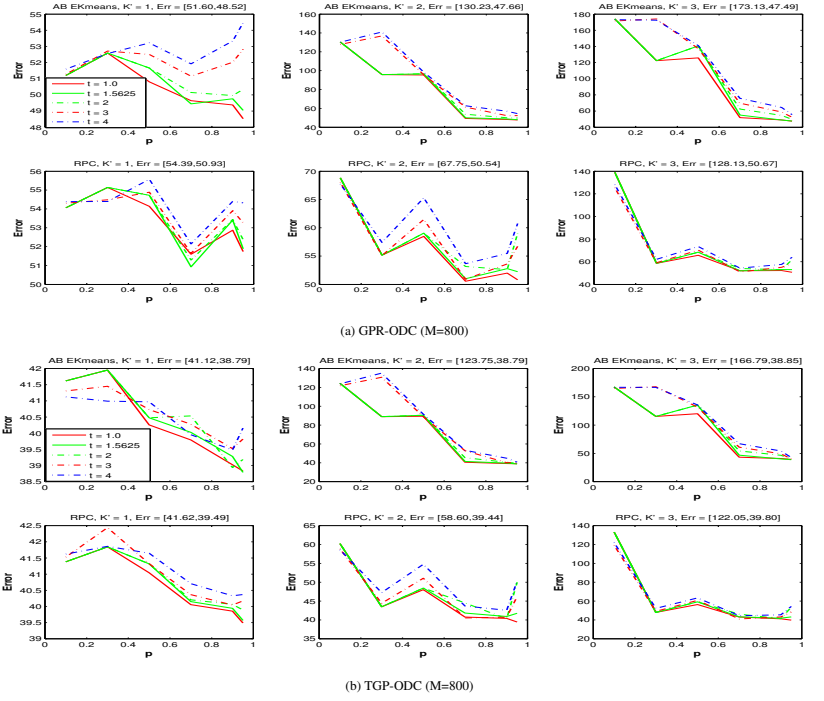
ODC-Prediction is performed in three steps.

(1) Finding the closest subdomains. The closest $K' \ll K$ subdomains are determined based on the covariance norm of the displacement of the test input from the means of the subdomain distribution (i.e. $\|\mathbf{x} - \mu'_i\|_{{\Sigma'_i}^{-1}}, i = 1 : K$, where $\|\mathbf{x} - \mu'_i\|_{{\Sigma'_i}^{-1}} = (\mathbf{x} - \mu'_i)^T {\Sigma'_i}^{-1}(\mathbf{x} - \mu'_i)$. The reason behind using the covariance norm is that it captures details of the density of the distribution in all dimensions. Hence, it better models $p(\mathbf{x}|D_i)$, indicating better prediction of $\mathbf{x}$ on D_i.

(2) Closest subdomains Prediction. Having determined the closest subdomains, predictions are made for each of the closest clusters. We denote these predictions as $\{Y^i_{x_*}\}_{i=1}^{K'}$. Each of these prediction are computed according to the selected kernel machine. For GPR, predictive mean and variance are $O(M \cdot d_X)$ and $O(M^2 \cdot d_Y)$ respectively, for each output dimension. For TGP, the prediction is $O(l_2 \cdot M^2 \cdot d_Y)$; see Eq 2.

(3) Subdomains weighting and Final prediction. The final predictions are formulated as $\mathbf{Y}(x_*) = \sum_{i=1}^{K'} a_i \mathbf{Y}^i_{x_*}, a_i > 0, \sum_{i=1}^{K'} a_i = 1$. $\{a_i\}_{i=1}^{K'}$ are computed as follows. Let the distribution of domain $\{D^i_{x_*} = \|x - \mu'_i\|_{{\Sigma'_k}^{-1}}\}_{i=1}^{K'}$ denotes to the distances to the closest subdomains, $\{L^i_{x_*} = 1/D^i_{x_*}\}_{i=1}^{K'}, a_i = L^i_{x_*} / \sum_{i=1}^{K'} L^i_{x_*}$.

It is not hard to see that when $K' = 1$, the prediction step reduces to regression using the closest subdomain to the test point. However it is reasonable in most of the prior work to make prediction using the closest model, we generalized it to K' closest kernel machines and combining their predictions, so as to study how consistency of the combined prediction behaves as the overlap increases (i.e., p); see the experiments.



(a) GPR-ODC (M=800)

(b) TGP-ODC (M=800)

Figure 3: ODC framework Parameter Analysis of GPR and TGP on Human Eva Dataset

5 Experimental Results

We evaluated our framework on three human pose estimation datasets, Poser, HumanEva, and Human3.6M. Poser dataset [1] consists of 1927 training and 418 test images. The image features, corresponding to bag-of-words representation with silhouette-based shape-context features. The error is measured by the root mean-square error (in degrees), averaged over all joints angles, and is given by: $Error(\hat{y}, y^*) = \frac{1}{54}\sum_{m=1}^{54} \|\hat{y}^m - y^{*m} mod\ 360^\circ\|$, where $\hat{y} \in R^{54}$ is an estimated pose vector, and $y^* \in R^{54}$ is a true pose vector. HumanEva datset [19] contains synchronized multi-view video and Mocap data of 3 subjects performing multiple activities. We use HOG features [2] ($\in R^{270}$) proposed in [2]. We use training and validations subsets of HumanEva-I and only utilize data from 3 color cameras with a total of 9630 image-pose frames for each camera. This is consistent with experiments in [2] and [25]. We use half of the data for training and half for testing. Human3.6M [9] is a dataset of millions of Human poses. We managed to evaluate our proposed ODC-framework on six Subjects (S1, S2, S6, S7, S8, S9) from it, which is ≈ 0.5 million poses. We split them into 67% training 33% is testing. HOG features are extracted for 4 image-views for each pose and concatenated into 3060-dim vector. Error for each pose, in both HEva (in *mm*) and Human 3.6 (in *cm*), is measured as $Error(\hat{y}, y^*) = \frac{1}{L}\sum_{m=1}^{L} \|\hat{y}^m - y^{*m}\|$.

There are four control parameters in our ODC framework: M, p, t, and K'. Figure 3 shows our parameter analysis with different values of p, t and K' on HumanEva dataset for GPR and TGP as local regression machines, where $M = 800$. Each sub-figure consists of six plots in two rows. The first row indicates the results using AB-Ekmeans clustering scheme, while the second row shows the results for RPC clustering scheme. Each row has three plots, one for $K' = 1$, 2, and 3 respectively. Each plot shows the error of different t against

Table 3: Error & Time on Poser and Human Eva datasets (Intel core-i7 2.6GHZ), M = 800

		Poser			HumanEva		
		Error (deg)	**Training Time**	**Prediction Time**	**Error (mm)**	**Training Time**	**Prediction Time**
TGP	**NN [8]**	5.43	-	188.99 sec	**38.1**	-	6364 sec
	ODC ($p = 0.9, t = 1, K' = 1$)**-Ekmeans**	**5.40**	(3.7 +25.1) sec	**16.5** sec	**38.9**	(2001 + 45.4) sec	**298** sec
	ODC ($p = 0, t = 1, K' = 1$)**-Ekmeans**	7.60	(3.9 + 1.33) sec	14.8 sec	41.87	(240 + 4.9) sec	257 sec
	ODC ($p = 0.9, t = 1, K' = 1$)**-RPC**	5.60	(0.23 +41.6) sec	15.8 sec	39.9	(0.45 + 49.1) sec	277 sec
	ODC ($p = 0, t = 1, K' = 1$)**-RPC**	7.70	(0.15 + 1.7) sec	13.89 sec	42.32	(0.19 + 5.2) sec	242 sec
GPR	**NN**	6.77	-	24 sec	54.8	-	618 sec
	ODC ($p = 0.9, t = 1, K' = 1$)**-Ekmeans**	**6.27**	(3.7 +11.1) sec	**0.56** sec	**49.3**	(2001 + 42.85)sec	**79** sec
	ODC($p = 0.0, t = 1, K' = 1$)**-Ekmeans**	7.54	(3.9 + 1.38 sec)	0.35 sec	49.6	(240 + 6.4) sec	48 sec
	ODC ($p = 0.9, t = 1, K' = 1$)**-RPC**	6.45	(0.23 +17.3) sec	0.52 sec	52.8	(0.49 + 46.06) sec	64 sec
	ODC ($p = 0.0, t = 1, K' = 1$)**-RPC = [8]**	7.46	(0.15 + 1.5) sec	0.27 sec	54.6	(0.26 + 4.6) sec	44 sec
	FIC [6]	7.63	(- + 20.63)	0.3106	68.36	-	102 sec

p from 0 to 0.95; i.e., it shows how the overlap affects the performance for different values of t. Each plot shows, on its top caption, the minimum and the maximum overlap regression errors where $t \to 1$. Looking at these plots, there are a number of observations:

(1) As $t \to 1$ (the solid red line), the behavior of the error tends to reduce as p increases, i.e., the overlap.

(2) Comparing different K', the behavior of the error indicates that combining multiple predictions (i.e., $K' = 2$ and $K' = 3$), gives poor performance, compared with $K' = 1$, when the overlap is small. However, all of them, $K' = 1$, 2, and 3, performs well as $p \to 1$; see column 2 and 3 in figure 3. This indicates consistent prediction of neighboring subdomains as p increases. The main reason behind this behavior is that as p increases, the local models of the neighboring subdomains normally share more training points on their boundaries, which is reflected as shared constraints during the training of these models making them more consistent on prediction.

(3) Comparing the first row to the second row in each subfigure, it is not hard to see that our AB-Ekmeans partitioning scheme consistently outperforms RPC [8], *e.g.* the error in cases of GPR (M=800) is 47.48mm for AB-EKmeans and 50.66mm for RPC, TGP (M=800) is 38.8mm for AB-EKmeans and 39.8mm for RPC. This problem is even more severe when using smaller M, *e.g.* the error in case of TGP (M=400) is 39.5mm for EKmeans and 47.5mm for RPC, (A detailed plot attached in SM for M=400).

(4) TGP gives better prediction than GPR (i.e., 38mm using TGP compared with 47mm using GPR).

(5) As M increases, the prediction error decreases. For instance, when $M = 200$, The error for TGP best performance increased to 43.88mm instead of 38.9mm for $M = 800$. We found these observation to be also consistent on Poser dataset.

This analysis helped us conclude recommending choosing t close to 1, big overlap (p closer to 1), and $K' = 1$ is sufficient for accurate prediction.

Having accomplished the performance analysis which comprehensively interprets our parameters, we used the recommended setting to compare the performance with other methods and show the benefits of this framework. Figure 4 shows the speedup gained by retrieving the matrix inverses on test time, compared with computing them at test time by NN scheme. The figure shows significant speedup from precomputing local kernel machines.

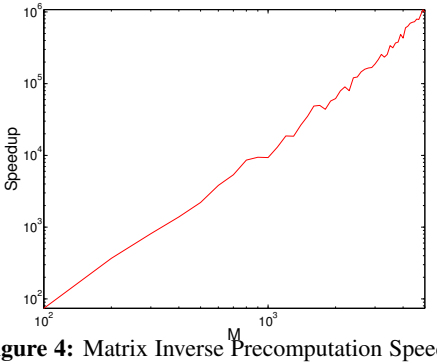


Figure 4: Matrix Inverse Precomputation Speedup of ODC framework prediction as M increases (log-log scale)

Table 3 shows error, training time and prediction time of NN, FIC, and different variations of ODC on Poser and Human-

Eva datasets. Training time is formatted as $(t_c + t_p)$, where t_c is the clustering time and t_p is the remaining training time excluding clustering. As indicated in the top part of table 3, TGP under our ODC-framwork can significantly speedup the prediction compared with NN-scheme in [2], while achieving competitive performance; better in case Poser Dataset. As illustrated in our analysis in Figure 3, higher overlap (p) gives better performance. From time analysis perspective, higher p costs more training time due that more subdomains are created and trained. While, Figure 3 and Table 3 indicates that AB-Ekmeans gives better performance than RPC under both GPR and TGP, AB-Ekmeans takes more time for clustering. Yet, it is feasible to compute in all the datasets, we used in our experiments. Our experiments also indicate that as $p \rightarrow 1$ in TGP and GPR, $K' = 2$ and $K' = 3$ takes double and triple the prediction time respectively, compared with $K' = 1$, with almost no error reduction. We also compared our model to FIC in case of GPR, and our model achieved smaller error and smaller prediction time; see bottom part in Table 3. However, TGP consistently gives better results on both Poser and HumanEva datasets. We also tried full TGP and GPR on Poser and Human Eva Datasets. Full TGP error is 5.35 for Poser and 40.3 for Human Eva. Full GPR error is 6.10 for Poser and 59.62 for Human Eva. The results indicate that ODC achieves either better or competitive to the full models.

Based on our comprehensive experiments on HumanEva and Poser datasets, we conducted an experiment on Human3.6M dataset with TGP kernel machine, where $M = 1390$, $t = 1$, $p = 0.6, K' = 1$, Ekmeans for clustering. We achieved a speedup of $41.7X$ on prediction time using our ODC framework compared with NN-scheme, i.e., 7 days if NN-scheme is used versus 4.03 hours in our case. The error is 13.5 (cm) for NN and 13.8 (cm) for ODC.

6 Conclusion

We proposed an efficient ODC framework for kernel machines and validated the framework on structured regression machines on three human pose estimation datasets. The key idea is to equally partition the data and create cohesive overlapping subdomains, where local kernel machines are computed for each of them. The framework is general and could be applied to various kernel machine beyond GPR, TGP, IWTGP validated in this work. Similar to TGP and IWTGP, our framework could be easily applied to the recently proposed Generalized TGP [8] which is based on Sharma Mittal divergence, a relative entropy measure brought from Physics community. We theoretically justified our framework's notion in Lemma 4.1.

Acknowledgment. This research was partially funded by NSF award # 1409683.

References

[1] A. Agarwal and B. Triggs. Recovering 3d human pose from monocular images. *PAMI*, 2006.

[2] Liefeng Bo and Cristian Sminchisescu. Twin gaussian processes for structured prediction. *Int. J. Comput. Vision*, 87(1-2):28–52, March 2010. ISSN 0920-5691.

[3] Francesco Bonchi, Aristides Gionis, and Antti Ukkonen. Overlapping correlation clustering. *Knowledge and information systems*, 35(1):1–32, 2013.

[4] Vlad Olaru Catalin Ionescu, Dragos Papava and Cristian Sminchisescu. Human3.6M: Large Scale Datasets and Predictive Methods for 3D Human Sensing in Natural En-

vironments. Technical report, nstitute of Mathematics at the Romanian Academy and University of Bonn, 01 2012.

[5] Krzysztof Chalupka, Christopher K. I. Williams, and Iain Murray. A framework for evaluating approximation methods for gaussian process regression. *JMLR*, 14(1), February 2013.

[6] J. Cristianini, N. Shawe-Taylor and J. S. Kandola. Spectral kernel methods for clustering. In *NIPS*, 2001.

[7] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *CVPR*, volume 1, pages 886 –893 vol. 1, june 2005. doi: 10.1109/CVPR.2005.177.

[8] Mohamed Elhoseiny and Ahmed Elgammal. Generalized twin gaussian processes using sharma-mittal divergence. *Machine Learning*, pages 1–26, 2015.

[9] Robert B. Gramacy and Herbert K. H. Lee. Bayesian treed gaussian process models with an application to computer modeling. *Journal of the American Statistical Association*, 2007.

[10] Arthur Gretton, Olivier Bousquet, Alex Smola, and Bernhard Schölkopf. Measuring statistical dependence with hilbert-schmidt norms. In *International conference on Algorithmic Learning Theory*, 2005.

[11] J. A. Hartigan and M. A. Wong. Algorithm as 136: A k-means clustering algorithm. *J. Roy. Statist. Soc. Ser. C)*, 1979.

[12] A. E. Hoerl and R. W. Kennard. Ridge Regression: Biased Estimation for Nonorthogonal Problems. *Technometrics*, 1970.

[13] Rohit Khandekar, Guy Kortsarz, and Vahab Mirrokni. On the advantage of overlapping clusters for minimizing conductance. *Algorithmica*, 69(4):844–863, 2014.

[14] M. Muja and D. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *VISAPP*, pages 331–340, 2009.

[15] Chiwoo Park, Jianhua Z. Huang, and Yu Ding. Domain decomposition approach for fast gaussian process regression of large spatial data sets. *Journal of Machine Learning Research*, 12:1697–1728, 2011.

[16] Airel Pérez-Suárez, José F Martínez-Trinidad, Jesús A Carrasco-Ochoa, and José E Medina-Pagola. Oclustr: A new graph-based algorithm for overlapping clustering. *Neurocomputing*, 121:234–247, 2013.

[17] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005. ISBN 026218253X.

[18] Matthias Seeger. Fast forward selection to speed up sparse gaussian process regression. In *in Workshop on AI and Statistics 9*, 2003.

[19] Leonid Sigal, Alexandru O. Balan, and Michael J. Black. Humaneva: Synchronized video and motion capture dataset and baseline algorithm for evaluation of articulated human motion.

[20] Edward Snelson. Local and global sparse gaussian process approximations. In *AISTATS*, 2007.

[21] Edward Snelson and Zoubin Ghahramani. Sparse gaussian processes using pseudo-inputs. In *NIPS*, 2006.

[22] Volker Tresp. A bayesian committee machine. *NEURAL COMPUTATION*, 12:2000, 2000.

[23] R. Urtasun and T. Darrell. Sparse probabilistic regression for activity-independent human pose inference. In *CVPR*, 2008.

[24] Christopher Williams and Matthias Seeger. Using the nystrÃűm method to speed up kernel machines. In *NIPS*, 2001.

[25] Makoto Yamada, Leonid Sigal, and Michalis Raptis. No bias left behind: covariate shift adaptation for discriminative 3d pose estimation. In *ECCV*, 2012.

[26] S.E. Yuksel, J.N. Wilson, and P.D. Gader. Twenty years of mixture of experts. *Neural Networks and Learning Systems, IEEE Transactions on*, 2012.

[27] Shunzhi Zhu, Dingding Wang, and Tao Li. Data clustering with size constraints. *Knowledge-Based Systems*, 23(8):883 – 889, 2010. ISSN 0950-7051.