Vignette: How to use ABEILLE

We will demonstrate in this vignette how to use ABEILLE and show how we can customize the package to perform a different task. We first need to load the library. The instructions to install can be found here.

```
# Loading ABEILLE library
library("ABEILLE")
```

We will use the toy dataset located in this repository to perform our demonstration

```
toy_dataset <- ExampleAbeilleDataSet
print(paste("The dataframe contains",
            dim(toy_dataset)[1],"transcripts and",
            dim(toy_dataset)[2],"samples"))

## [1] "The dataframe contains 14000 transcripts and 128 samples"

head(toy_dataset[,1:6])

##              GTEx_1 GTEx_2 GTEx_3 GTEx_4 GTEx_5 GTEx_6
## Transcript_1      2      0      0      0      0      0
## Transcript_2     60     67     73     57    135     46
## Transcript_3      0      0      0      0      0      0
## Transcript_4      0      0      0      0      0      0
## Transcript_5   7638   4057   7978   3913    175   3471
## Transcript_6      2      0      0      1      3      0
```

1 Quality control of the data

Once the dataset loaded as dataframe in which genes are in rows and samples in columns, we need now to ensure that the data are conform and can be used within ABEILLE, so we run **DataIntegrity** function.

```
DataIntegrity(toy_dataset)

## Data integrity is validated
```

2 Remove unexpressed genes in all patients

Filter out unexpressed genes in all patients with the function **RemoveZeroCounts**.

```
toy_dataset[14001,] <- 0
print(dim(toy_dataset))

## [1] 14001   128
```

```
toy_dataset <- RemoveZeroCounts(toy_dataset)

## 1 transcripts removed over 14001 (which corresponds to 0.01 % of the dataset)
```

Save the dataset that was created after removing the zero counts.

```
write.csv(toy_dataset, "toy_dataset.csv")
```

3 Reconstruct the data with a Variational AutoEncoder (VAE)

The third step is to use the VAE to get reconstructed data from the original filtered data.

```
library(reticulate)
#Source my own python environment
#To find your python path you can use os.path.dirname(sys.executable) on python
use_python("C:/ProgramData/Anaconda3")
#If you don't have environment ready, you can load library needed in python
py_install("numpy")
py_install("pandas")
py_install("tensorflow")
#Load the VAE function
source_python("abeille.py")
toy_dataset_recons <- abeille_VAE("toy_dataset.csv")
```

Here is the list of all the arguments usable in the abeille_VAE function:

WARNING: if you use it on R arguments using an integer needs to be set as 20L for 20 (for example)

- **file** (only mandatory parameter), the path to the file to pass in the autoencoder.
- **logarithm** (default True), boolean to log the input data.
- **read_count** (default True), boolean to correct value bellow 0 (needed if you use read count data).
- **batch_size** (default 30), number of training examples utilized in one iteration.
- **epochs** (default 1500), number of time the dataset is passed forward and backward through the neural network.
- **kernel** (default "lecun_normal"), type of initialization of the neurons.
- **kl_loss_weight** (default 0.5), weight of the Kullback - Leibler loss.
- **edl1** (default 2048), stand for encoder dense layer 1, number of layer in the first hidden layer.
- **edl2** (default 1024), stand for encoder dense layer 2, number of layer in the second hidden layer.
- **edl3** (default 512), stand for encoder dense layer 3, number of layer in the third hidden layer.

- **edl4** (default 256), stand for encoder dense layer 4, number of layer in the fourth hidden layer.
- **latent_size** (default 128), size of the latent space
- **ddl1** (default 256), stand for decoder dense layer 1, number of layer in the sixth layer.
- **ddl2** (default 512), stand for decoder dense layer 2, number of layer in the seventh layer.
- **ddl3** (default 1024), stand for decoder dense layer 3, number of layer in the eigth layer.
- **ddl4** (default 2048), stand for decoder dense layer 4, number of layer in the ninth layer.

An example of reconstructed data is available in ABEILLE package, we will use it for the rest of the pipeline.

```
toy_dataset_recons <- ExampleAbeilleReconstructed
```

To judge the performance of the reconstruction, use the function **StatsPred**.

```
StatsPred(toy_dataset, toy_dataset_recons)

## Number of observations : 1792000
## Mean : 683.2478
## Median : 1
## Min : 0
## Quantile 10% : 0
## Quantile 25% : 0
## Quantile 75% : 28
## Quantile 90% : 798
## Max : 2904284
## Standard deviation : 9527.474
## Skewness : 90.22985
## Empty samples: 0 on 128
## Empty transcripts: 5419 on 14000
## Number of zeros: 888391 on 799381
```

4 Compute two novel metrics

From input and reconstructed data, two metrics are computed : the divergence score and the delta count with **DivergenceScore** and **DeltaCount** functions.

```
divergence_score <- DivergenceScore(toy_dataset, toy_dataset_recons)
delta_count <- DeltaCount(toy_dataset, toy_dataset_recons)
```

5 Identify AGE

Finally, to obtain the AGEs, use the function **IdentifyAGE**. This function will also use the function **ComputeAnomalyScore** that will return an anomaly score associated to each AGE between 0 and 1.

```
ages <- IdentifyAGE(divergence_score, delta_count, toy_dataset,
                    toy_dataset_recons)
print(head(ages))
```

The default decision tree **my_tree** was built to find AGEs following the OUTRIDER AGEs distribution but the function can work with your own decision tree.

```
light_tree <- function(linear_reg_data){
  bool_vector <- ifelse(linear_reg_data["cooksD"] >= 15.0,TRUE,FALSE)
  return(bool_vector)
}
ages <- IdentifyAGE(divergence_score, delta_count, toy_dataset,
                    toy_dataset_recons, decision_tree = light_tree)
print(head(ages))
```