GPrank User Guide

Hande Topa and Antti Honkela

October 26, 2016

Contents

1 Introduction

The *GPrank* package has been built upon the *gptk* (Gaussian process toolkit) package (Kalaitzis and Lawrence, 2011) with the addition of "fixed variance" kernel which allows to incorporate additional variance information from pre-processing of the observations into the Gaussian process (GP) regression models.

GPs are an ideal model for short and irregularly sampled time series and they can be used to model and rank multiple time series, each generated by different items within an experiment.

In (Topa *et al.*, 2015) and (Topa and Honkela, 2016), we have evaluated the performance of our GP-based ranking method by comparing the precision and recall under different scenarios with and without variance usage. Simulation

results have shown that variance usage leads to a higher average precision, which means less false positives appearing in the top of the ranked list. Motivated by these results, here we will explain how to use GPrank package and then provide examples for two different applications we had in our papers.

2 Citing *GPrank*

To cite *GPrank* in publications, please cite relevant of the two methodology papers (Topa *et al.*, 2015; Topa and Honkela, 2016) that the software is based on.

3 Methods overview

Our GP-based ranking method uses Bayes factors to rank muliple time series where the Bayes factors are computed for each item by the ratio of its marginal likelihood under two alternative GP models, namely time-dependent and time-independent. Time-independent model (which is referred as the *null model* in the package) assumes no temporal dependency between observations and uses only a white noise kernel to model the noise. Time-dependent model (which is referred as the *model* in the package) on the other hand, assumes a smooth temporal behavior, and in addition to the white noise kernel, it also includes an rbf kernel to capture the temporal dependency. Furthermore, we use a fixed variance kernel in both models in order to incorporate variance information which could be obtained by appropriate estimation methods. For more technical details about the GP models, please refer to the papers mentioned in Section 2.

4 General usage of *GPrank*

4.1 Installing the package

In order to install *GPrank* package from the GitHub repository, start R and run the following command:

```
> devtools::install_github("PROBIC/GPrank")
```

In order to install from CRAN, simply use the following command:

```
> install.packages("GPrank")
```

To load the package, run:

```
> library("GPrank")
```

4.2 Necessary data

In order to construct a GP model, three vectors must be provided for each item. These vectors are:

- t: vector containing the input values, i.e., sampled time points.
- y: vector containing the observed values at the corresponding time points in vector t.

- v: vector containing the variances at the corresponding time points in vector t.

Once we have obtained these vectors, we can construct a GP model with `constructModel` function using different kernels such as "rbf", "white", and "fixedvariance": *Example:*

```
> t=seq(0,20,5)
> y=sin(t)
> v=0.01*runif(5)
> kernelTypes=c('rbf','white','fixedvariance')
> model=constructModel(t,y,v,kernelTypes)
```

Please make sure that the three vectors have the same length with each other. If the data is replicated, please remember to adjust the input vector accordingly. For example, if there are two replicates observed at n time points from time t_1 to t_n, vector t must be defined as: $t = [t_1, t_1, t_2, t_2, \ldots, t_n, t_n]$.

4.3 Fitting the models

`gpTest` function takes t, y, and v vectors as input arguments and fits two alternative GP models to the data, and computes the log Bayes factors:

```
> test_result=gpTest(t,y,v,
+ nullModelKernelTypes=c("white","fixedvariance"),
+ modelKernelTypes=c("rbf","white","fixedvariance"))
> null_model=test_result$nullModel
> model=test_result$model
> logBF=test_result$logBF
```

4.4 Visualizing the models

In order to visualize the fitted GP model, one can use the `plotGP` function. One can also specify the color of the plot with the second argument. One can optionally specify the limits of the y axis as the third argument. This helps to adjust the plotting area when GP models of multiple items are displayed in a single figure. For example, y-axis limits can be determined by using `getYlimits` function in such cases. This function adjusts the plotting area between the minimum and maximum values of multiple models also taking into account two standard deviation confidence intervals. In addition, a color palette containing the distinctive colors from *RColorBrewer* package can be obtained with the function `getColorVector`. The generated plot in Figure 1 displays ± 2 standard deviations confidence region (estimated from the fitted model) around the fitted line and errorbars denoting ± 2 standard deviations (provided from fixed variances) around the observations.

```
> color="lightpink" # color=getColorVector()[1]
> ylimits=getYlimits(y,v) # optional argument, also default
> plotGP(model, color, ylimits)
> title(xlab="t", ylab="y")
```

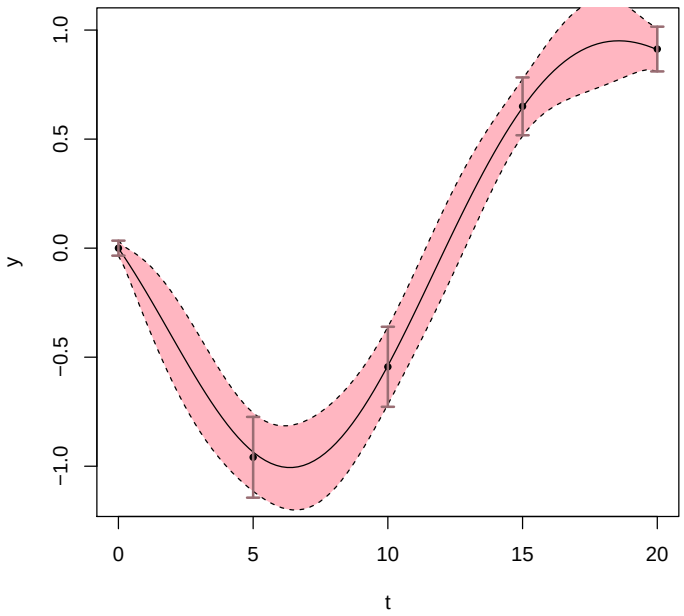

Figure 1: Fitted GP model for the example in Section 4.4.

4.5 Building SQLite database

Once we have had all the results ready, and saved the figures in png format, we can use `createDatabase` function to create a database which can be used to view the results in the web browser with the help of *tigreBrowser* package (Honkela *et al.*, 2011). *tigreBrowser* can display the selected GP profiles on a web browser enabling to rank them according to provided parameters, such as Bayes factors and fold changes. In order to view database in the browser, please remember to place other required tigreBrowser files in the same folder with the database as well. Also modify the database name accordingly in "tigreBrowser.cfg". For more information please see: `https://github.com/PROBIC/tigreBrowser`.

```
> BF=c(3,10,2) # Bayes factors
> FoldChange=c(0.5,3,5) # Fold changes
> dbParams=list("BF"=BF,"Fold change"=FoldChange)
> identifiers=c("geneA","geneB","geneC")
> dbInfo=list(database_name="testdb","database_params"=dbParams,
+ "identifiers"=identifiers)
> figuresPath="figures/" # directory where the figures are saved
> multi=1 # multiple figures will be displayed for each item
> createDatabase(dbInfo,figuresPath,multi)
```

Note: Please place all figures in a subdirectory and specify its path in `figuresPath`, name the figures starting with the name of their corresponding iden-

tifiers. If there are multiple figures to be displayed for each item, add the specific extension after an underscore and set `multi` to TRUE. For example: "geneA_gene.png", "geneA_abstr.png", "geneA_reltr.png". If there is only a single figure for each item, name it with its identifier. For example: "geneA.png".

5 Applications

5.1 RNA-seq transcript expression analysis using BitSeq

5.1.1 Sample data

For demonstrating the usage of the functions with examples, we will be using a small sample data from an RNA-seq time series experiment which was introduced in (Honkela *et al.*, 2015). The sample data set, named RNAseqDATA, contains mean and standard deviation information on the expression levels of 5 transcripts (which were originated from 2 genes) at 10 time points (0, 5, 10, 20, 40, 80, 160, 320, 640, 1280 mins) for three settings: "gene", "abstr" (absolute transcript), and "reltr" (relative transcript) expression levels. In addition, the fields "gene_mapping" and "time_mapping" includes information which is useful to match the genes with transcripts and the time points with data files, respectively. In order to load the data set, type:

```
> library("GPrank")
> data(RNAseqDATA)
```

If one is interested in getting this data structure from raw BitSeq output files himself, he may use the `bitseq_rnaSeqData` function:

```
> t=log(c(0,5,10,20,40,80,160,320,640,1280)+5) # One can apply
> #transforation on time points
> names(t)=c("t0000.rpkm","t0005.rpkm","t0010.rpkm","t0020.rpkm",
+ "t0040.rpkm","t0080.rpkm","t0160.rpkm","t0320.rpkm","t0640.rpkm",
+ "t1280.rpkm") # matches with the names of the BitSeq output files
> trFileName="example_tr"
> bitseq_sampleData=bitseq_rnaSeqData(t,trFileName)
```

5.1.2 Fitting the models

From now on, let us continue with the gene-level data although one can simply perform the same with `reltr` and `abstr` levels as well. The function `bitseq_fitGPs` can be used to fit two GP models to each gene and compute the log Bayes factors:

```
> gene_gpData=RNAseqDATA$gene
> gene_GP_models=bitseq_fitGPs(gene_gpData)
```

If one is interested in saving the results into files, one should remember to specify the file names for `fileName_logBF`, `fileName_ModelParams`, `fileName_NullModelParams` and input them as arguments in the `bitseq_fitGPs` function:

```
> gene_GP_models=bitseq_fitGPs(gene_gpData, fileName_logBF,
+ fileName_ModelParams,fileName_NullModelParams)
```

5.1.3 Visualizing the models

Having the GP models fitted to the genes, one can plot the GP profile of a specified gene with the function `bitseq_plotGP`. For example, the GP profile of the gene ARAP2 shown in Figure 2 can be obtained by the following codes:

```
> item="ARAP2"
> multi=0 # single GP plot in the figure
> ylimits=NULL
> x_ticks=NULL
> x_label="log(5 + t/min)"
> y_label="Expression level (log-rpkm)"
> bitseq_plotGP(item, gene_GP_models, gene_gpData, multi, ylimits,
+ x_ticks, x_label, y_label)
```

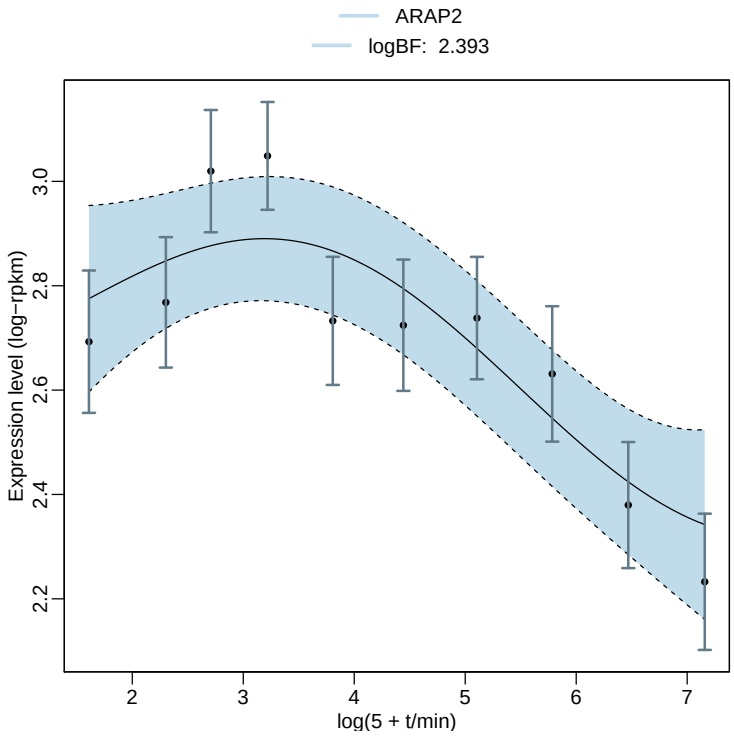

Figure 2: Fitted GP model for the overall gene expression levels

In order to save the figure, one can also specify the figure name in `plotName` option:

```
> bitseq_plotGP(item, gene_GP_models, gene_gpData, multi, ylimits,
+ x_ticks, x_label, y_label, plotName="ARAP2_gene.png")
```

The input `multi` determines whether multiple plots (=1) or only a single plot (=0) will be plotted on the same figure. For example, if we would like to plot the GP profiles of all the transcripts of ARAP2 gene, we can display all

on the same plot by setting `multi` to 1. Let's try that for absolute transcript expression levels and produce Figure 3:

```
> abstr_gpData=RNAseqDATA$abstr
> abstr_GP_models=bitseq_fitGPs(abstr_gpData)
> item="ARAP2"
> multi=1
> ylimits=NULL
> x_ticks=NULL
> x_label="log(5 + t/min)"
> y_label="Expression level (log-rpkm)"
> bitseq_plotGP(item, abstr_GP_models, abstr_gpData, multi, ylimits,
+ x_ticks, x_label, y_label)
```

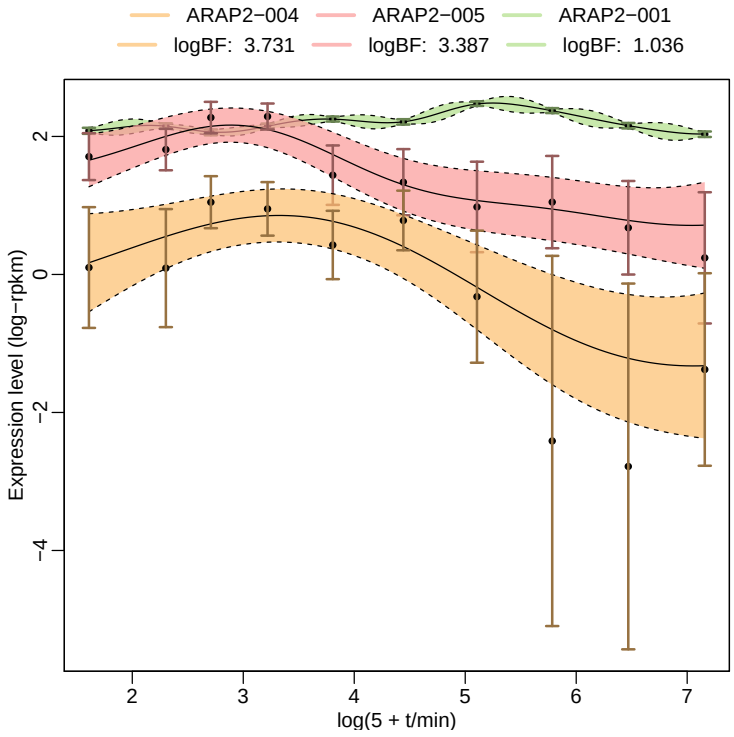

Figure 3: Fitted GP model for the absolute transcript expression levels

Let us also do the same for relative transcript expression levels and obtain Figure 4:

```
> reltr_gpData=RNAseqDATA$reltr
> reltr_GP_models=bitseq_fitGPs(reltr_gpData)
> item="ARAP2"
> multi=1
> ylimits=c(0,1) # ratio range between 0 and 1
> x_ticks=NULL
```

```
> x_label="log(5 + t/min)"
> y_label="Relative expression level"
> plotName="ARAP2_reltr.pdf"
> bitseq_plotGP(item, reltr_GP_models, reltr_gpData, multi, ylimits,
+ x_ticks, x_label, y_label)
```

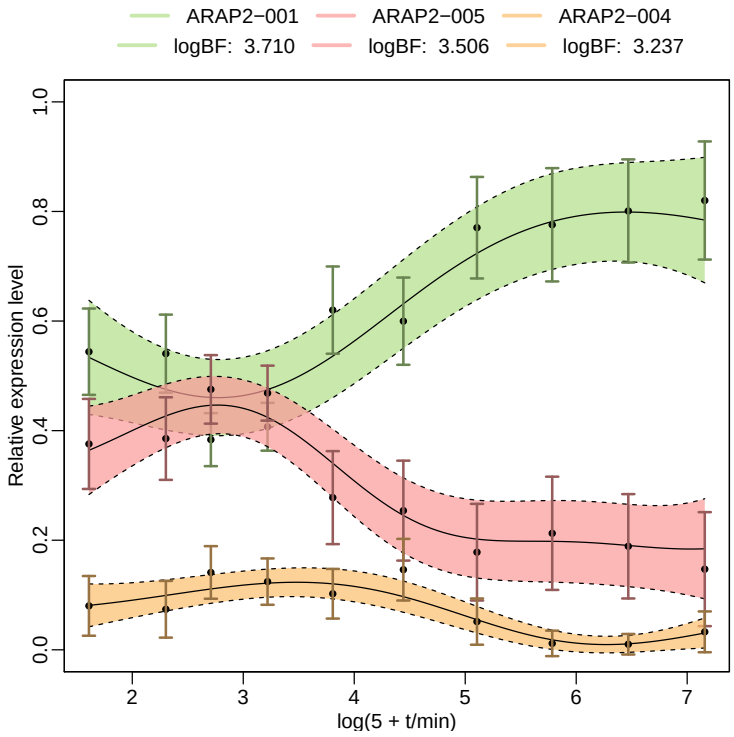

Figure 4: Fitted GP model for the relative transcript expression levels

5.2 Quantitative analysis of population sequencing data

In (Topa *et al.*, 2015), we developed a GP-based method BBGP — beta binomial Gaussian process — for modeling the SNP frequencies in fruit fly populations across several generations in an experimental evolution study. The same method can in principle be used to analyse any population sequencing data where one is interested in the proportion of reads aligning to a specific location that contain a specific feature (such as a SNP).

5.2.1 Sample data

Here we will use a small sample data set named `snpData`. This sample data set contains 5 replicates of counts and sequencing depth information for 5 SNPs at the generations (0, 10, 20, 30, 40, 50, 60). In order to load the data set, run the command:

```
> data(snpData)
```

If one is interested in getting this data structure from raw sample files himself, he may use the `bbgp_snpData` function:

```
> dataFileName="sampleCountsData"
> sampleSNPdata=bbgp_snpData(dataFileName)
```

5.2.2 Fitting the models

Given the counts and sequencing depth, we can use `get_bbgpMeanStd` function in order to get the posterior means and standard deviations of the frequencies using a beta binomial model with parameters α and β set to 1.

```
> x=as.matrix(as.numeric(colnames(snpData$counts)))
> # take the fifth SNP in the sample data as example:
> counts=as.matrix(snpData$counts[5,])
> seq_depth=as.matrix(snpData$seq_depth[5,])
> bbgp=get_bbgpMeanStd(x,counts,seq_depth)
> t=bbgp$time
> y=bbgp$posteriorMean
> v=(bbgp$posteriorStd)^2
```

Then, we can perform our GP-based test with `gpTest` function:

```
> snp_gpTest=gpTest(t,y,v)
```

5.2.3 Visualizing the models

Once we have fitted the GP model, we can visualize it using `plotGP` function and obtain Figure 5:

```
> model=snp_gpTest$model
> ylims=c(0,1)
> plotGP(model, ylimits=ylims)
> title(xlab="Time", ylab="SNP frequency")
```

6 Session info

```
> sessionInfo()

R version 3.3.1 (2016-06-21)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 14.04.5 LTS

locale:
 [1] LC_CTYPE=fi_FI.UTF-8    LC_NUMERIC=C            LC_TIME=en_GB
 [4] LC_COLLATE=C            LC_MONETARY=fi_FI.UTF-8 LC_MESSAGES=en_GB
 [7] LC_PAPER=en_GB          LC_NAME=C               LC_ADDRESS=C
[10] LC_TELEPHONE=C          LC_MEASUREMENT=en_GB    LC_IDENTIFICATION=C

attached base packages:
```

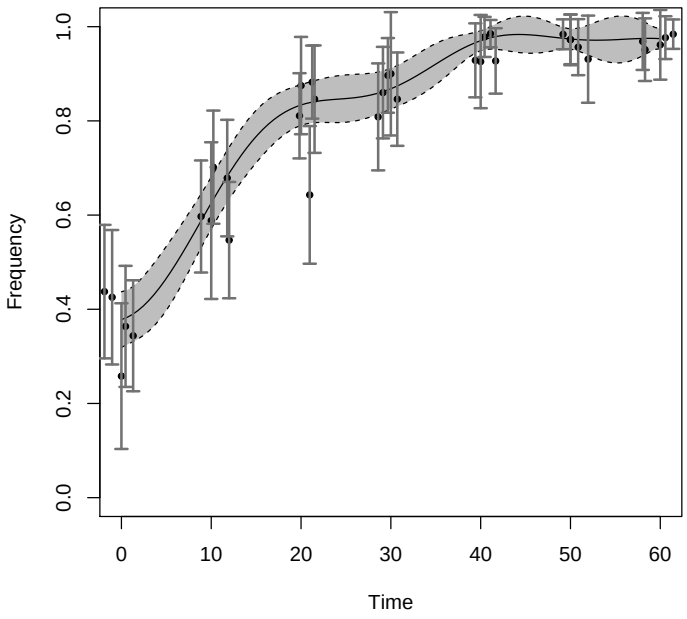

Figure 5: Fitted GP model of the SNP frequencies across generations

```
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
[1] GPrank_0.1

loaded via a namespace (and not attached):
 [1] Matrix_1.2-6             DBI_0.5-1                tools_3.3.1
 [4] gptk_1.08                RColorBrewer_1.1-2       fields_8.4-1
 [7] maps_3.1.0               tigreBrowserWriter_0.1.1 RSQLite_1.0.0
[10] grid_3.3.1               spam_1.3-0               matrixStats_0.51.0
[13] lattice_0.20-33
```

References

Honkela, A et al. (2011). tigre: Transcription factor inference through gaussian process reconstruction of expression for bioconductor. *Bioinformatics*, **27(7)**, 1026–1027.

Honkela, A et al. (2015). Genome-wide modeling of transcription kinetics reveals patterns of RNA production delays. *Proceedings of the National Academy of Sciences*, **112**(42), 13115–13120.

Kalaitzis, A. A. and Lawrence, N. D. (2011). A simple approach to ranking differentially expressed gene expression time courses through Gaussian process regression. *BMC Bioinformatics*, **12**, 180.

Topa, H. and Honkela, A. (2016). Analysis of differential splicing suggests different modes of short-term splicing regulation. *Bioinformatics*, **32(12)**, i147–i155.

Topa, H et al. (2015). Gaussian process test for high-throughput sequencing time series: application to experimental evolution. *Bioinformatics*, **31(11)**, 1762–1770.