FMPR: Fused Multi-task Penalised Regression

Gad Abraham
gad.abraham@unimelb.edu.au

September 14, 2012

1 Introduction

High throughput technologies, such as gene expression microarrays and single nucleotide polymorphism (SNP) microarrays, have made it possible to assay thousands and sometimes millions of potential biological markers, in order to detect associations with clinical phenotypes such as human disease. At the same time, the definition of what is a phenotype has become more encompassing. Rather than consider only macro-level clinical phenotypes such as presence of disease, other finer-level phenotypes, such as gene expression and metabolite levels, are becoming of interest. For example, it is now routine to scan SNPs for potential expression quantitative loci (eQTL), regulating the expression of genes (Mackay et al., 2009), or the levels of metabolites in the blood (Inouye et al., 2010a,b; Tukiainen et al., 2011). In this case, there can be hundreds to thousands of phenotypes, which typically show high degrees of correlation with each other, however, many existing approaches treat these phenotypes as independent.

`FMPR` is an R package for linear modelling of multiple related phenotypes (tasks) measured for the same samples. `FMPR` uses the correlation between the phenotypes to borrow statistical power, thus potentially increasing the ability to detect weaker associations that would have been missed using methods that do not account for task relatedness, such as the lasso (Tibshirani, 1996) or ridge regression.

2 Methods

Assume we have an $N \times p$ input matrix $\mathbf{X}$ (such as SNPs) and an $N \times K$ output matrix $\mathbf{Y}$ (such as gene expression levels), where N is the number of samples, p is the number of inputs (variables), and K is the number of tasks (outputs). We wish to use a linear model of the outputs

$$\mathbf{Y} = \mathbf{XB} + \mathbf{E}, \tag{1}$$

where $\mathbf{B}$ is a $p \times K$ matrix of weights and $\mathbf{E} \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$ is iid random noise. We assume that both $\mathbf{X}$ and $\mathbf{Y}$ are standardised to zero mean and unit variance, therefore, there is no intercept term in the model. The ith row of $\mathbf{X}$ is denoted x_i (a p-vector of observations for the ith sample). The kth column of $\mathbf{B}$ is denoted β_k (the p-vector of weights for the kth task).

The graph-guided fused lasso (GFlasso) (Kim and Xing, 2009) assumes that correlations in the outputs $\mathbf{Y}$ are caused by to similar inputs acting across multiple outputs, such as one eQTL affecting the expression level of multiple related genes. This assumption is encoded in the model by imposing an ℓ_1 fusion penalty on the difference in weights $\mathbf{B}_{jk}$ for each input $j = 1, \ldots, p$ across the different tasks $k = 1, \ldots, K$. The fusion penalty encourages sparsity in the differences, leading to borrowing of power across correlated outputs and tending to select the same inputs across multiple related outputs. In addition, a standard lasso ℓ_1 penalty is imposed on the weights within each task to promote sparsity in the model. The GFlasso-penalised loss function for the

squared loss (linear regression) is

$$\mathbf{B}^* = \arg\min_{\mathbf{B}\in\mathbb{R}^{p\times K}} \frac{1}{2}\sum_{k=1}^{K}\sum_{i=1}^{N}(y_{ik} - x_i^T\beta_k)^2 + \lambda\sum_{k=1}^{K}\sum_{j=1}^{p}|\beta_{jk}| + \gamma\sum_{(m,l)\in\mathcal{E}} f(r_{ml})\sum_{j=1}^{p}|\beta_{jm} - \text{sign}(r_{ml})\beta_{jl}|, \quad (2)$$

where $\mathbf{B} = [\beta_1, \ldots, \beta_K]$, r_{ml} is the Pearson correlation between the mth and lth phenotypes, $f(r_{ml})$ is a function monotonic in the correlation, $\mathcal{E}$ is the set of inter-task edges induced by thresholding the Pearson correlation r_{ml} ($\mathcal{E}$ is the same for all the p variables), and $\lambda \geq 0$, $\gamma \geq 0$ are the lasso and fusion penalties, respectively. Since the fusion penalty is not differentiable and not separable (Tseng, 2001; Friedman et al., 2007), the standard coordinate descent which is used in lasso cannot be applied to solve the GFlasso problem, as the optimisation process may get stuck in suboptimal solutions where no single move will minimise the loss further. Therefore, the GFlasso problem is solved using approaches such as the Smoothed Proximal Gradient (SPG) (Chen et al., 2012)[1].

Instead of using an ℓ_1 fusion, another option is to use an ℓ_2 fusion penalty, which has the nice properties that it is differentiable and therefore can be optimised using simple and fast methods such as coordinate descent, leading to substantial speedups over methods such as SPG. We formulate our fused multitask penalised squared loss as follows:

$$B^* = \arg\min_{B\in\mathbb{R}^{p\times K}} \frac{1}{2}\sum_{k=1}^{K}\sum_{i=1}^{N}(y_{ik} - x_i^T\beta_k)^2 + \lambda\sum_{k=1}^{K}\sum_{j=1}^{p}|\beta_{jk}| + \frac{\gamma}{2}\sum_{(m,l)\in\mathcal{E}} f(r_{ml})\sum_{j=1}^{p}[\beta_{jm} - \text{sign}(r_{ml})\beta_{jl}]^2. \quad (3)$$

The λ penalty tunes sparsity within each task (lasso). The γ penalty shrinks the differences between weights for related tasks towards zero, but unlike the GFlasso, does not necessarily encourage sparsity in differences between the weights for related tasks. The lasso regression is a special case of the fused ℓ_2 loss, achieved by setting $\gamma = 0$.

3 Prerequisites

There are several prerequisites to running `FMPR` successfully.

3.1 Missing data

`FMPR` currently does not support any missing data (`NA`, `Inf`, etc) in either the inputs or the outputs. These must be imputed or the offending data removed.

3.2 Output transformations

The linear model in `FMPR` assumes that the outputs $\mathbf{Y}$ are approximately normally distributed. Gross deviations from normality or the presence of many outliers may invalidate the assumptions of the model. We recommend examining the outputs and possibly transforming them to approximate normality using the Box-Cox power transformation (see function `BoxCox` in package `forecast`, for example).

[1] `FMPR` includes a C implementation of the original SPG MATLAB code from `http://www.cs.cmu.edu/~xichen/Code/SPG_Multi_Graph.zip`; see the function `run.spg.test`.

3.3 Defining the graph

A suitable graph induction function is required in order to create a graph from the outputs $\mathbf{Y}$. The function must be monotonic in the absolute value of the correlation. By default, `FMPR` uses the correlation matrix of $\mathbf{Y}$, and transforms it using either the square $f(r_{ml}) = r_{ml}^2$ (cortype 2) or the absolute value $f(r_{ml}) = |r_{ml}|$ (cortype 1). In R:

```
C <- gennetwork(Y, corthresh=0, cortype=2)
```

3.4 Penalties

The range of penalties must be defined by the user. Typically, we use a descending sequence of λ values from the smallest value that makes all $\widehat{\mathbf{B}}$ zero (ignoring the fusion penalty) and ending at some small fraction of it (say, 0.01 of the largest value). For γ, we use a multiplicative scale such as $10^{-5,\ldots,5}$. By default, `FMPR` uses grid search over the (λ, γ) penalty combinations, within cross-validation, estimating the R^2 over all tasks for each penalty pair. See below for an example of how to produce a suitable penalty range.

4 Toy Example

First we load the packages. The `doMC` package is recommended, allowing R to use multiple cores in parallel (here, two cores). Otherwise, `FMPR` will run serially on one core.

```
> library(FMPR)
> library(doMC)
> registerDoMC(cores=2)
```

We start by generating some random data, with identical weights of 0.1 across the tasks, and sparsity in each task (most weights are zero). The outputs $\mathbf{Y}$ are corrupted by iid Gaussian noise.

```
> N <- 100
> p <- 100
> K <- 30
> w <- 0.1
> B <- getB(p=p, K=K, w=w, type="same", sparsity=0.8)
> print(table(sign(B)))

   0    1
2430  570

> X <- scale(matrix(rnorm(N * p), N, p))
> Y <- scale(X %*% B + rnorm(N * K))
```

Now we induce the inter-task graph based on the square of the correlation of $\mathbf{Y}$. Note that `FMPR` and `SPG` use a different format to indicate task relatedness.

```
> C <- gennetwork(Y, corthresh=0, cortype=2)
```

Next we define the penalties λ and γ, and the number of cross-validation folds for optimising the penalties.

```
> l <- max(maxlambda1(X, Y))
> ngrid <- 25
> lambda <- 2^seq(-0.01, -10, length=ngrid) * l
> gamma <- c(0, 10^seq(-5, 5, length=ngrid))
> nfolds <- 5
```

Now we run grid search for `FMPR` and `SPG` (implementing GFlasso), to find the optimal penalties, in terms of cross-validated R^2. Note that the object `C` is not passed to the cross-validation, as it is re-estimated from scratch within each cross-validation replication. Instead, we pass `cortype=2`:

```
> # L2 fusion penalty
> system.time({
+    opt.f <- optim.fmpr2(X=X, Y=Y, cortype=2, corthresh=0,
+       lambda=lambda, gamma=gamma, nfolds=nfolds)
+ })

inner fold 1
inner fold 2
inner fold 3
inner fold 4
inner fold 5
   user  system elapsed
323.791   1.242 188.798

> # L1 fusion penalty
> system.time({
+    opt.s <- optim.spg(X=X, Y=Y, cortype=2, corthresh=0,
+       lambda=lambda, gamma=gamma, nfolds=nfolds)
+ })

inner fold 1
spg: svd took 0.006 0 0.006 0 0
CNorm: 0.2543526
inner fold 2
spg: svd took 0.004 0 0.004 0 0
CNorm: 0.4592926
inner fold 3
spg: svd took 0.006 0 0.006 0 0
CNorm: 0.2792513
inner fold 4
spg: svd took 0.005 0 0.005 0 0
CNorm: 0.261811
inner fold 5
spg: svd took 0.006 0 0.005 0 0
CNorm: 0.236378
   user  system elapsed
750.916   1.733 437.379
```

The optimal penalties for each model are then used to train a model on the entire dataset, and return just one estimated matrix $\widehat{\mathbf{B}}$ (note the `simplify=TRUE` argument). Since the lasso is a special case of the ℓ_2 fusion loss with $\gamma = 0$, we can get the lasso solution as well.

```
> f <- fmpr2(X=X, Y=Y, C=C, lambda=opt.f$opt["lambda"],
+    gamma=opt.f$opt["gamma"], simplify=TRUE)
> s <- spg(X=X, Y=Y, C=C, lambda=opt.s$opt["lambda"],
+    gamma=opt.s$opt["gamma"], simplify=TRUE)

spg: svd took 0.007 0 0.007 0 0
CNorm: 0.228369

> # Lasso, special case of FMPR
> w <- which(opt.f$R2[, 1, ] == max(opt.f$R2[, 1, 1]))
> l <- fmpr2(X=X, Y=Y, lambda=lambda[w], gamma=0, simplify=TRUE)
```

Next, we estimate receiver-operating characterstic (ROC) and precision-recall (PRC) curves for the absolute value of the estimated model weights versus the true non-zero status of the weights $\mathbf{B}$, and plot them in Figure 1.

```
> measures <- list(ROC=c("sens", "spec"), PRC=c("prec", "rec"))
> res <- lapply(list(FMPR=f, GFlasso=s, Lasso=l), function(B2) {
+    lapply(measures, function(m) {
+       performance(prediction(
+          labels=as.numeric(B != 0),
+          predictions=as.numeric(abs(B2))
+          ), m[1], m[2])
+    })
+ })

> par(mfrow=c(1, 2), mar=c(4, 4, 2, 2) + 0.1)
> plot(res$FMPR$ROC, col=1, main="ROC", cex=2, lwd=3,
+    ylim=c(0, 1), xlim=c(0, 1))
> plot(res$GFlasso$ROC, col=2, add=TRUE, lty=2, lwd=3)
> plot(res$Lasso$ROC, col=3, add=TRUE, lty=3, lwd=3)
> abline(1, -1, lty=4, col=4, lwd=3)
> legend(x=0, y=0.2,
+    legend=c("FMPR-w2", "GFlasso-w2", "Lasso", "Random"),
+    col=1:4, lwd=4, lty=1:4)
> plot(res$FMPR$PRC, col=1, main="PRC", cex=2, lwd=3,
+    ylim=c(0, 1), xlim=c(0, 1))
> plot(res$GFlasso$PRC, col=2, add=TRUE, lty=2, lwd=3)
> plot(res$Lasso$PRC, col=3, add=TRUE, lty=3, lwd=3)
> abline(h=mean(B != 0), lty=4, col=4, lwd=3)
>
```

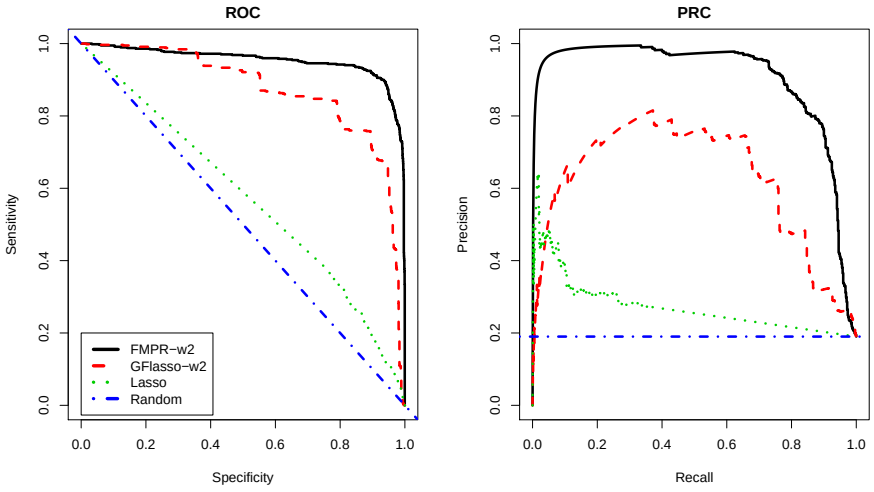

Figure 1: ROC and PRC curves for recovering the true non-zero weights $\mathbf{B}$. The Random model is the null model with no predictive ability (area under ROC curve of 0.5 for ROC and proportion of non-zero entries of $\mathbf{B}$ for PRC).

Finally, we generate an independent test dataset using the same weights $\mathbf{B}$,

```
> X2 <- scale(matrix(rnorm(N * p), N, p))
> Y2 <- scale(X2 %*% B + rnorm(N * K))
```

and estimate the R^2 over all tasks for each method in the independent data

```
> sapply(list(FMPR=f, GFlasso=s, Lasso=l), function(B2) {
+    pr <- X2 %*% B2
+    R2(pr, Y2)
+ })

      FMPR    GFlasso      Lasso
0.13204757 0.08926211 0.01390364
```

References

T. F. C. Mackay, E. A. Stone, and J. F. Ayroles. The genetics of quantitative traits: challenges and prospects. *Nat. Rev. Genet.*, 10:565–577, 2009.

M. Inouye, J. Kettunen, P. Soininen, S. Ripatti, L. S. Kumpula, E. Hämäläinen, P. Jousilahti, A. J. Kangas, S. Männistö, M. J. Savolainen, A. Jula, J. Leiviskä, A. Palotie, V. Salomaa, M. Perola, M. Ala-Korpela, and L. Peltonen. Metabonomic, transcriptomic, and genetic variation of a population cohort. *Mol. Sys. Biol.*, 6:441, 2010a.

M. Inouye, K. Silander, E. Hamalainen, V. Salomaa, K. Harald, et al. An Immune Response Network Associated with Blood Lipid Levels. *PLoS Genet.*, 6:e1001113, 2010b.

T. Tukiainen, J. Kettunen, A. J. Kangas, L.-P. Lyytikåinen, et al. Detailed metabolic and genetic characterization reveals new associations for 30 known lipid loci. *Hum. Mol. Genet.*, 2011. To appear.

R. Tibshirani. Regression Shrinkage and Selection via the Lasso. *J. R. Statist. Soc. B*, 58:267–288, 1996.

S. Kim and E. P. Xing. Statistical estimation of correlated genome associations to a quantitative trait network. *PLoS Genet.*, 5:e1000587, 2009.

P. Tseng. Convergence of a Block Coordinate Descent Method for Nondifferentiable Minimization. *J. Opt. Theory Appl.*, 109:475–494, 2001.

J. Friedman, T. Hastie, H. Höfling, and R. Tibshirani. Pathwise coordinate optimization. *Ann. Appl. Statist.*, 1:302–332, 2007.

X. Chen, Q. Lin, S. Kim, J. G. Carbonell, and E. P. Xing. A smoothing proximal gradient method for general structured sparse regression. *Ann. Appl. Statist.*, 2012. To appear.


```
> sessionInfo()

R version 2.15.1 (2012-06-22)
Platform: x86_64-apple-darwin9.8.0/x86_64 (64-bit)

locale:
[1] en_AU.UTF-8/en_AU.UTF-8/en_AU.UTF-8/C/en_AU.UTF-8/en_AU.UTF-8

attached base packages:
[1] grid      methods   stats     graphics  grDevices utils     datasets
[8] base

other attached packages:
 [1] doMC_1.2.5        multicore_0.1-7   iterators_1.0.6   FMPR_0.01
```

```
 [5] reshape_0.8.4     plyr_1.7.1        ROCR_1.0-4        gplots_2.11.0
 [9] KernSmooth_2.23-8 caTools_1.13      bitops_1.0-4.1    gdata_2.11.0
[13] gtools_2.7.0      Matrix_1.0-6      lattice_0.20-10   foreach_1.4.0
[17] ggplot2_0.9.1     MASS_7.3-21

loaded via a namespace (and not attached):
 [1] codetools_0.2-8    colorspace_1.1-1   compiler_2.15.1    dichromat_1.2-4
 [5] digest_0.5.2       labeling_0.1       memoise_0.1        munsell_0.3
 [9] proto_0.3-9.2      RColorBrewer_1.0-5 reshape2_1.2.1     scales_0.2.1
[13] stringr_0.6.1      tools_2.15.1
```