`GURLS_mkl`: A PFBS-based Implementation for Multiple Kernel Learning

(Jeremiah) Zhe Liu
Department of Biostatistics
Harvard University
Boston, MA 02115
`zhl112@mail.harvard.edu`

Contents

1 Introduction

1.1 Multiple Kernel Learning Problem

Multiple kernel learning (MKL) (Bach et al. [2004]) is the process of finding an optimal kernel from a prescribed (convex) set $\mathcal{K}$ of basis kernels, for learning a real-valued function by regularization. In this setting, we consider a RKHS $\mathcal{H} = \mathcal{H}_1 \oplus \mathcal{H}_2 \cdots \oplus \mathcal{H}_M$ with reproducting kernel $\mathbf{k} \in \mathcal{K} = \{\sum_{i=1}^M c_i \mathbf{k}_i | (c_i \geq 0 \forall i) \wedge \sum_{i=1} c_i = 1\}$ such that $f = \sum_{i=1}^M f_i, f_i \in \mathcal{H}_i$. By Rosasco et al. [2009] , the problem of multiple kernel learning under square loss can be formulated as a elastic-net-regulated problem:

$$\arg\min_{f \in \mathcal{H}} \Big\{ \frac{1}{n} \sum_{i=1}^n (\sum_{j=1}^M f_j(x_i) - y_i)^2 + \mu \sum_{j=1}^M ||f_j||^2_{\mathcal{H}} + 2\tau \sum_{j=1}^M ||f_j||_{\mathcal{H}} \Big\} \tag{1}$$

1.2 Iterative PFBS Algorithm

By Theorem 1 of Rosasco et al. [2009], since the penalty function is lower semicontinuous, coercive, convex and one-homogenous, solution to problem 1 f^* is the unique fixed point of the the contractive mapping with step size σ:

$$\mathcal{T}_\sigma(f) = (\mathbf{I} - \pi_{\frac{\tau}{\sigma}K})\big(f - \frac{1}{2\sigma} \nabla_f [\frac{1}{n} ||f - y||^2]\big)$$

where $\pi_{\frac{\tau}{\sigma}K}(g)$ is a project operator which project g to $\mathcal{H}' = \{f \in \mathcal{H} \big| \; ||f_j||_{\mathcal{H}_j} \leq \frac{1}{\tau/\sigma} \forall j\}$.

Above mapping can also be written in terms of Kernel matrices by generalizing representer theorem and write $f^*_j(x) = \sum_{i=1}^n \alpha^T_{ji} k_j(x_i, x) = \boldsymbol{\alpha}^T_j \mathbf{k}_j(x)$, where $\boldsymbol{\alpha}_j$ and $\mathbf{k}_j(x)$ are $n \times 1$ vectors. Further, if denote:

$$\boldsymbol{\alpha}_{Mn \times 1} = (\boldsymbol{\alpha}_1, \dots, \boldsymbol{\alpha}_M)^T, \quad \mathbf{k}(x)_{Mn \times 1} = (\mathbf{k}_1(x)^T, \dots, \mathbf{k}_M(x)^T)^T$$

$$\mathbf{K}_{Mn \times Mn} = \begin{bmatrix} \mathbf{K}_1 & \dots & \mathbf{K}_M \\ \vdots & \ddots & \vdots \\ \mathbf{K}_1 & \dots & \mathbf{K}_M \end{bmatrix}, \text{where } \mathbf{K}_i = \mathbf{k}_i(.)\mathbf{k}_i(.)^T$$

$$\mathbf{y}_{Mn \times 1} = (y^T_{n \times 1}, \dots, y^T_{n \times 1})^T$$

The contraction mapping can be written as:

$$\mathcal{T}_\sigma(f) = (\mathbf{I} - \pi_{\frac{\tau}{\sigma}K})\big(\big[(1 - \frac{\mu}{\sigma})\boldsymbol{\alpha} - \frac{1}{\sigma n}(\mathbf{K}\boldsymbol{\alpha} - \mathbf{y})\big]^T \mathbf{k}\big) \qquad \text{where} \tag{2}$$

$$(1 - \frac{\mu}{\sigma})(\boldsymbol{\alpha})_j = \mathbf{S}_{\frac{\tau}{\sigma}}(K, \boldsymbol{\alpha})_j = \frac{\boldsymbol{\alpha}^T_j}{\sqrt{\boldsymbol{\alpha}^T_j \mathbf{K}_j \boldsymbol{\alpha}_j}} (\sqrt{\boldsymbol{\alpha}^T_j \mathbf{K}_j \boldsymbol{\alpha}_j} - \frac{\tau}{\sigma})_+$$

which is the soft-thresholding operator. We thus have below PFBS algorithm:

Algorithm 1 MKL PFBS algorithm

```
1: procedure rls_dual_mkl_pfbs($\mathbf{K}$, $\mathbf{y}$, $(\tau, \mu, \sigma)$)
2:     $\boldsymbol{\alpha}^0 = \mathbf{0}$
3:     for $p = 1$ to MAX_ITER do
4:         $\boldsymbol{\alpha}^p_0 = (1 - \frac{\mu}{\sigma})\boldsymbol{\alpha}^{p-1} - \frac{1}{\sigma n}(\mathbf{K}\boldsymbol{\alpha}^{p-1} - \mathbf{y})$
5:         $\boldsymbol{\alpha}^p = \mathbf{S}_{\frac{\tau}{\sigma}}(K, \boldsymbol{\alpha}^p_0)$
6:     end for
7:     return $f^{\texttt{MAX\_ITER}} = (\boldsymbol{\alpha}^{\texttt{MAX\_ITER}})^T \mathbf{k}$
8: end procedure
```

1.3 Implementation Detail

1.3.1 Block-wise Update

Notice that in (2), $\mathcal{T}_\sigma$ updates $\boldsymbol{\alpha}$ by group, it is thus possible to write $\mathcal{T}$ at p^{th} step as:

$$\mathcal{T}^p_\sigma = [\mathcal{T}^p_{\sigma,1}, \mathcal{T}^p_{\sigma,2}, \dots, \mathcal{T}^p_{\sigma,M}] \qquad \text{with} \qquad \mathcal{T}^p_{\sigma,j} = \mathbf{S}_{\frac{\tau}{\sigma}}\Big(K, \boldsymbol{\alpha}_0\Big)_j$$

$$\boldsymbol{\alpha}_0 = (1 - \frac{\mu}{\sigma})\boldsymbol{\alpha}^{p-1}_j - \frac{1}{n\sigma} * \boldsymbol{\epsilon}^{p-1} \qquad \text{where } \boldsymbol{\epsilon}^{p-1} = (\sum_{j=1}^M \mathbf{K}_j \boldsymbol{\alpha}^{p-1}_j - y)$$

by using above method we are able to avoid working directly with the $Mn \times Mn$ matrix $\mathbf{K}$ (as defined earlier in section 1.2), which led to reduced memory cost [1] and reduced difficulty in selecting stepsize and regularization parameters.

1.3.2 Choice of Stepsize

Based on Bach et al. [2004], it can be shown that a suitable choice of σ is $\sigma = \frac{1}{4}(a * L_{min} + b * L_{max}) + \mu$, where (b, a) denotes the lower/upper bound on the eigenvalue of $\mathbf{K}$, and (L_{min}, L_{max}) denotes the lower/upper bound of $\nabla^2 Q(\mathbf{f}, \mathbf{y})$. Specifically, in the context where Q is square loss (i.e. $\nabla^2_{\mathbf{f}} Q(\mathbf{f}, \mathbf{y}) = 2$), we have:

$$\sigma = \frac{1}{2}(a + b) + \mu$$

A naive choice of a would be the largest eigenvalue of $\mathbf{K}_{nM \times nM}$, which not only is computationally expensive but also leads to overly slow convergence. In pactice, if denote the maximum eigenvalue of each kernel matrix K_j to be a_j, it is found that setting a to be $\max\limits_{j \in \{1,..,M\}} (a_j)$ is suffice to guarantee convergence. This is because the mapping $\boldsymbol{\alpha}^{p-1} \mapsto \boldsymbol{\alpha}^p_0$ can be written as:

$$\boldsymbol{\alpha}_0 = (1 - \frac{\mu}{\sigma})\boldsymbol{\alpha}_j^{p-1} - \frac{1}{n} * \sum_{j=1}^{M} \frac{1}{\sigma}(\mathbf{K}_j \boldsymbol{\alpha}_j^{p-1} - \frac{y}{n})$$

As shown, in $\boldsymbol{\alpha}^{p-1} \mapsto \boldsymbol{\alpha}^p_0$ we actually updated $\boldsymbol{\alpha}^{p-1}$ M times, with step size $\frac{1}{\sigma}(\mathbf{K}_j \boldsymbol{\alpha}_j^{p-1} - \frac{y}{n})$ in each step. It is thus sufficient to find a a that properly scale the magnitude of all $||\mathbf{K}||$, leading natually to the choice $a = \max\limits_{j \in \{1,..,M\}} (a_j)$.

1.3.3 Role and range of μ and τ

In the context of MKL PFBS, μ are usually selected to be a small positive constant in order to guarantee the contraction property of $\mathcal{T}_\sigma$ is a contraction without compromising prediction accuracy. More specifically, since the Lipschitz constant for $\mathcal{T}_\sigma$ is $\mathcal{L}_\sigma \leq |1 - \frac{\mu}{\sigma}|$. We may follow [Mosci et al., 2010] and setting directly the default candidate values of $\frac{\mu}{\sigma}$ to be within the range of $[0, 0.1]$ during parameter selection.

The ratio $\frac{\tau}{\sigma}$ serves as a cutoff value for $|f_j^p| = \sqrt{\boldsymbol{\alpha}_j^T \mathbf{K}_j \boldsymbol{\alpha}_j}$ in the soft-thresolding operator $\mathbf{S}_{\frac{\tau}{\sigma}}$, with higher $\frac{\tau}{\sigma}$ leading to stronger spasity penalty on kernel effect estimates. In practice, we restrict the upperbound of the candidate values to be $\frac{1}{M} * ||y||/\sqrt{\max_j ||\mathbf{K}_j||}$, which approximate the average upper bound of $\sqrt{\boldsymbol{\alpha}_j^T \mathbf{K}_j \boldsymbol{\alpha}_j}$ since:

$$\begin{aligned} ||\boldsymbol{\alpha}_j^T \mathbf{K}_j \boldsymbol{\alpha}_j|| &\approx ||\boldsymbol{\alpha}_j^T (\frac{1}{M} \sum_j \mathbf{K}_j \boldsymbol{\alpha}_j)|| = \frac{1}{M} ||\boldsymbol{\alpha}_j^T y|| \\ &\leq \frac{1}{M} ||y|| ||\boldsymbol{\alpha}_j^T|| = \frac{1}{M} ||y|| ||\boldsymbol{\alpha}_j^T \mathbf{K}_j \mathbf{K}_j^*|| \\ &\leq \frac{1}{M} ||y||^2 / ||\mathbf{K}_j|| \end{aligned}$$

1.3.4 Continuation strategy on regularization path

A core component of the τ parameter selection is to compute the regularization path over $\tau_1 > \cdots > \tau_n$, which can be computed efficiently using *continuation strategy*, i.e. using $\alpha^{\tau_{k-1}}$ calculated for τ_{k-1} as the initial value for α^{τ_k}. Note that the corresponding solution can be usually computed in a fast way since it is very sparse, though possibly under-fitting the data [Rosasco et al., 2009].

2 Software Structure and Usage

2.1 Software Detail

`GURL_MKL` is developed to work seamlessly with the `gurls` function in GURLS package. A complete list of `GURLS_mkl` functions and the related option field `opt.mkl` can be found in Appendix. As shown in Figure 1, a typical `GURLS_mkl` pipeline is composed of 4 core functions for, respectively, generating multiple kernels (`kernel_mkl`), hold-out-based L_1/L_2 tunning parameter selection (`paramsel_homkl`), PFBS-based dual parameter estimation (`rls_dual_mkl`), and dual prediction (`pred_dual_mkl`). The configuration of the entire process is controlled by the option struct `opt.mkl`, whose important fields are:

[1] $O(Mn^2)$ instead of $O(M^2n^2)$

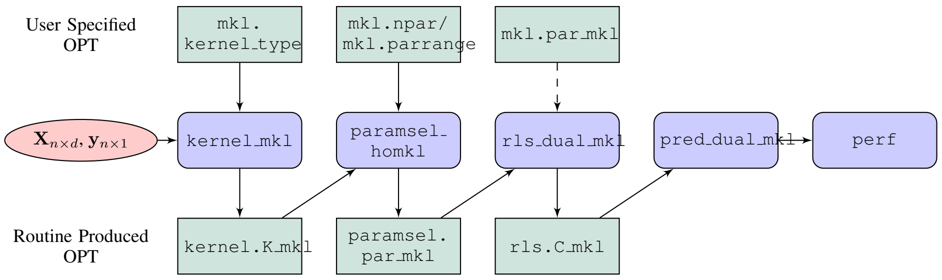


Figure 1: `GURLS_mkl` process

- For Kernel Generation:
  - `mkl.type`: A cell array specifying the type and parameters for the kernels to be used. For example, if we want to use 3 Gaussian kernels with $\sigma = 1, 2, 3$ and 1 Linear kernel, then `mkl.type` = $\{\{$`'kernel_rbf'`$, 1 : 3\}, \{$`'kernel_linear'`$, 0\}\}$. Kernel type must be the name of an existing GURLS kernel function. Currently only RBF and linear kernels are supported. User **must** set this field by hand.
- For tunning parameter selection:
  - `mkl.npar`: A 1×2 cell array of scalers indicating the number of candidate L_1/L_2 parameters to be used for parameter selection. `paramsel_homkl` will guess the candidate parameters as discussed in section 1.3.3. By default `mkl.npar` = $\{[25], [5]\}$.
  - `mkl.parrange`: A 1×2 cell array of vectors indicating the candidate L_1/L_2 parameters to be used for parameter selection. `paramsel_homkl` will ignore `mkl.npar` if `mkl.parrange` is specified.
- For dual parameter estimation:
  - `mkl.npar_mkl`: A 1×2 cell array of scalers indicating the L_1/L_2 parameters to be used for `rls_dual_mkl`. User may use this field if they prefer to set the parameter directly without going through the parameter selection stage.

Aside from the fields listed above, `opt.mkl` also contains other fields that are used to control various techinical aspects in parameter selection (e.g. whether to use continuation strategy) and PFBS-based optimization (e.g. maximum number of iterations allowed, convergence criteria). A complete list of these option fields can be found in Appendix B. These fields can be set automatically by the procedure `utils/gurls_defopt_mkl`.

2.2 Example Pipeline

Below example shows how to execute a `GURLS_mkl` pipeline through the GURLS framework:

```
%%%% Step 1: Option Configuration %%%%
name = 'mkl_demo';
opt = gurls_defopt(name);
opt = gurls_defopt_mkl(opt); % initialize opt.mkl
opt.mkl.type = ... % specify kernel type/parameter by hand
    {{'kernel_rbf', 1:5}, {'kernel_linear', 0}};

%%%% Step 2: Task Sequence Configuration %%%%
opt.seq = {...
    'split:ho', ...
    % kernel_mkl MUST be called before paramsel/rls
    'kernel:mkl', ...
    'paramsel:homkl', ...
    'rls:dual_mkl', ...
    'predkernel:traintest_mkl', ...
    'pred:dual_mkl', ...
    %perf support only rmsestd (regression)\macroavg (classification)
    'perf:rmsestd'};

opt.process{1} = [2,2,2,2,0,0,0];
opt.process{2} = [3,3,3,3,2,2,2];

%%%% Step 3: Execute Task Sequence %%%%
gurls(Xtr, ytr, opt, 1);
gurls(Xte, yte, opt, 2);
```

As shown, to execute a `GURLS_mkl` pipeline, one only need to tweak their standard GURLS pipeline as below:

1. In `Option Configuration` step, after initializing `opt` using `gurls_defopt`:
    (a) Initialize `opt.mkl` using `gurls_defopt_mkl` (Required)
    (b) Specify `opt.mkl.type` by hand (supports only `kernel_rbf` and `kernel_linear`).
    (c) Optionally, tweak `mkl.parrange`, `mkl.npar`, etc to customize `paramsel` procedure.
    (d) Optionally, specify `opt.mkl.par_mkl` to skip `paramsel` procedure.
2. In `Task Sequence Configuration` step, one must
    (a) Call `kernel_mkl` to compute MKL kernels.
    (b) Use either `rmsestd` (for regression) or `macravg` (for classification) for performance metric.

3 Example

In this section we apply `GURLS_mkl` to artifical and real datasets to illustrate ways to visualize estimation results and customize pipeline. Script for these examples can be found in folder `gurls/demo` under file names `demo_mkl_*.m`.

3.1 Regression on Gaussian Data with Mixed Kernel

In this example, Gaussian data is generated under a mixture of linear and RBF kernel. Namely, for $\boldsymbol{\alpha}_{n\times1}, \boldsymbol{\beta}_{p\times1} \sim MVN(\mathbf{0}, \mathbf{I})$, $\mathbf{X}_{n\times p} = [\mathbf{x}_1, \ldots, \mathbf{x}_p] \overset{iid}{\sim} MVN(\mathbf{0}, \mathbf{I})$, and $\mathbf{K} = \texttt{RBFKernel}(\mathbf{X}, \sigma = 10)$, we generate response as $\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \mathbf{K}\boldsymbol{\alpha}$.

Setting $n = 500, p = 50$, we apply MKL framework to above data with $M + 1$ kernels, including $M = 50$ RBF kernels with $2\sigma^2 \in [1.2^{0,\ldots,M-1}]$, and one linear kernel. We perform 5-fold hold-out parameter selection, with 5 candidate λs and 25 candidate τ's within the range specified in section 1.3.3. Note since contiuation strategy may underfit the data, we choose not to use continuation strategy in this task by setting:

```
opt.mkl.strategy = false;
```

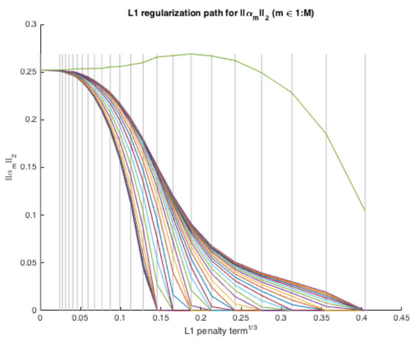
We considered standardized RMSE as the performance metric for regression, which is defined as:

$$\texttt{rmsestd}(\hat{\mathbf{y}}, \mathbf{y}) = \sqrt{\frac{||\hat{\mathbf{y}} - \mathbf{y}||_2^2}{||\mathbf{y}||_2^2}}$$

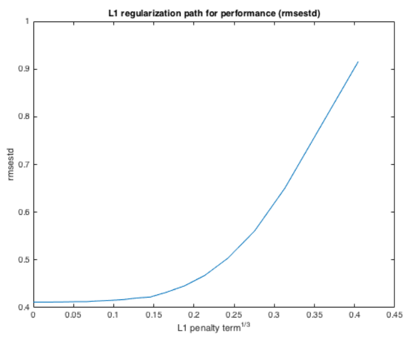
As shown, standardized RMSE is more informative than RMSE since it can be interpretated as the percentage of variation in $\mathbf{y}$ that is not explained by $\hat{\mathbf{y}}$, and it is expected to range between 0 and 1.

After parameter selection, we can inspect the effect of tunning parameters on the norm of kernel-specific effects $||\boldsymbol{\alpha}_j||_2$ and the model fit (`rmsestd`) by considering the regularization path along candidate τ's, fixing λ at the selected value. This can be accomplished using the `plot_mkl_path` function with below code:

```
plot_mkl_path(X_tr, y_tr, opt, 'norm'); % Figure (a)
plot_mkl_path(X_tr, y_tr, opt, 'perf'); % Figure (b)
```



(a) Kernel Norms



(b) Standardized RMSE

Figure 2: L_1 Regularization Path for Regression on Gaussian Data, with Continuation Strategy

As shown, as τ increase, all $||\boldsymbol{\alpha}_j||_2$'s decrease uniformly except for the linear kernel (the green line), whose effect grow slightly when the effect of all other RBF kernels are penalized toward zero, but eventually start decreasing when τ become too large. The hold-out standardized RMSE increased uniformly with τ, indicating the fact that using multiple kernels with moderate sparsity requirement is preferred in current senario.

We may also perform explicit kernel selection among the candidate kernels by inspecting the size of $||\boldsymbol{\alpha}_j||_2$, which is stored in `opt.paramsel.norm_path` as a `nL`$_2$ $\times$ `nKernel` $\times$ `nholdout` matrix. For example, if we want to find out the 5 most important kernels, we can define a term `kernel_importance` which is the sum of $||\boldsymbol{\alpha}_{j,\tau_k}||_2$ over all possible τ_k's (which is an approximation of the area under the kernel regularization paths in Figure 2 (a)). This can be calculated as:

```
norm_summary = median(opt.paramsel.norm_path, 3); % median norm across holdout
kernel_importance = sum(norm_summary, 1); % sum of norm across tau
[norm_value, norm_order] = sort(kernel_importance, 'descend');
>> [norm_order(1:5); norm_value(1:5)]

ans =

   51.0000    50.0000    49.0000    48.0000    47.0000
    5.9023     4.1103     4.1069     4.1027     4.0976
```

As shown, `GURLS_mkl` selected the linear kernel (`norm_order` $= 51$) as the most important kernel, and the five rbf kernels (`norm_order` $= 47:50$) with the largest sigma values ($\sigma \in [46, 61]$) as the most important RBF kernel. It is slightly surprising to see that the RBF kernel corresponding to the true data generation mechanism, i.e. RBF kernel with $\sigma = 10$, is not selected. It is likely due to the fact that in the soft-thresholding operator $\mathbf{S}_{\frac{\tau}{\sigma}}$ we imposed uniform threshold ($\frac{\tau}{\sigma}$) on all kernel specific norms $\sqrt{\boldsymbol{\alpha}_j^T \mathbf{K}_j \boldsymbol{\alpha}_j}$, hence implicitly perfering $\mathbf{K}_j$'s with larger norm/eigenvalue.

Finally, we compare the performance of current model against the true MKL model using data generation kernel by checking prediction `rmsestd` on a validation sample of $n = 200$. The data generation model can be fit directly without `paramsel` by supplying tunning parameters τ, λ:

```
opt.mkl.type = {{'kernel_rbf', 10}, {'kernel_linear', 0}};
opt.mkl.par_mkl = {[0], [0]};
opt.seq = {... % skip paramsel
    'split:ho', 'kernel:mkl', 'rls:dual_mkl', 'predkernel:traintest_mkl', ...
    'pred:dual_mkl',  'perf:rmsestd'};
opt.process{1} = [2,2,2,0,0,0];
opt.process{2} = [3,3,3,2,2,2];
gurls(X_tr, y_tr, opt, 1);
gurls(X_va, y_va, opt, 2);
opt.perf.rmsestd

ans =

    0.0601
```

As seen, the validation `rmsestd` under true data generation model is 0.0601, while the validation `rmsestd` for our current model is 0.0675. We have achieved similar model performance using MKL model without knowledge of the true data generation mechanism.

3.2 Classification on Ionosphere Data

We take this example from Xu et al. [2013] to illutrate the usage of `GURLS_mkl` in classification tasks. In this example, we consider the Ionosphere data taken from the UCI repository [2] describing radar data collected in Goose Bay, Labrador to detect whether there is evidence of some type of structure in the ionosphere. Following Xu et al. [2013], we applied MKL framework with 5 RBF kernels with $\sigma = 1 : 0.2 : 4$ and 1 linear kernel. The model pipeline is similiar to the pipeline for regression except that one need to set the hold-out performance metric to `macro_avg`:

```
opt.hoperf = @perf_macroavg;
```

The accuracy on validation sample is 0.9857, comparible to the performance of L_2 MKL (i.e. RLS MKL) in Xu et al. [2013].

[2] https://archive.ics.uci.edu/ml/datasets/Ionosphere

Appendix A List of `GURL_mkl` Functions

1. Parameter Specification
   - `utils/gurls_defopt_mkl`: generate default option struct `opt.mkl`
2. Kernel Generation
   - `kernel/kernel_mkl`:
     use information supplied in `opt.mkl.type` to generate MKL kernels. Supports rbf and linear kernels.
3. Parameter Selection
   - `paramsel/paramsel_homkl`
     Using kernel specified in `opt.kernel.K_mkl` to perform hold-out parameter selection. Candidate tunning parameter are either taken directly from `opt.mkl.parrange` or guessed automatically. Continuation strategy can be activated/deactivated using `opt.ml.strategy` = `true/false`.
   - `utils/paramsel_L1ratioguesses`: function to guess candidate L_1 tunning parameters.
4. Optimization
   - `optimizers/rls_dual_mkl`
     interface to pass the selected tunning parameter in `opt.paramsel.par_mkl` or user specified tunning parameter in `opt.mkl.par_mkl` to the core optimization routine `rls_dual_mkl_pfbs`.
   - `optimizers/rls_dual_mkl_pfbs`
     Core optimization procedure. Read in tunning parameters and other techinical parameters (`iter_max`, `crit`. etc) from `opt` to perform PFBS, then return estimated $\boldsymbol{\alpha}$.
   - `utils/ConsoleProgressBar`
     Utility function to plot the progress of holdout selection/PFBS fitting.
5. Prediction
   - `kernel/predkernel_traintest_mkl`: generate prediction kernels.
   - `pred/pred_dual_mkl` perform prediction
6. Performance assessment
   - `perf/perf_rmsestd`: Calculate standardized RMSE ($||\hat{\mathbf{y}} - \mathbf{y}||_2/||\mathbf{y}||_2$)
   - `summary/plot_mkl_path`:
     Read in `opt.paramsel.perf`$_\mathtt{p}$`ath` and `opt.paramsel.norm`$_\mathtt{p}$`ath` generated by `paramsel_homkl` to plot the regularization path of kernel-specific estimate norm/model performance metric over candidate τ's.

Appendix B List of `GURL_mkl`-related Options

1. **Option** `mkl`
   - `type`: kernel function and parameter value used for the M kernels, $M \times 1$ cell array of structs
   - `parrange`: candidate tunning parameters, 2×1 cell array of vectors
   - `npar`: number of candidate tunning parameter to guess, 2×1 cell array of scalers.
   - `smallnumber`: lower limit of non-zero candidate tunning parameters.
   - `verbose`: whether to print fitting progress in `paramsel/rls`. e.g. $\{'\mathtt{paramsel}', \mathtt{true}, '\mathtt{rls}', \mathtt{true}\}$
   - `iter_max`: max number of iteration for PFBS in `paramsel/rls`. e.g. $\{'\mathtt{paramsel}', \mathtt{1e4}, '\mathtt{rls}', \mathtt{1e5}\}$
   - `crit`: convergence criteria on performance metric in PFBS. e.g. $\{'\mathtt{paramsel}', \mathtt{1e}-\mathtt{3}, '\mathtt{rls}', \mathtt{1e}-\mathtt{5}\}$
   - `strategy` whether to use continuation strategy in `paramsel`.
   - `par_mkl` tunning parameter to be used for `rls`, if one wish to skip `paramsel`.
2. **Option** `kernel`
   - `K_mkl` $\mathtt{n} \times \mathtt{n} \times \mathtt{M}$ matrix specifying MKL kernel matrices
   - `eig_mkl` $\mathtt{M} \times \mathtt{1}$ doubles of the norms (largest eigenvalue) of the largest matrix.
3. **Option** `paramsel`
   - `par_mkl`: $(\frac{\mu}{\sigma}, \frac{\tau}{\sigma})$ selected by `paramsel`, 2×1 cell array.
   - `perf_path`: holdout performence (`rmsestd/macroavg`) along candidate τ, (λ fixed at selected value).
   - `norm_path`: estimated $||\boldsymbol{\alpha}_j||$ (`rmsestd/macroavg`) along candidate τ, (λ fixed at selected value).
   - `guesses_mkl`: candidate parameter used in `paramsel`, 2×1 cell array of scalers.
   - `cont_strategy`: whether continuation strategy is used in `paramsel`, logic.
4. **Option** `rls`
   - `C_mkl` $M \times n$ estimate of $\boldsymbol{\alpha}_j$'s.

Appendix C References

References

F Bach, G Lanckriet, and M Jordan. Multiple kernel learning, conic duality, and the smo algorithm. *ACM International Conference Proceeding Series*, 69, 2004.

L Rosasco, S Mosci, M Santoro, A Verri, and S Villa. Iterative projection methods for structured sparsity regularization. Technical report, MIT, 2009.

S Mosci, L Rosasco, M Santoro, A Verri, and S Villa. Solving structured sparsity regularization with proximal methods. *Machine Learning and Knowledge Discovery in Databases*, II:418, September 2010.

X Xu, I Tsang, and D Xu. Soft margin multiple kernel learning. *IEEE Transactions on Neural Networks and Learning Systems*, 24:749–761, 2013.