www.sciencetranslationalmedicine.org/cgi/content/full/5/181/181ra50/DC1

Supplementary Materials for

Development of a Prognostic Model for Breast Cancer Survival in an Open Challenge Environment

Wei-Yi Cheng, Tai-Hsien Ou Yang, Dimitris Anastassiou*

*Corresponding author. E-mail: da8@columbia.edu

Published 17 April 2013, *Sci. Transl. Med.* **5**, 181ra50 (2013)
DOI: 10.1126/scitranslmed.3005974

The PDF file includes:

Table S1. **Molecular features used in the model.**

Metagene	**Gene members**	**Condition**	**Note**
CIN feature	*CENPA*		pan-cancer
	DLGAP5		
	MELK		
	BUB1		
	KIF2C		
	KIF20A		
	KIF4A		
	CCNA2		
	CCNB2		
	NCAPG		
MES feature	*COL5A2*	Patients with early-stage tumors (no positive lymph node and tumor size < 30 mm)	pan-cancer
	VCAN		
	SPARC		
	THBS2		
	FBN1		
	COL1A2		
	COL5A1		
	FAP		
	AEBP1		
	CTSK		
LYM feature	*PTPRC*	1. ER- 2. Positive lymph node number > 3	pan-cancer
	CD53		
	LCP2		
	LAPTM5		
	DOCK2		
	IL10RA		
	CYBB		
	CD48		
	ITGB2		
	EVI2B		
Estrogen Receptor Attractor (ER)	*AGR3*		breast cancer specific
	CA12		
	FOXA1		
	GATA3		
	MLPH		

	AGR2		
	ESR1		
	TBC1D9		
HER2 Attractor	*ERBB2*		breast cancer specific
	PGAP3		
	STARD3		
	MIEN1		
	GRB7		
	PSMD3		
	GSDMB		
Adipocyte Attractor	*ADIPOQ*		breast cancer specific
	ADH1B		
	FABP4		
	PLIN1		
	RBP4		
	PLIN4		
	G0S2		
	GPD1		
	CD36		
	AOC3		
Chr8q24.3 Attractor	*EXOSC4*		pan-cancer
	PUF60		
	BOP1		
	SLC52A2		
	SHARPIN		
	HSF1		
	FBXL6		
	CYC1		
	SCRIB		
	GPAA1		
Chr7p11.2 Attractor	*MRPS17*		attractor involving EGFR
	LANCL2		
	SEC61G		
	CCT6A		
	CHCHD2		
	EGFR		
ZMYND10 Metagene	*ZMYND10*		highly co-expressed genes with protective
	LRRC48		

	CASC1		CI
FGD3-SUSD3 Metagene	*FGD3*		adjacent genes with most protective CI
	SUSD3		
PGR-RAI2 Metagene	*PGR*		highly co-expressed genes with protective CI
	RAI2		
Chr15q26.1 Attractor	*PRC1*		pan-cancer
	BLM		
	FANCI		

Table S2. Cox proportional hazards model trained on molecular and clinical features on the basis of AIC.

Molecular Features		
Feature*	**Coefficient**	***P value***
CIN	0.321	8.10E-07
ER	0.151	3.80E-06
FGD3-SUSD3	-0.169	2.90E-05
MES--early stage	0.184	2.70E-03
LYM-- lymph node number > 3	0.431	7.60E-03
LYM -- ER-	-0.248	3.80E-02
HER2	0.075	5.80E-02
PGR-RAI2	-0.143	5.90E-02
Clinical Features		
Feature	**Coefficient**	***P value***
Number of positive lymph nodes	0.062	<2E-16
Age	0.032	<2E-16
Tumor Size	0.012	6.50E-12
ER-negative	0.545	3.40E-10
Radiation Therapy	-0.244	4.80E-04
Histological subtype -- Medullary	-1.100	2.20E-03
Histological subtype -- Tubular	-0.652	1.70E-02
Grade 3	0.183	1.50E-02

* The symbols represent the metagenes given in Table S1

Table S3. Cox proportional hazards models trained on empirically selected features.

Feature	**Coefficient**	***P value***
Number of positive lymph nodes	0.062	<2E-16
Age	0.034	<2E-16
*FGD3-SUSD3**	-0.181	1.30E-09
CIN*	0.281	1.70E-06
MES -- early stage*	0.187	3.40E-03
LYM -- ER-*	-0.279	1.40E-02

- Symbols represent the metagenes given in Table S1

Table S4. Source code of the final model. The source code of the final model is divided in two parts. (i) syn1417992_OSDS.R: a meta-model that trains the ensemble model twice using OS-based and DS-based survival data. It combines predictions from the two trained models by taking the weighted average of the two, where the weights are determined by maximizing the training score. (ii) syn1417992_fix.R: the ensemble model as described in Materials and Methods and Fig. 5. in the main text. Source code is available in Sage Synapse under ID syn1417992.

1. syn1417992_OSDS.R

```
require(rms)

require(MASS)

require(survival)

require(predictiveModeling)

require(gbm)

require(caret)

require(devtools)


# install the latest version of DreamBox7 package from github

install_github(repo="DreamBox7", username="weiyi-bitw", ref="master")

library(DreamBox7)
```

```
setRefClass(Class = "PredictiveModel")


#' GoldiloxModel

#'

#' Modified from DemoClinicalOnlyModel from BCC challenge. This meta-model trains the same model for both OS and DS

#' survival, and combine two predictions by taking weighted average where the weights were chosed to maximize

#' the performance on training set.

#'

#' For details of functions, see the DreamBox7 package at:

#'   https://github.com/weiyi-bitw/DreamBox7

#' The source code is available at:

#'   https://github.com/weiyi-bitw/BCCModels

#'

#' @author Wei-Yi Cheng

#' @Revise Tai-Hsien Ou Yang

#' @export


GoldiloxModel <- setRefClass(

  Class  = "GoldiloxModel",

  contains = "PredictiveModel",
```

```
  fields   = c("model", "attractome", "annot", "predictions","mdns",
"chosenProbes_g","chosenProbes", "dssurv", "w"),

  methods  = list(

    # ============================

    #

    # initialize(...)

    #

    # function called when the model is initialized

    #

    # ============================

    initialize = function(...){

      return(.self) # do nothing

    },

    # ============================

    #

    # customTrain(exprData, copyData, clinicalFeaturesData,

    #             clinicalSurvData, clinicalSurvData_DS, ...)

    #

    # training method for the model

    #   exprData: An ExpressionSet (as in Biobase package) of gene expression data

    #   copyData: An ExpressionSet of copy number variation data

    #   clinicalFeaturesData: Data frame with clinical information

    #   clinicalSurvData: a Surv object with overall survival information
```

```
#   clinicalSurvData_DS: a Surv object with disease-specific survival information

#

# ============================


customTrain = function(exprData, copyData,
clinicalFeaturesData,clinicalSurvData,clinicalSurvData_DS,...){

  if(class(clinicalSurvData) != "Surv"){

    stop("Expecting 'responseData' object of type 'Surv'")

  }

  # load the structure of underlying ensemble models

  source_url("https://raw.github.com/weiyi-
bitw/BCCModels/master/syn1417992_fix.R")

  # trained an OS-based model

  gdModel.os <- GoldiModel$new()

  gdModel.os$customTrain(exprData, copyData,
clinicalFeaturesData,clinicalSurvData)

  # fit the training set and make predictions

  p1 <- gdModel.os$customPredict(exprData, copyData, clinicalFeaturesData)



  # trained an DS-based model

  gdModel.ds <- GoldiModel$new()

  gdModel.ds$customTrain(exprData, copyData, clinicalFeaturesData,
clinicalSurvData_DS)

  p2 <- gdModel.ds$customPredict(exprData, copyData, clinicalFeaturesData)
```

```
  # no need for CNV data and expression data anymore, delete them
  rm(copyData)
  rm(exprData)

  pp <- rbind(p1, p2)

  # find the best way to combine the features by a brute-force method
  # optimizing the training score
  weights <- BFFW(pp, clinicalSurvData, w = rep(1, 2), maxIter=1000)
  # save the weights for making predictions in the future
  .self$w <- weights
  # save the trained models for making predictions
  .self$model <- list(
    osmodel = gdModel.os,
    dsmodel = gdModel.ds
  )
},
# ============================
#
# customPredict(exprData, copyData, clinicalFeaturesData)
#
# predicting method for the model
#   exprData: An ExpressionSet (as in Biobase package) of gene expression data
```

```
    #   copyData: An ExpressionSet of copy number variation data

    #   clinicalFeaturesData: Data frame with clinical information

    #

    # ============================

    customPredict = function(exprData, copyData, clinicalFeaturesData){



      # making predictions using OS-based model

      p1 <- .self$model$osmodel$customPredict(exprData, copyData,
clinicalFeaturesData)

      # making predictions using DS-based model

      p2 <- .self$model$dsmodel$customPredict(exprData, copyData,
clinicalFeaturesData)



      # no need for CNV data and expression data anymore, delete them

      rm(exprData)

      rm(copyData)



      .self$predictions <- rbind(p1, p2)



      # combining predictions using previously trained weights

      p <- p1 + .self$w[2] * p2

      return (p)

    }

))
```

2. syn1417992_fix.R

```
require(rms)

require(MASS)

require(survival)

require(predictiveModeling)

require(gbm)

require(caret)

require(randomSurvivalForest)

require(devtools)


# install the latest version of DreamBox7 package from github

install_github(repo="DreamBox7", username="weiyi-bitw", ref="master")

library(DreamBox7)


setRefClass(Class = "PredictiveModel")


#' GoldiloxModel

#'

#' Modified from DemoClinicalOnlyModel from BCC challenge. This meta-model trains the same model for both OS and DS

#' survival, and combine two predictions by taking weighted average where the weights were chosed to maximize

#' the performance on training set.

#'
```

```
#' For details of functions, see the DreamBox7 package at:
#'   https://github.com/weiyi-bitw/DreamBox7
#' The source code is available at:
#'   https://github.com/weiyi-bitw/BCCModels
#'
#' @author Wei-Yi Cheng
#' @export


GoldiModel <- setRefClass(
  Class  = "GoldiModel",
  contains = "PredictiveModel",
  fields   = c("model", "attractome", "annot", "predictions", "chosenProbes"),
  methods  = list(
    # ============================
    #
    # initialize(...)
    #
    # function called when the model is initialized
    #
    # ============================
    initialize = function(...){
      # load the gene member list of each attractor metagene
      data(attractome.minimalist)
```

```
    .self$attractome = attractome.minimalist
    # load the annotation file of Illumina HT12v3 and v4
    data(map)
    .self$annot = map
    return(.self)
  },
  # ============================
  #
  # customTrain(exprData, copyData, clinicalFeaturesData,
  #             clinicalSurvData, clinicalSurvData_DS, ...)
  #
  # training method for the model
  #   exprData: An ExpressionSet (as in Biobase package) of gene expression data
  #   copyData: An ExpressionSet of copy number variation data
  #   clinicalFeaturesData: Data frame with clinical information
  #   clinicalSurvData: a Surv object with overall survival information
  #
  # ============================
  customTrain = function(exprData, copyData,
clinicalFeaturesData,clinicalSurvData,...){
    if(class(clinicalSurvData) != "Surv"){
      stop("Expecting 'responseData' object of type 'Surv'")
    }
```

```
# no need for cnv data, remove it
rm(copyData)
# impute missing numerical clinical information by mean
clnc <- lazyImputeDFClncOslo(clinicalFeaturesData)
# binarize categorical variables
clinical <- expandClncOslo(clnc)
# remove chemotherapy and hormonal therapy to fit oslo
clinical$tr.CT <- NULL; clinical$tr.HT <- NULL

cat("Create metagene space...");flush.console()
# summarize gene expression into metagene expression
o <- CreateMetageneSpace(exprs(exprData), .self$attractome, .self$annot)
meta <- o$metaSpace
# store the used probes
.self$chosenProbes <- o$pbs

# save LYM values for conditioning
ls <- meta["ls",]
# median center metagene expression
meta <- t(apply(meta, 1, function(x){ x - median(x)}))
# conditioning MES by lymph node status and tumor size
idx <- (clinical[,"lymph_nodes_positive"]<1 & clinical[,"size"] < 30)
mes.lymphneg <- meta["mt",] * idx
```

```
mes.lymphneg[idx] <- mes.lymphneg[idx] - median(mes.lymphneg[idx])
meta <- rbind(meta, mes.lymphneg)

# conditioning LYM by lymph node number
idx <- (clinical[,"lymph_nodes_positive"] > 3)
ls.lymphpos <- ls * idx
ls.lymphpos[idx] <- ls.lymphpos[idx] - median(ls.lymphpos[idx])
meta <- rbind(meta, ls.lymphpos)

# conditioning LYM by ER and HER2 attractor expression
idx <- (meta["er",] < 0 & meta["erbb2", ] < 0)
ls.erneg <- ls * idx
ls.erneg[idx] <- ls.erneg[idx] - median(ls.erneg[idx])
meta <- rbind(meta, ls.erneg)

# some other empirically chosen features, not really useful
lym.N <- factor(clnc$lymph_nodes_positive < 1)
lymph <- clinical[,"lymph_nodes_positive"]
lymph <- sapply(lymph, function(x){min(x, 7)})
lsxlymph <- ls * (7 - lymph)

gpr4 <- exprs(exprData)["ILMN_2074477",]
```

```
    # finish with expression data, remove it

    rm(exprData)

    cat("done!\n");flush.console()


#===== 1. clinical Cox-AIC model =====

    cat("1.  Training Cox-AIC model using clinical features...");flush.console()

    X <- clinical

    upper <- terms(clinicalSurvData~(.), data = X)

    cm <- step(coxph(clinicalSurvData~1, data=X),scope=upper, direction="both", k=2,
trace=FALSE)

    cat("done!\n");flush.console()

#===== 2. clinical GBM model =====

    cat("2.  Training gbm model...");flush.console()

    X <- X[, attr(cm$term, "term.labels")]

    cgbm <- gbm.cvrun(clinicalSurvData~., data=X ,distribution="coxph",
shrinkage=0.002, n.trees=1500, interaction.depth=8, cv.folds=5, verbose=F, seed=53)

    cat("done!\n");flush.console()

#===== 3. molecular Cox-AIC model =======

    cat("3.  Training Cox-AIC model using metagenes ...");flush.console()

    X <- data.frame(t(meta))

    upper <- terms(clinicalSurvData~(.), data = X)

    coxmodel <- step(coxph(clinicalSurvData~., data=X), scope=upper, direction="both",
k=2, trace=FALSE)

    cat("done!\n");flush.console()
```

```
#===== 4. GBM model =========

    cat("4.  Training gbm model...");flush.console()

    X <- data.frame(t(meta))

    gbmmodel <- gbm.cvrun(clinicalSurvData~.,data=X, distribution="coxph",
shrinkage=0.002, n.trees=1500, interaction.depth=6, cv.folds=5, verbose=F, seed=913) #
my bday ;D

    cat("done!\n");flush.console()

#===== 5. KNN model ======

    cat("5.  Creating KNN database ...");flush.console()

    t <- clinicalSurvData[,1]

    defSurvSamples <- which(clinicalSurvData[,2]==1 | clinicalSurvData[,1] > 365 * 10)

    ccdi <- getAllCCDIWz(meta, clinicalSurvData)

    idx <- ccdi[c("er", "mitotic", "puf60", "erbb2", "chr7p11.2", "ls")]



    knnmodel <- list()

    knnmodel$x.train <- list(meta=meta[names(idx), defSurvSamples],
time=t[defSurvSamples], concordance=idx)

    knnmodel$c.train <- preproClncKNN(clnc, clinicalSurvData, ccdi.upper=0.6,
ccdi.lower=0.4)

    cat("done!\n");flush.console()



#===== 6. Training Cox model using empirically selected features =======

    cat("6.  Cox regression using empirically selected features...");flush.console()

    X <- data.frame( cbind(meta["mitotic",], meta["ls.erneg",],
clinical$lymph_nodes_positive, meta["mes.lymphneg",], meta["susd3",],
clinical$age_at_diagnosis) )
```

```
    colnames(X) <- c("CIN", "LYM_ERNeg", "lymNum", "MES_lymNumNeg",
"SUSD3", "age")

    cox.a <- coxph(clinicalSurvData~., data=X)

    cat("done!\n");



#===== 7. Training GBM model using empirically selected features =========

    cat("7.  Training gbm model...");flush.console()

    gbm.a <- gbm.cvrun(clinicalSurvData~., data=X ,distribution="coxph",
shrinkage=0.002, n.trees=1500, interaction.depth=6, cv.folds=5, verbose=F, seed=913)
#seed = my birthday :D !

    cat("done!\n");flush.console()



#===== 8. Training Cox model using another set of empirically selected features
=======

    cat("8.  Cox regression on empirically selected features B...");flush.console()

    X <- data.frame( cbind(meta["mitotic",], meta["mes.lymphneg",], lsxlymph,
clinical$size, clinical$h.IDCnMED, gpr4) )

    colnames(X) <- c("CIN", "MES_lymNumNeg", "LYMxlymNum", "size", "MED",
"GPR4_g" )

    cox.b <- coxph(clinicalSurvData~., data=X)

    cat("done!\n");flush.console()



    .self$model <- list(

        cm=cm,

        cgbm=cgbm,

        coxmodel=coxmodel,
```

```
        gbmmodel=gbmmodel,
        knnmodel=knnmodel,
        cox.a=cox.a,
        gbm.a=gbm.a,
        cox.b=cox.b
    )
  },
  # ============================
  #
  # customPredict(exprData, copyData, clinicalFeaturesData)
  #
  # predicting method for the model
  #   exprData: An ExpressionSet (as in Biobase package) of gene expression data
  #   copyData: An ExpressionSet of copy number variation data
  #   clinicalFeaturesData: Data frame with clinical information
  #
  # ============================
  customPredict = function(exprData, copyData, clinicalFeaturesData){
  #===== Preprocessing data the same way as training, see customTrain for line-by-line
comment
    rm(copyData)

    clnc <- lazyImputeDFClncOslo(clinicalFeaturesData)
```

```
    clinical <- expandClncOslo(clnc)

    clinical$tr.CT <- NULL; clinical$tr.HT <- NULL


    cat("Create metagene space...");flush.console()

    meta <- CreateMetageneSpace(exprs(exprData), .self$attractome,
.self$annot)$metaSpace

    ls <- meta["ls",]

    meta <- t(apply(meta, 1, function(x){ x - median(x)}))


    idx <- (clinical[,"lymph_nodes_positive"]<1 & clinical[,"size"] < 30)

    mes.lymphneg <- meta["mt",] * idx

    mes.lymphneg[idx] <- mes.lymphneg[idx] - median(mes.lymphneg[idx])

    meta <- rbind(meta, mes.lymphneg)


    idx <- (clinical[,"lymph_nodes_positive"] > 3)

    ls.lymphpos <- ls * idx

    ls.lymphpos[idx] <- ls.lymphpos[idx] - median(ls.lymphpos[idx])

    meta <- rbind(meta, ls.lymphpos)


    idx <- (meta["er",] < 0 & meta["erbb2", ] < 0)

    ls.erneg <- ls * idx

    ls.erneg[idx] <- ls.erneg[idx] - median(ls.erneg[idx])

    meta <- rbind(meta, ls.erneg)
```

```
    lym.N <- factor(clnc$lymph_nodes_positive < 1)
    lymph <- clinical[,"lymph_nodes_positive"]
    lymph <- sapply(lymph, function(x){min(x, 7)})
    lsxlymph <- ls * (7 - lymph)

    gpr4 <- exprs(exprData)["ILMN_2074477",]
    rm(exprData)

    cat("done!\n");flush.console()

    p <- matrix(NA,nrow=length(.self$model), ncol=ncol(meta))
#===== 1. Predict using clinical Cox-AIC =====
    cat("1.  Predicting using clinical Cox-AIC model...");flush.console()
    X <- clinical
    p[1,] <- predict(.self$model$cm, X)
    cat("done!\n");flush.console()
#===== 2. Predict using clinical GBM =====
    cat("2.  Predicting using clinical GBM model...");flush.console()
    X <- X[, attr(.self$model$cm$term, "term.labels")]
    best.iter <- gbm.perf(.self$model$cgbm, method="cv", plot.it=FALSE)
    cat("Best iter: ", best.iter, "\n", sep="");flush.console()
    p[2,] <- predict.gbm(.self$model$cgbm, X, best.iter)
```

```
    cat("done!\n");flush.console()

#===== 3. Predict using molecular Cox-AIC model =====

    cat("3.  Predicting using molecular Cox-AIC model...");flush.console()

    X <- data.frame(t(meta))

    p[3,] <- predict(.self$model$coxmodel, X)

    cat("done!\n");flush.console()

#===== 4. Predict using molecular GBM model =====

    cat("4.  Predicting using gbm model...");flush.console()

    X <- data.frame(t(meta))

    best.iter <- gbm.perf(.self$model$gbmmodel, method="cv", plot.it=FALSE)

    cat("Best iter: ", best.iter, "\n", sep="");flush.console()

    p[4,] <- predict.gbm(.self$model$gbmmodel, X, best.iter)

    cat("done!\n");flush.console()

#===== 5. Predict using KNN model =====

    cat("5.  Predicting using KNN model ...");flush.console()

    knnmodel <- .self$model$knnmodel

    qX <- meta[names(knnmodel$x.train$concordance),]

    qC <- clnc[,names(knnmodel$c.train$distWeight)]

    qC <- t(preproClncKNN(qC, isFactorIn=knnmodel$c.train$isFactor,
dwIn=knnmodel$c.train$distWeight)$clinical)

    wvec <- c(abs(knnmodel$x.train$concordance-0.5),
abs(knnmodel$c.train$concordance-0.5))

    qAll <- rbind(qX, qC)

    trainDB <- rbind(knnmodel$x.train$meta, t(knnmodel$c.train$clinical))
```

```
    trainTime <- knnmodel$x.train$time

    out <- ewknn.predict(trainDB, trainTime, qAll, wvec, k=floor(0.1*ncol(trainDB)))

    p[5,] <- 365/out



    cat("done!\n");flush.console()

#===== 6. Predict using Cox model on empirically selected features =====

    cat("6.  Predicting using Cox model trained on ES features ...");flush.console()

    X <- data.frame( cbind(meta["mitotic",], meta["ls.erneg",],
clinical$lymph_nodes_positive, meta["mes.lymphneg",], meta["susd3",],
clinical$age_at_diagnosis) )

    colnames(X) <- c("CIN", "LYM_ERNeg", "lymNum", "MES_lymNumNeg",
"SUSD3", "age")

    p[6,] <- predict(.self$model$cox.a, X)

    cat("done!\n");flush.console()

#===== 7. Predict using gbm model on ES features =====

    cat("7.  Predicting using gbm model trained on ES features...");flush.console()

    best.iter <- gbm.perf(.self$model$gbm.a, method="cv", plot.it=FALSE)

    cat("Best iter: ", best.iter, "\n", sep="");flush.console()

    p[7,] <- predict.gbm(.self$model$gbm.a, X, best.iter)

    cat("done!\n");flush.console()

#===== 8. Predict using Cox model on ES features B=====

    cat("8. Predicting using disease specific Minimalist model ...");flush.console()

    X <- data.frame( cbind(meta["mitotic",], meta["mes.lymphneg",], lsxlymph,
clinical$size, clinical$h.IDCnMED, gpr4) )
```

```
      colnames(X) <- c("CIN", "MES_lymNumNeg", "LYMxlymNum", "size", "MED",
"GPR4_g" )

      p[8,] <- predict(.self$model[[8]], X)

      cat("done!\n");flush.console()


#===== Combining predictions =====

      .self$predictions <- p

      pout <- matrix(NA, nrow=2, ncol=ncol(p))

      pout[1,] <- apply(p[c(1:5, 8),], 2, mean)

      pout[2,] <- apply(p[6:7,], 2, mean)

      pz <- apply(pout, 1, scale)

      p <- apply(pz, 1, mean)

      return (p)

    }

))
```