Penalty function learning

Toby Dylan Hocking

November 1, 2024

1 Introduction

In supervised changepoint detection, we are given n labeled data sequences. For each data sequence $i \in \{1, \dots, n\}$ we have a vector of noisy data $\mathbf{z}_i \in \mathbb{R}^{d_i}$, a set of labels L_i which indicates appropriate changepoint positions, and an input/feature $\mathbf{x}_i \in \mathbb{R}^p$. Note that the number of data points to segment d_i can be different for every data sequence i, but the number of features p is constant.

The optimal changepoint model for data sequence i, and a penalty parameter λ, is given by

$$\mathbf{m}_i^\lambda = \arg\min_{\mathbf{m} \in \mathbb{R}^{d_i}} \ell(\mathbf{m}, \mathbf{z}_i) + \lambda \mathcal{C}(\mathbf{m}), \tag{1}$$

where

- ℓ is a loss function, e.g. the square loss $\ell(\mathbf{m}, \mathbf{z}_i) = \sum_{j=1}^{d_i} (m_j - z_{ij})^2$,
- $\mathcal{C}$ is a model complexity function, e.g. the number of changes $\mathcal{C}(\mathbf{m}) = \sum_{j=1}^{d_i - 1} I(m_j \neq m_{j+1})$,
- $\lambda \geq 0$ is a penalty constant. Larger values penalize more complex models, and result in few changepoints ($\lambda = \infty$ means no changes). Smaller values penalize less and result in more changepoints ($\lambda = 0$ means $d_i - 1$ changes).

In this context, we would like to learn a different penalty constant $\log \lambda_i = f(\mathbf{x}_i)$ for every data sequence, where $f : \mathbb{R}^p \to \mathbb{R}$ is a function that we will learn by minimizing the number of incorrect labels in the training data:

$$\min_f \sum_{i=1}^n e[\mathbf{m}_i^{\exp f(\mathbf{x}_i)}, L_i]. \tag{2}$$

The function e is the number of labels in L_i which are incorrectly predicted by the changepoint model $\mathbf{m}_i^{\exp f(x_i)}$.

After having learned f on training data, it can be used to predict a changepoint model for a test data sequence $\mathbf{z} \in \mathbb{R}^d$ which has features $\mathbf{x} \in \mathbb{R}^p$. First compute a predicted penalty value $\lambda = \exp f(\mathbf{x})$, and then use it to compute a predicted segmentation model $\mathbf{m}^\lambda$.

2 Details

2.1 Changepoint model fitting

For each of several labeled segmentation problems (data sequences that are separate but have a similar signal/noise pattern), use your favorite changepoint detection package to compute a sequence of models of increasing complexity (say from 0 to 20 changepoints). In contrast to unsupervised changepoint detection (where we usually compute just one changepoint model per data sequence), it is essential to compute several models in supervised changepoint detection (so that we can learn which models and penalty values result in changepoints with minimum error with repect to the labels). Below we use the Segmentor function to compute a maximum likelihood Gaussian model.

```
> library(data.table)
> data(neuroblastoma, package="neuroblastoma")
> ids.str <- paste(c(1, 4, 6, 8, 10, 11))
> someProfiles <- function(all.profiles){
+   data.table(all.profiles)[profile.id %in% ids.str, ]
+ }
> profiles <- someProfiles(neuroblastoma$profiles)
> labels <- someProfiles(neuroblastoma$annotations)
> problem.list <- split(profiles, profiles[, paste(profile.id, chromosome)])
> segs.list <- list()
> loss.list <- list()
> for(problem.i in seq_along(problem.list)){
+   problem.name <- names(problem.list)[[problem.i]]
+   pro <- problem.list[[problem.name]]
+   meta <- pro[1, .(profile.id, chromosome)]
+   max.segments <- min(nrow(pro), 10)
+   fit <- jointseg::Fpsn(pro$logratio, max.segments)
+   break.mat <- fit$t.est
+   rss.vec <- rep(NA, max.segments)
+   for(n.segments in 1:max.segments){
+     end <- break.mat[n.segments, 1:n.segments]
+     data.before.change <- end[-n.segments]
+     data.after.change <- data.before.change+1
+     pos.before.change <- as.integer(
+       (pro$position[data.before.change]+pro$position[data.after.change])/2)
+     start <- c(1, data.after.change)
+     chromStart <- c(pro$position[1], pos.before.change)
+     chromEnd <- c(pos.before.change, max(pro$position))
+     seg.mean.vec <- sapply(seq_along(start), function(i){
+       indices <- start[i]:end[i]
+       mean(pro$logratio[indices])
+     })
+     data.mean.vec <- rep(seg.mean.vec, end-start+1)
```

```
+     residual.vec <- pro$logratio - data.mean.vec
+     rss.vec[n.segments] <- sum(residual.vec * residual.vec)
+     segs.list[[paste(problem.name, n.segments)]] <- data.table(
+       meta,
+       n.segments,
+       start,
+       end,
+       chromStart,
+       chromEnd,
+       mean=seg.mean.vec)
+   }
+   loss.list[[paste(problem.name, n.segments)]] <- data.table(
+     meta,
+     n.segments=1:max.segments,
+     loss=rss.vec)
+ }
> loss <- do.call(rbind, loss.list)
> segs <- do.call(rbind, segs.list)
>
```

Note that since we have saved the segment starts and ends, we can easily derive the changepoint positions of each model:

```
> print(segs)
```

```
      profile.id chromosome n.segments start   end chromStart  chromEnd
          <fctr>     <fctr>      <int> <num> <num>      <int>     <int>
   1:          1          1          1     1   474     809681 249063592
   2:          1          1          2     1   438     809681 212809180
   3:          1          1          2   439   474  212809180 249063592
   4:          1          1          3     1   437     809681 212280934
   5:          1          1          3   438   460  212280934 234068672
  ---
7871:          8          Y          3     4     4   16036249  25746303
7872:          8          Y          4     1     1     293556   1443054
7873:          8          Y          4     2     2    1443054   4459374
7874:          8          Y          4     3     3    4459374  16036249
7875:          8          Y          4     4     4   16036249  25746303
             mean
            <num>
   1:  0.31345083
   2:  0.35186515
   3: -0.15392338
   4:  0.35242544
   5:  0.02954558
  ---
7871: -1.51045706
```

```
7872: -0.09080294
7873: -0.60384051
7874: -2.25842515
7875: -1.51045706
```

Also note that we have saved the loss and model complexity (n.segments) of each model,

```
> print(loss)
```

```
      profile.id chromosome n.segments       loss
          <fctr>     <fctr>      <int>      <num>
   1:          1          1          1 15.9149875
   2:          1          1          2  7.4048569
   3:          1          1          3  5.5191996
   4:          1          1          4  4.3030047
   5:          1          1          5  4.0235352
  ---
1430:          8          X         10  0.2985018
1431:          8          Y          1  2.7740679
1432:          8          Y          2  0.4113319
1433:          8          Y          3  0.1316038
1434:          8          Y          4  0.0000000
```

2.2 Model selection functions

We use penaltyLearning::modelSelection to compute the exact path of models that will be selected for every possible non-negative penalty value,

```
> selection <- loss[, {
+   penaltyLearning::modelSelection(.SD, "loss", "n.segments")
+ }, by=list(profile.id, chromosome)]
> print(selection[profile.id==1 & chromosome==1])
```

```
   profile.id chromosome min.lambda max.lambda min.log.lambda max.log.lambda
       <fctr>     <fctr>      <num>      <num>          <num>          <num>
1:          1          1 0.00000000 0.07175223           -Inf     -2.6345364
2:          1          1 0.07175223 0.12470984     -2.6345364     -2.0817655
3:          1          1 0.12470984 0.12800334     -2.0817655     -2.0556989
4:          1          1 0.12800334 0.20965938     -2.0556989     -1.5622711
5:          1          1 0.20965938 0.27946950     -1.5622711     -1.2748621
6:          1          1 0.27946950 1.21619490     -1.2748621      0.1957271
7:          1          1 1.21619490 1.88565729      0.1957271      0.6342765
8:          1          1 1.88565729 8.51013055      0.6342765      2.1412573
9:          1          1 8.51013055        Inf      2.1412573            Inf
   cum.iterations n.segments      loss
           <int>      <int>     <num>
```

```
1:             10         10  3.361407
2:              9          9  3.433159
3:              8          8  3.557869
4:              5          6  3.813876
5:              4          5  4.023535
6:              3          4  4.303005
7:              2          3  5.519200
8:              1          2  7.404857
9:              0          1 15.914987

>
```

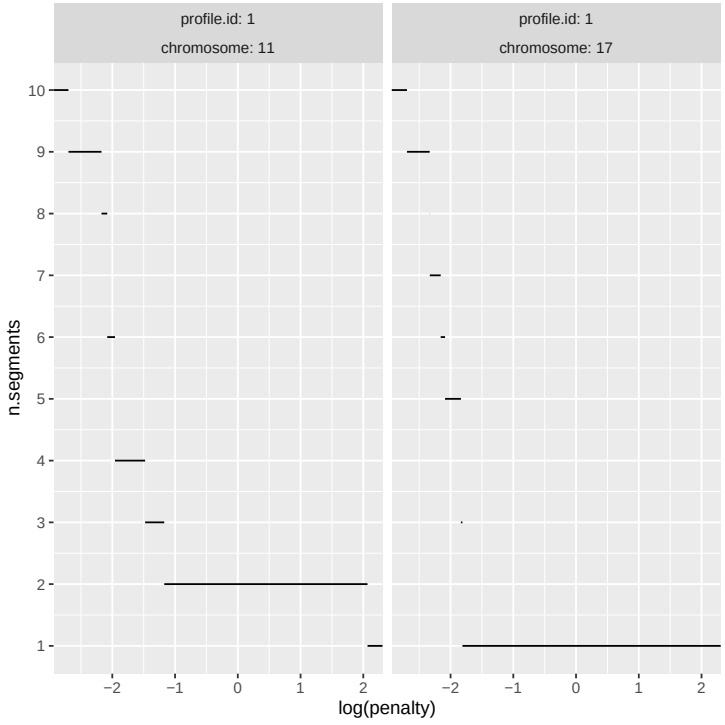
Note that the model selection function $s_i(\lambda)$ is the number of segments that is selected for a given penalty λ and data sequence i,

$$s_i(\lambda) = \arg\min_s \ell(\mathbf{m}, \mathbf{z}_i) + \lambda \mathcal{C}(\mathbf{m}). \tag{3}$$

These functions are piecewise constant, and specific to each data sequence:

```
> some.selection <- selection[profile.id==1 & chromosome %in% c(11, 17)]
> some.selection[, list(
+   pid.chr=paste(profile.id, chromosome),
+   min.log.lambda, max.log.lambda, n.segments, loss
+ )]

    pid.chr min.log.lambda max.log.lambda n.segments      loss
     <char>          <num>          <num>      <int>     <num>
 1:    1 11           -Inf      -2.698480         10  1.514946
 2:    1 11      -2.698480      -2.172301          9  1.582253
 3:    1 11      -2.172301      -2.080750          8  1.696169
 4:    1 11      -2.080750      -1.957421          6  1.945842
 5:    1 11      -1.957421      -1.477048          4  2.228286
 6:    1 11      -1.477048      -1.172320          3  2.456597
 7:    1 11      -1.172320       2.067398          2  2.766245
 8:    1 11       2.067398            Inf          1 10.670475
 9:    1 17           -Inf      -2.694380         10  1.398609
10:    1 17      -2.694380      -2.332349          9  1.466194
11:    1 17      -2.332349      -2.330538          8  1.563261
12:    1 17      -2.330538      -2.157175          7  1.660504
13:    1 17      -2.157175      -2.087849          6  1.776156
14:    1 17      -2.087849      -1.834952          5  1.900109
15:    1 17      -1.834952      -1.810653          3  2.219352
16:    1 17      -1.810653            Inf          1  2.546446

> library(ggplot2)
> gg.selection <- ggplot()+
+   theme_grey()+
```

```
+   facet_grid(. ~ profile.id + chromosome, labeller=label_both)+
+   geom_segment(aes(
+     min.log.lambda, n.segments,
+     xend=max.log.lambda, yend=n.segments),
+     data=some.selection)+
+   scale_y_continuous(breaks=1:max(some.selection$n.segments))+
+   xlab("log(penalty)")
> print(gg.selection)
>
```



2.3 Label error

To compute the label error function $E_i(\lambda)$ (number of incorrect labels for data sequence i as a function of penalty λ), we need to first get the positions of predicted changepoints:

```
> changes <- segs[1 < start, ]
> changes[, list(profile.id, chromosome, n.segments, changepoint=chromStart)]
```

```
   profile.id chromosome n.segments changepoint
       <fctr>     <fctr>      <int>       <int>
1:          1          1          2   212809180
2:          1          1          3   212280934
```

```
   3:          1          1          3   234068672
   4:          1          1          4    40348010
   5:          1          1          4   212280934
  ---
6437:          8          Y          3     4459374
6438:          8          Y          3    16036249
6439:          8          Y          4     1443054
6440:          8          Y          4     4459374
6441:          8          Y          4    16036249

>
```

Then, we use penaltyLearning::labelError to compute the number of incorrect labels for each model, for each labeled data sequence. The named arguments specify how the three input data tables are used:

change.var is the column name of changes that will be used as the changepoint position.

label.vars are the column names of the start and end of the labeled regions.

problem.vars are the column names (common to all data tables) that are used to identify independent data sequences.

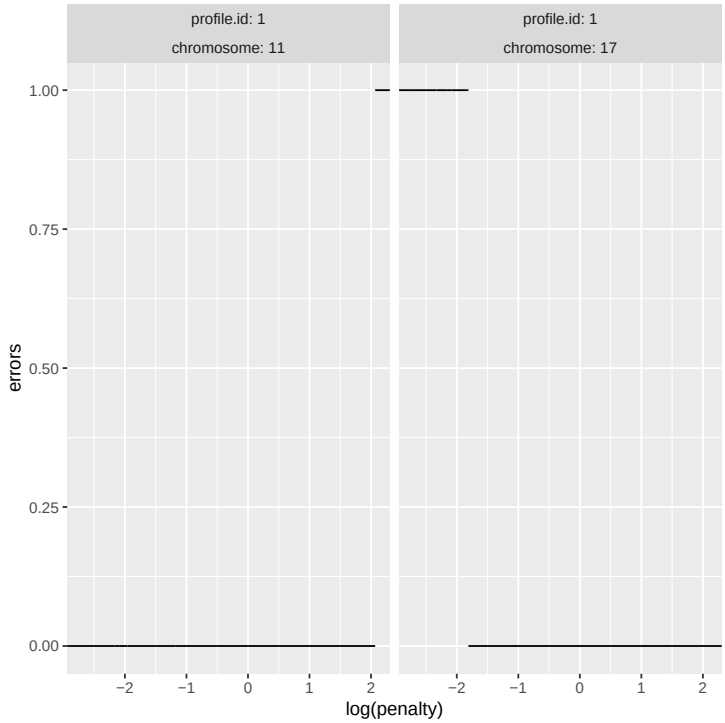
```
> errors <- penaltyLearning::labelError(
+   selection, labels, changes,
+   change.var="chromStart",
+   label.vars=c("min", "max"),
+   problem.vars=c("profile.id", "chromosome"))
>
```

The labelError function returns a list of two data tables: label.errors has one row for each label and each model, and model.errors has one row for each data sequence and each model. In this case they are the same size because the neuroblastoma data set has only one label per data sequence. The model.errors $E_i(\lambda)$ is a piecewise constant function for every data sequence i (the number of incorrect labels as a function of penalty λ):

$$E_i(\lambda) = e[\mathbf{m}_i^\lambda, L_i]. \tag{4}$$

```
> some.errors <- errors$model.errors[some.selection, on=list(
+   profile.id, chromosome, n.segments)]
> gg.err <- ggplot()+
+   theme_grey()+
+   facet_grid(. ~ profile.id + chromosome, labeller=label_both)+
+   geom_segment(aes(
+     min.log.lambda, errors,
+     xend=max.log.lambda, yend=errors),
```

```
+     data=some.errors)+
+   xlab("log(penalty)")
> print(gg.err)
>
```



2.4 Target intervals

Now, let's perform a computational cross-validation experiment to train and evaluate a learned penalty function. We use labels in chromosome 11 as a test set, and use all other labels as a train set.

```
> all.errors <- data.table(errors$model.errors)
> all.errors[, set := ifelse(chromosome=="11", "test", "train")]
```

```
     profile.id chromosome min.lambda max.lambda min.log.lambda max.log.lambda
         <fctr>     <fctr>      <num>      <num>          <num>          <num>
  1:          1          1 0.00000000 0.07175223           -Inf     -2.6345364
  2:          1          1 0.07175223 0.12470984     -2.6345364     -2.0817655
  3:          1          1 0.12470984 0.12800334     -2.0817655     -2.0556989
  4:          1          1 0.12800334 0.20965938     -2.0556989     -1.5622711
  5:          1          1 0.20965938 0.27946950     -1.5622711     -1.2748621
 ---
252:         11         17 0.34732889 0.46128451     -1.0574831     -0.7737403
```

```
253:         11         17 0.46128451 0.61549695     -0.7737403     -0.4853253
254:         11         17 0.61549695 0.72402388     -0.4853253     -0.3229309
255:         11         17 0.72402388 2.82055129     -0.3229309      1.0369324
256:         11         17 2.82055129        Inf      1.0369324            Inf
     cum.iterations n.segments      loss possible.fp    fp possible.fn    fn
              <int>      <int>     <num>       <num> <num>       <num> <num>
  1:             10         10  3.361407           1     1           0     0
  2:              9          9  3.433159           1     1           0     0
  3:              8          8  3.557869           1     1           0     0
  4:              5          6  3.813876           1     1           0     0
  5:              4          5  4.023535           1     1           0     0
 ---
252:              8          7  5.021862           1     1           0     0
253:              7          6  5.483146           1     1           0     0
254:              6          5  6.098643           1     1           0     0
255:              1          2  8.270715           1     1           0     0
256:              0          1 11.091266           1     0           0     0
     labels errors    set
      <num>  <num> <char>
  1:      1      1  train
  2:      1      1  train
  3:      1      1  train
  4:      1      1  train
  5:      1      1  train
 ---
252:      1      1  train
253:      1      1  train
254:      1      1  train
255:      1      1  train
256:      1      0  train

>
```

To train a penalty learning model, we compute target intervals of log(penalty) values that achieve the minimal number of incorrect labels (for each problem independently). This is the "output" in the machine learning problem, which is a (n x 2) matrix that we compute using the penaltyLearning::targetIntervals function.

```
> target.dt <- penaltyLearning::targetIntervals(
+   all.errors[set=="train", ],
+   c("profile.id", "chromosome"))
> target.mat <- target.dt[, cbind(min.log.lambda, max.log.lambda)]
> rownames(target.mat) <- target.dt[, paste(profile.id, chromosome)]
> print(head(target.mat))

     min.log.lambda max.log.lambda
1 1      0.1957271            Inf
```

```
1 2      -1.1087554            Inf
1 3      -1.0006564            Inf
1 4      -0.9951232            Inf
1 17     -1.8106532            Inf
4 1            -Inf       3.265185

>
```

Note that the first column is the lower limit (possibly -Inf) of acceptable log(penalty) values, and the second column is the upper limit (possibly Inf).

2.5 Feature matrix

Then we compute a feature matrix (problems x features), which is the "input" in the machine learning problem. Here we compute a simple matrix with just 2 columns/features (log.var $= \log \sigma_i$ is a variance estimate, and log.data $= \log d_i$ is log of the number of data points to segment).

```
> feature.dt <- profiles[, list(
+   log.data=log(.N),
+   log.var=log(median(abs(diff(logratio))))
+ ), by=list(profile.id, chromosome)]
> all.feature.mat <- feature.dt[, cbind(log.data, log.var)]
> rownames(all.feature.mat) <- feature.dt[, paste(profile.id, chromosome)]
> train.feature.mat <- all.feature.mat[rownames(target.mat), ]
> print(head(train.feature.mat))

     log.data   log.var
1 1  6.161207 -2.654569
1 2  5.521461 -2.421264
1 3  5.252273 -2.070554
1 4  5.036953 -2.412361
1 17 5.141664 -2.509952
4 1  6.059123 -2.369222

>
```

You can also use penaltyLearning::featureMatrix for computing a larger feature matrix in real data sets where you aren't sure what features are relevant.

2.6 Learn a penalty function

Then we use penaltyLearning::IntervalRegressionUnregularized to learn a penalty function for this small data set (see Hocking et al ICML'13 for details). Because we only computed two features $\log d_i, \log \sigma_i$, the function that we learn here is $f(x_i) = \beta + w_1 \log d_i + w_2 \log \sigma_i$ (we learn the intercept/bias β and feature weights w_1, w_2). Remember that the prediction function models log(penalty) values $f(x_i) = \log \lambda_i$, so the penalty function we learn here is $\lambda_i = e^{\beta} d_i^{w_1} \sigma_i^{w_2}$.

```
> fit <- penaltyLearning::IntervalRegressionUnregularized(
+   train.feature.mat, target.mat)
> print(fit)

IntervalRegression model for margin=1 regularization=0 with weights:
 (Intercept)  log.data log.var
    1.579128 0.6612357 1.78769
```

For data sets with more labels and features, penaltyLearning:;IntervalRegressionCV would be preferable, since it also performs variable selection using L1-regularization.

2.7 Prediction

Then we use the model to predict log(penalty) values for each segmentation problem (even those in the test set).

```
> feature.dt[, pred.log.lambda := predict(fit, all.feature.mat)]

     profile.id chromosome log.data   log.var pred.log.lambda
         <fctr>     <fctr>    <num>     <num>           <num>
  1:          8          1 6.013715 -2.877904      0.41081127
  2:          8          2 5.375278 -2.928180     -0.10122396
  3:          8          3 5.081404 -2.917816     -0.27701599
  4:          8          4 4.882802 -3.045320     -0.63627765
  5:          8          5 5.164786 -2.833040     -0.07032902
 ---
140:         10         20 4.828314 -2.882404     -0.38106243
141:         10         21 4.110874 -2.239610      0.29365581
142:         10         22 4.543295 -2.847312     -0.50679475
143:         10          X 5.231109 -2.847312     -0.05198766
144:         10          Y 3.295837 -2.900422     -1.42660240
```

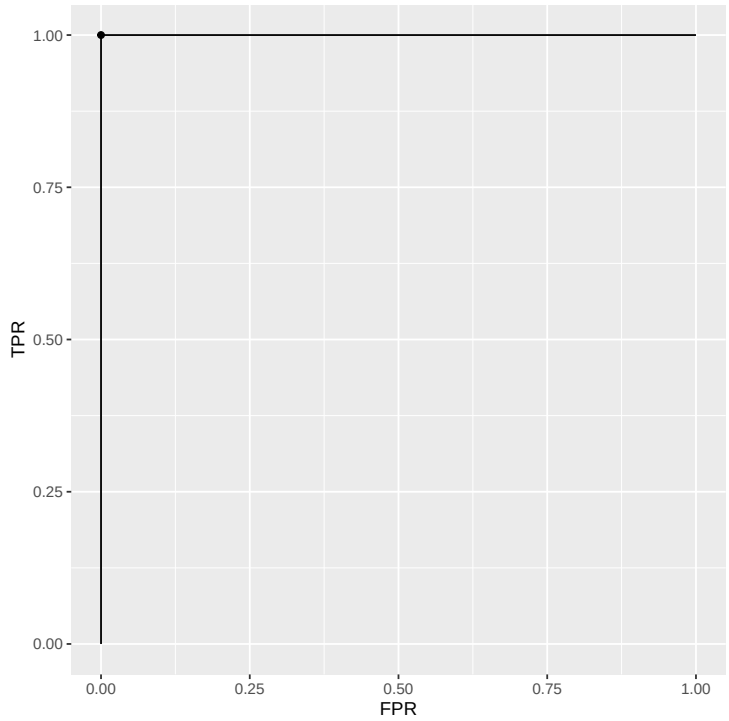
Note that this is the predict method for the IntervalRegression class.

2.8 Evaluation

We can use penaltyLearning::ROChange to compute test ROC curves, and Area Under the Curve (AUC), which in this case is 1, indicating a function which can perfectly discriminate between positive and negatve labels in the test set.

```
> test.pred <- feature.dt[chromosome=="11",]
> roc <- penaltyLearning::ROChange(
+   all.errors, test.pred, c("profile.id", "chromosome"))
> pred.thresh <- roc$thresholds[threshold=="predicted"]
> gg.roc <- ggplot()+
+   geom_path(aes(
+     FPR, TPR),
```

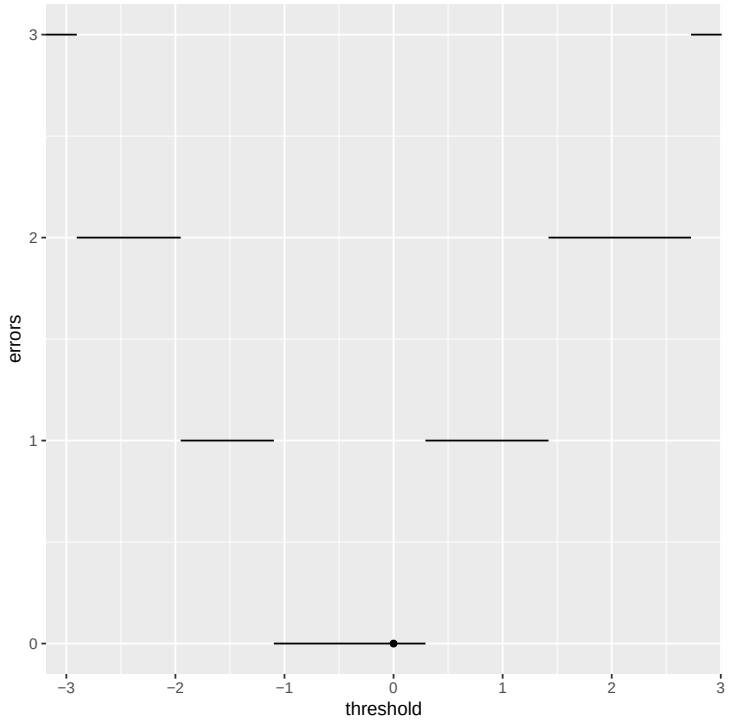
```
+     data=roc$roc)+
+   geom_point(aes(
+     FPR, TPR),
+     data=pred.thresh)
> print(gg.roc)
> print(pred.thresh)

   threshold labels possible.fp possible.fn min.thresh max.thresh    fp    fn
      <char>  <num>       <num>       <num>      <num>      <num> <num> <num>
1: predicted      6           3           3  -1.097374  0.2923082     0     0
   min.fp.fn errors   FPR    tp   TPR error.percent
       <num>  <num> <num> <num> <num>         <num>
1:         0      0     0     3     1             0

>
```



Using the ROC output you can also easily plot the number of incorrect labels as a function of threshold $\tau \in \mathbb{R}$ added to the predicted log(penalty) function $f(x_i) + \tau$,

```
> gg.thresh <- ggplot()+
+   geom_segment(aes(
+     min.thresh, errors,
+     xend=max.thresh, yend=errors),
```

```
+     data=roc$roc)+
+   geom_point(aes(
+     0, errors),
+     data=pred.thresh)+
+   xlab("threshold")
> print(gg.thresh)
>
```



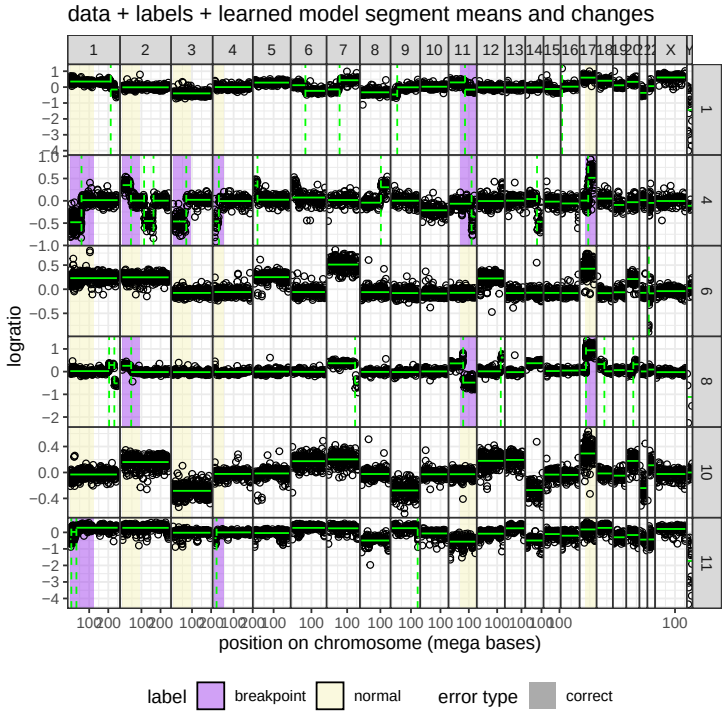
We draw a dot at $\tau = 0$ to show the number of incorrectly predicted test labels of the learned prediction function (0 in this case).

2.9 Visualizing predictions

Finally we use a non-equi join of feature.dt (which contains the predicted penalty values) with selection (which contains the number of segments for each penalty value). Then we plot the predicted models along with the data and labels.

```
> pred.models <- feature.dt[selection, nomatch=0L, on=list(
+   profile.id, chromosome,
+   pred.log.lambda < max.log.lambda,
+   pred.log.lambda > min.log.lambda)]
> pred.segs <- segs[pred.models, on=list(profile.id, chromosome, n.segments)]
> pred.changes <- pred.segs[1 < start, ]
```

```
> pred.labels <- errors$label.errors[pred.models, nomatch=0L, on=list(
+   profile.id, chromosome, n.segments)]
> breakpoint.colors <- c(
+   "breakpoint"="#a445ee",
+   "normal"="#f6f4bf")
> viz.learned <- ggplot()+
+   ggtitle("data + labels + learned model segment means and changes")+
+   theme_bw()+
+   theme(
+     legend.position="bottom",
+     legend.box="horizontal",
+     panel.margin=grid::unit(0, "lines"))+
+   facet_grid(profile.id ~ chromosome, scales="free", space="free_x")+
+   penaltyLearning::geom_tallrect(aes(
+     xmin=min/1e6, xmax=max/1e6, fill=annotation, linetype=status),
+     data=pred.labels)+
+   scale_linetype_manual("error type",
+                         values=c(correct=0,
+                                  "false negative"=3,
+                                  "false positive"=1))+
+   scale_fill_manual("label", values=breakpoint.colors)+
+   geom_point(aes(position/1e6, logratio),
+              data=profiles,
+              shape=1)+
+   scale_x_continuous(
+     "position on chromosome (mega bases)",
+     breaks=c(100, 200))+
+   geom_segment(aes(chromStart/1e6, mean, xend=chromEnd/1e6, yend=mean),
+                data=pred.segs,
+                color="green")+
+   geom_vline(aes(xintercept=chromStart/1e6),
+              data=pred.changes,
+              color="green",
+              linetype="dashed")
> print(viz.learned)
>
```



The plot above can be interpreted in the following manner

- black dots are the noisy data points in which we have attempted to detect changepoints.
- each panel is a separate data sequence (a separate multiple changepoint detection problem).
- horizontal solid green lines are the learned segment means.
- vertical dashed green lines are the predicted changepoint positions.
- colored rectangles are the labels. yellow normal labels should have no changepoints, and purple breakpoint labels should have at least one changepoint.

It is clear that in these data, all of the predicted models are consistent with the labels. You can also see that the model makes reasonable predictions for the unlabeled chromosomes.

2.10 Comparison with BIC

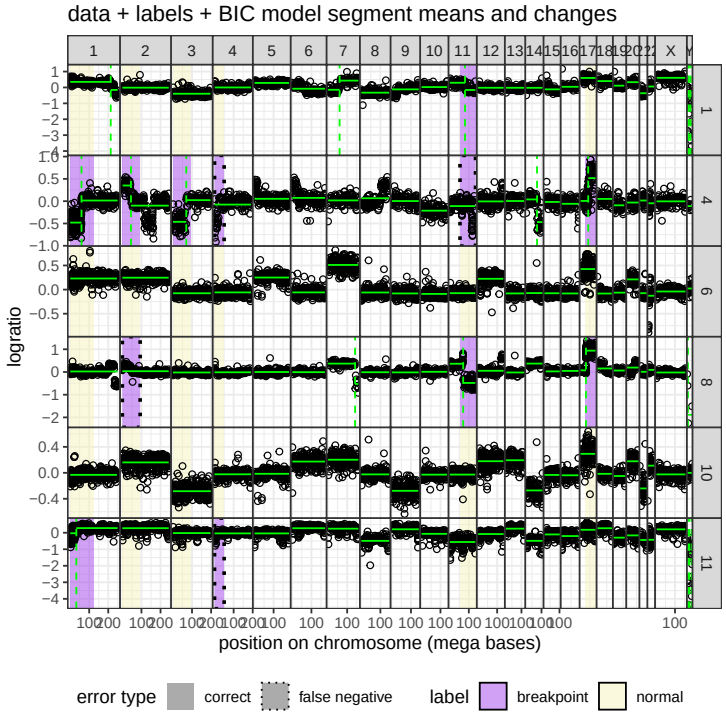
The Bayesian Information Criterion (BIC) corresponds to using the penalty function $f(x_i) = \log \log d_i$ which is unsupervised (nothing to learn). Below we compute the ROC curve for the same test set (chromosome 11) as we used for the learned model.

```
> bic.dt <- profiles[, list(
+   pred.log.lambda=log(log(.N))
+ ), by=list(profile.id, chromosome)]
> bic.test <- bic.dt[chromosome==11]
> bic.roc <- penaltyLearning::ROChange(
+   all.errors, bic.test, c("profile.id", "chromosome"))
> print(bic.roc$thresholds[threshold=="predicted"])

   threshold labels possible.fp possible.fn min.thresh max.thresh    fp    fn
      <char>  <num>       <num>       <num>      <num>      <num> <num> <num>
1: predicted      6           3           3  -1.075214  0.4493126     0     1
   min.fp.fn errors   FPR    tp       TPR error.percent
       <num>  <num> <num> <num>     <num>         <num>
1:         0      1     0     2 0.6666667      16.66667

>
```

It is clear that the BIC suffers from a false negative in chromosome 11 (a purple breakpoint region with no predicted changepoint). We visualize the model along with the data and labels in the plot below,

```
> bic.models <- bic.dt[selection, nomatch=0L, on=list(
+   profile.id, chromosome,
+   pred.log.lambda < max.log.lambda,
+   pred.log.lambda > min.log.lambda)]
> bic.segs <- segs[bic.models, on=list(profile.id, chromosome, n.segments)]
> bic.changes <- bic.segs[1 < start, ]
> bic.labels <- errors$label.errors[bic.models, nomatch=0L, on=list(
+   profile.id, chromosome, n.segments)]
> viz.bic <- ggplot()+
+   ggtitle("data + labels + BIC model segment means and changes")+
+   theme_bw()+
+   theme(
+     legend.position="bottom",
+     legend.box="horizontal",
+     panel.margin=grid::unit(0, "lines"))+
+   facet_grid(profile.id ~ chromosome, scales="free", space="free_x")+
+   penaltyLearning::geom_tallrect(aes(
+     xmin=min/1e6, xmax=max/1e6, fill=annotation, linetype=status),
+     data=bic.labels)+
+   scale_linetype_manual("error type",
+                         values=c(correct=0,
+                                  "false negative"=3,
+                                  "false positive"=1))+
+   scale_fill_manual("label", values=breakpoint.colors)+
+   geom_point(aes(position/1e6, logratio),
+              data=profiles,
```

```
+              shape=1)+
+   scale_x_continuous(
+     "position on chromosome (mega bases)",
+     breaks=c(100, 200))+
+   geom_segment(aes(chromStart/1e6, mean, xend=chromEnd/1e6, yend=mean),
+                data=bic.segs,
+                color="green")+
+   geom_vline(aes(xintercept=chromStart/1e6),
+              data=bic.changes,
+              color="green",
+              linetype="dashed")
> print(viz.bic)
>
```

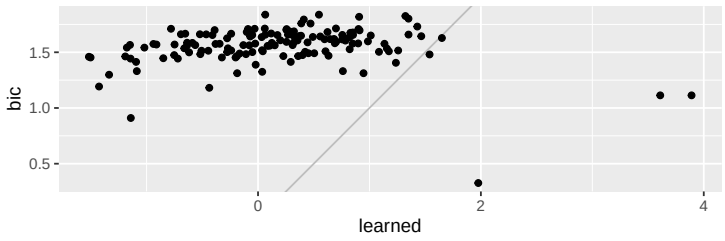


It is clear from the plot above that there are many false negatives – purple breakpoint labels in which there should be at least one change, but the BIC model predicts none.

Finally we can make the scatterplot that compare the predicted penalty values of the learned model and the BIC:

```
> scatter.dt <- data.table(
+   bic=bic.dt$pred.log.lambda,
+   learned=feature.dt$pred.log.lambda)
```

```
> gg.scatter <- ggplot()+
+   geom_abline(
+     slope=1, intercept=0, color="grey")+
+   geom_point(aes(
+     learned, bic),
+     data=scatter.dt)+
+   coord_equal()
> print(gg.scatter)
>
```



It is clear from the scatterplot that the learned model predicts log(penalty) values which are quite different from the BIC.

3 Conclusion

We have shown an application of the penaltyLearning package to the neuroblastoma data set. We showed how to learn a penalty function based on set of data sequences which have labels that indicate presence or absence of changes in specific regions. We showed that the learned penalty function is more accurate than an unsupervised penalty function (BIC).