

Analysis of Rail Lubrification

Autoren	e-sr & lucmiaz
Version	Draft
Letzte Änderung	November 21, 2015
Letzte Änderung durch	lucmiaz
Urheberrecht	Dieses Dokument ist urheberrechtlich geschützt. Jegliche kommerzielle Nutzung bedarf einer vorgängigen, ausdrücklichen Genehmigung.

Contents

1	Analysis of rail lubrication : Introduction	2
1.1	Finding an optimal algorithm for flanging noise detection	2
1.2	Hand selection of intervals	2
1.3	Development of a proto-algorithm	2
1.4	Optimisation of the proto-algorithm	2
2	Analysing the data	6
2.1	Reproducing the calculations	6
2.1.1	Structure of the database	6
2.2	Getting started in R	6
3	Appendices	7
3.1	Best thresholds for each author	7
3.2	ROC plot for other authors	7
3.3	Structure of the Graph Database	8
3.3.1	Nodes	8
3.3.2	Relationships	13

1 Analysis of rail lubrication : Introduction

1.1 Finding an optimal algorithm for flanging noise detection

The aim of this part of the project is to implement an algorithm capable of telling whether a train is making flanging or squealing noise on a audio recording. This is a requisite for the next stage, which is analysing the performance of rail lubrication on noise reduction. We have proceeded in three steps :

- Hand selection of intervals on a subset of the samples
- Development of a proto-algorithm
- Optimisation of the proto-algorithm

1.2 Hand selection of intervals

We have selected by hands the intervals with noise on a small subset of samples (about twenty). We asked other people of our team to do the same. This gives us a subjective set of data to compare our algorithms. The **subjective** part is very important as the perception of what an annoying noise is cannot be reduced solely to physical variables.

The process of selection was performed on an interface in PyQt created for this project. The python file to call to start this interface is `run_CaseCreatorWidget`. The interface has two shapes: the first one is the simplest and most stable, it is only designed to select the cases; the second is not fully stable but implements an administrator environment that allows to review the intervals selected by the authors, review the algorithms and see the spectrogram of the stft – it is a fairly long process, so be patient if you call it.

1.3 Development of a proto-algorithm

We have developed a simple algorithm for flanging noise based on the detection of steep change in band power ratio (BPR). The band power ratio is the relation of the low band frequencies to the high band frequencies taken on a small increment (dt) of the signal. This pattern leads to an algorithm with three variables: a delta time, a cutoff frequency (fc) and a threshold. This approach is described in general terms in Bullen and Jiang article for railways **Bullen2010**¹.

We computed the BPR for cutoff frequencies and delta times, first without worrying about the threshold. This gave us a list of BPRs to compare with the set of handmade data. We tested the combinations of the following fc and dt each giving a proto-algorithm (as it misses the threshold): fc in (2000, 3000, 3500, 4000, 45000) and dt in (0.02, 0.05, 0.1). This algorithm is named `Zischendetekt2` (Z2) in the python files.

We noticed that one way to improve this algorithm would be to change the lower and upper bounds of the low and high bands (we noted them f_{min} and f_{max} in the code, especially in `algorithm.py` from the `kg` package). The default for `Zischendetekt2` was $f_{min} = 100$. This could easily be raised to 300. This would be one way to further improve our results. However this was not done at first and we shall use Z2 in the following article.

1.4 Optimisation of the proto-algorithm

For all the previously obtained proto-algorithms, we computed the True Positive Rate (i.e. the ratio of increments that were rightly selected by the algorithm as containing noise) and the False Positive Rate (i.e. the ratio of increments wrongly selected as containing noise) with 300 thresholds slip between the smallest and highest BPR. With this we will compute the so-called Receiving Operating Characteristic (ROC). More can be found on this method in Swets, Dawes and Monahan popularisation article (2000). This procedure was performed with the python code `main_analysis.py`. Then we merged the multiple files created by `main_analysis.py` together in a csv file we called `datamaous.csv`. The rows of this file correspond to a delta time of the discretisation performed with `main_analysis.py`. The merge was done with the python script `commuter.py`.

For each now-complete algorithm, we selected the best one, that is - for us - the one that is the farthest from the diagonal spanning from (0,0) to (1,1),.

¹We found later that a similar algorithm was use for tonal bird sound recognition (**Jancovic2011**)

This is how we proceeded:

```
1 library('dplyr')
2 csv_file="../Algorithm_analysis/Datamaous.csv"
3 tf<-read.csv(csv_file)
4 tf<-tf %>% mutate(algorithm=paste(alg,alprop,sep="_"))
5 source('../Algorithm_analysis/R/find_best.R') #load the functions inside find_best.R
```

find_best.R contains two functions that basically do this:

Code 1: find_best.R

```
1 TFPF_func<-function(df, threshold){
2   #spec correspond to the BPR and disc to choise of the author : 1 is for flanging
3   #tally function will count the TP, i.e. BPR is over the threshold and the author
4   #said it is flanging
5   TP<-tally(filter(df, spec>threshold & disc==1)) #True positives
6   FP<-tally(filter(df, spec>threshold & disc==0)) #False positives
7   totP<-tally(filter(df, disc==1)) #Total positives
8   totF<-tally(filter(df, disc==0)) #Total negatives
9   if (totP>0 & totF>0){
10    FPR<-FP/totF
11    TPR<-TP/totP
12    d_ax<-(TPR+FPR)/2 #compute closest point on main diagonal
13    dist_ax<-sqrt( (d_ax-FPR)**2 + (d_ax-TPR)**2 ) **2 #compute euclidean distance
14    to d_ax
15    #returns a data.frame (empty if totP or totF is 0)
16  }
17 }
18
19 find_best<-function(tff, authors=list(), qualities=list(), fixedthreshold=FALSE,
20   bw=200){
21   #iterate on algorithms
22   ...
23   #iterate on algorithms parameters
24   ...
25   #iterate on thresholds
26   ...
27   call TFPF_func
28   add the best threshold (i.e. with dist_ax maximal) and its corresponding
29   attributes to the return data.frame
30 }
```

This gave us the following table for our four authors – the output give the name of the authors and the qualities selected, from 1 (bad) to 3 (good):

To get an idea of the comparative accuracy of these algorithms, we decided to look at their ROC graph. A ROC graph visually compares the True Positive Rate with the False Positive Rate. For a popularised introduction please see (**Swets2008**) or (**Fawcett2006**) for a more detailed discussion.

The accuracy of each algorithm with the optimised threshold depends on the author that selected the intervals. The choices of two of our authors (esr and PHF) could be approximated fairly well by an algorithm – the distance to the main diagonal was over 0.40². However, to take into account the other authors, we selected all combinations of algorithm-threshold-author that were distant from the diagonal by over 0.35 and added it to the list of the algorithm-threshold. For each algorithm, we took the mean of the thresholds available (i.e. if there was an other threshold from author esr or PHF, we took the mean of these thresholds with the threshold of all authors). Then we computed the TPR, TNR, FPR and FNR for all authors and globally. This was done like this:

```
1 walloffame<- rbind(cbind(best, author='all'), cbind(bestesr,author='esr'),
2   cbind(bestluc,author='luc'), cbind(bestPhC,author='PhC'),
3   cbind(bestPHF,author='PHF')) #group the best algorithms from all authors
```

²See Appendice ?? on page ?? to see the respective ROC plots

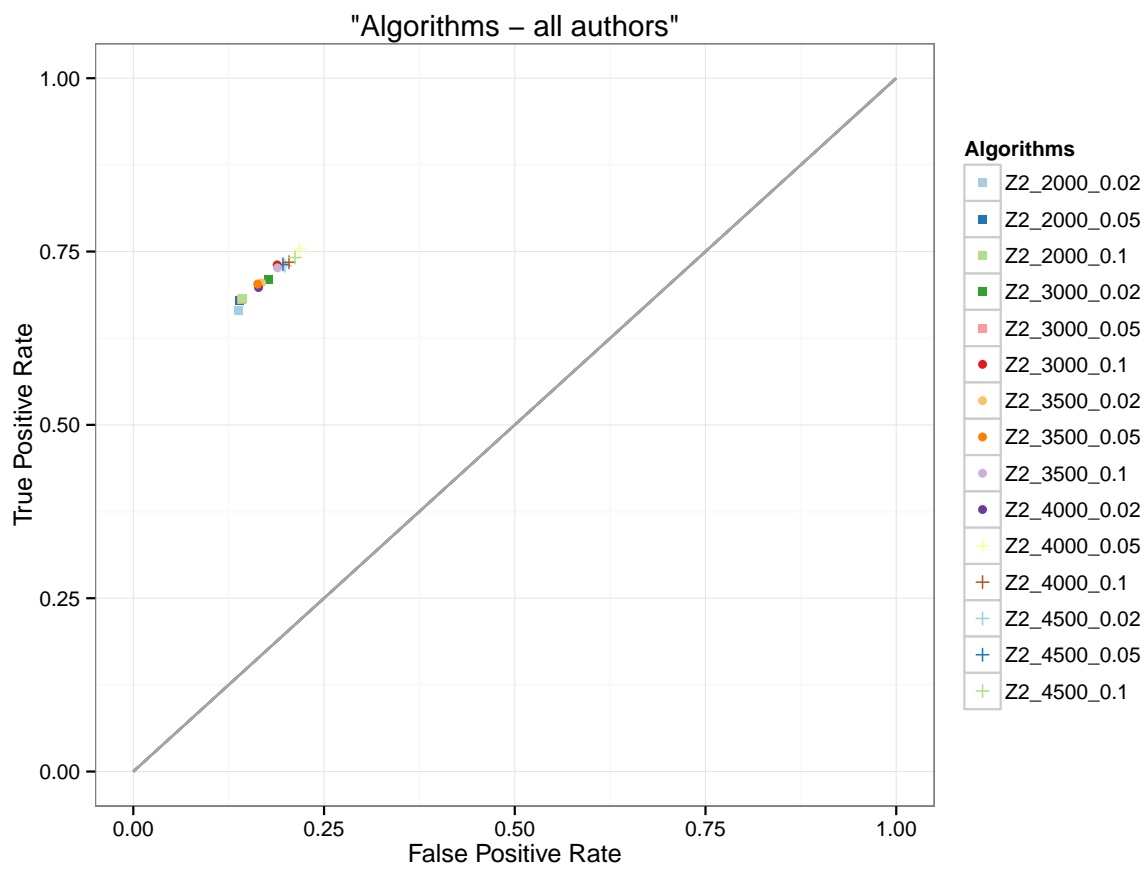


Figure 1: ROC plot of the algorithms with their best threshold, tested on all authors. The plot represents the True Positive Rate over the False Positive Rate.

	TPR	FPR	totP	totF	thd	dist_ax	algorithm	dif
1	0.665	0.137	15550	26382	2.657	0.373	Z2_2000_0.02	0.53
2	0.710	0.177	15550	26382	1.290	0.377	Z2_3000_0.02	0.53
3	0.705	0.168	15550	26382	1.087	0.379	Z2_3500_0.02	0.54
4	0.698	0.164	15550	26382	0.975	0.378	Z2_4000_0.02	0.53
5	0.729	0.199	15550	26382	0.615	0.375	Z2_4500_0.02	0.53
6	0.679	0.139	6216	10553	3.078	0.382	Z2_2000_0.05	0.54
7	0.681	0.141	6216	10553	1.955	0.382	Z2_3000_0.05	0.54
8	0.703	0.163	6216	10553	1.396	0.382	Z2_3500_0.05	0.54
9	0.753	0.218	6216	10553	0.814	0.379	Z2_4000_0.05	0.54
10	0.731	0.196	6216	10553	0.778	0.379	Z2_4500_0.05	0.54
11	0.682	0.143	3104	5287	3.197	0.381	Z2_2000_0.1	0.54
12	0.731	0.189	3104	5287	1.639	0.383	Z2_3000_0.1	0.54
13	0.726	0.189	3104	5287	1.370	0.380	Z2_3500_0.1	0.54
14	0.735	0.204	3104	5287	1.055	0.375	Z2_4000_0.1	0.53
15	0.741	0.212	3104	5287	0.842	0.374	Z2_4500_0.1	0.53

Table 1: Best thresholds for the fifteen algorithms obtained while considering all the authors together

```

2 walloffamest<-filter(walloffame, walloffame$dist_ax>0.45 |
  (walloffame$author=='all' & walloffame$dist_ax>0.35))#select a subset of
  walloffame with best thresholds
3 compressed<-walloffamest %>% group_by(algorithm) %>% summarise(
  avgthres=mean(thd)) %>% arrange(avgthres)#summarise walloffamest by taking the
  mean threshold for every duplicate algorithm (if such duplicate exists)

```

	algorithm	avgthres
1	Z2_4500_0.02	0.68
2	Z2_4500_0.1	0.73
3	Z2_4500_0.05	0.73
4	Z2_4000_0.02	0.91
5	Z2_3500_0.02	1.05
6	Z2_4000_0.05	1.39
7	Z2_3000_0.02	1.77
8	Z2_3500_0.05	1.79
9	Z2_4000_0.1	1.81
10	Z2_3500_0.1	2.00
11	Z2_3000_0.05	2.22
12	Z2_3000_0.1	2.58
13	Z2_2000_0.02	2.89
14	Z2_2000_0.05	3.23
15	Z2_2000_0.1	3.53

Table 2: Best thresholds for the fifteen algorithms obtained while considering all the authors together

We ran `main_analysis.py` for the proto-algorithms with their respective average threshold on the whole set of hand selected cases and as before merged them in one csv with `commuter.py`. This gave us one file that we called `Datamous2.csv`. The performance of these average algorithms is illustrated in figure ???. The full results being in table ???.

Looking at table ??? we chose two algorithms that seemed to fit fairly everyone. This resulted in the following selection³:

- `ZischenDetetkt2(4500,0.7267437,0.1)` : it is the farthest (0.3905650) from the diagonal when we consider all authors separately and then sum them up (`dist_tot`);
- `ZischenDetetkt2(3500,1.0474294,0.02)` : it is the farthest (0.3911725) from the diagonal when we consider all authors together (`dist_ax`);

³The properties of the algorithm `ZischenDetetkt2` are presented in this order : `fc`, `threshold`, `dt`.

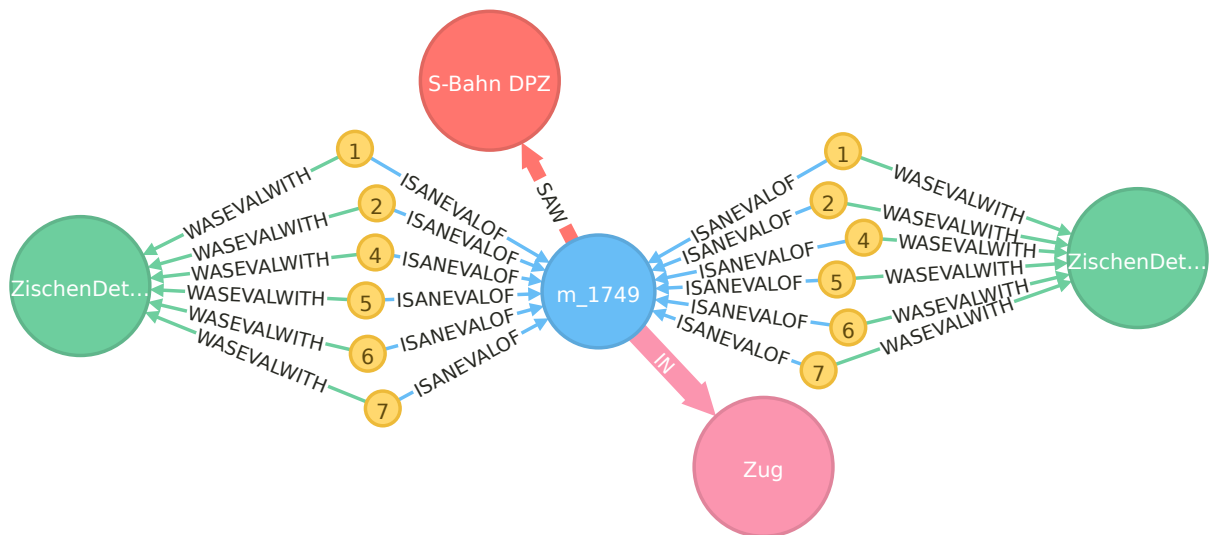


Figure 2: Illustration of the nodes and relationships types in the Neo4j database.

2 Analysing the data

2.1 Reproducing the calculations

The following calculations are done by calling data stored in a neo4j graph database from R. To insert data in this database, first use `main_analysis.py` to generate a series of json file containing all the useful info on your new data. Then use the code `toDB.py` to import these data in the neo4j database – before running the script, please read the head lines and make the necessary changes described there.

In R, the database is called as follows – don't forget to launch the neo4j interface and start the local server if necessary :

```
1 library(RNeo4j) #call RNeo4j package
2 graph=startGraph('http://localhost:7474/db/data', username='neo4j',
  password='admin') #opens a port to the db, with username 'neo4j' and password
  'admin'
3 #to access the database click this link :
  'http://neo4j:admin@localhost:7474/db/data'
```

2.1.1 Structure of the database

The databases in Neo4j are graphs, containing nodes and relationships between these nodes. Nodes and relationships have types (or labels). This will differentiate between Passings and Algorithms, in the case of Nodes, and between the relation of seeing (Passing saw this type of train) and the relation of taking place there (Passing took place here). To gain insight of this database please look at figure ?? and at the structure in appendice ?? on page ??.

2.2 Getting started in R

We will import the measurements for all the algorithms. RNeo4j accepts `data.frames` as input, so we will give it that. In the following R-code, we will call all evaluations made with the algorithms in the database, all Passings attached to those evaluations and all trains types of those Passings.

```
1 query='
2 MATCH (a:Algorithm)
3 MATCH (b)<-[:ISANEVALOF]-(c)-[:WASEVALWITH]->(a) //the relationship s contains the
  specific lengths of time (masked and not) of the microphone signal (c)
4 MATCH (t)<-[:SAW]-(b) //the relationship r contains info on the train of this
  Passing (lenght and track)
```

```

5 RETURN a.Name, c.tNoisemasked, c.tNoise, s.tEval, s.tEvalmasked, t.Name,
   r.trainLength, r.Track
6 '
7 q<-cypher(graph,query)#takes the info of each MicMes for one algorithm

```

3 Appendices

3.1 Best thresholds for each author

The following tables show the best algorithms (i.e. proto-algorithm + threshold) for the four authors.

	TPR	FPR	totP	totF	thd	dist_ax	algorithm	dif
1	0.783	0.132	6172	11718	2.657	0.460	Z2_2000_0.02	0.65
2	0.794	0.144	6172	11718	1.567	0.460	Z2_3000_0.02	0.65
3	0.826	0.168	6172	11718	1.087	0.465	Z2_3500_0.02	0.66
4	0.818	0.162	6172	11718	0.975	0.464	Z2_4000_0.02	0.66
5	0.819	0.164	6172	11718	0.790	0.463	Z2_4500_0.02	0.65
6	0.797	0.134	2469	4684	3.078	0.468	Z2_2000_0.05	0.66
7	0.792	0.135	2469	4684	1.955	0.465	Z2_3000_0.05	0.66
8	0.821	0.162	2469	4684	1.396	0.466	Z2_3500_0.05	0.66
9	0.845	0.193	2469	4684	0.993	0.461	Z2_4000_0.05	0.65
10	0.837	0.183	2469	4684	0.864	0.463	Z2_4500_0.05	0.65
11	0.779	0.120	1229	2351	3.513	0.466	Z2_2000_0.1	0.66
12	0.732	0.088	1229	2351	2.882	0.456	Z2_3000_0.1	0.64
13	0.771	0.125	1229	2351	2.004	0.456	Z2_3500_0.1	0.65
14	0.763	0.128	1229	2351	1.751	0.449	Z2_4000_0.1	0.64
15	0.852	0.218	1229	2351	0.842	0.448	Z2_4500_0.1	0.63

Table 3: Best thresholds for the fifteen algorithms obtained while considering all the authors together

	TPR	FPR	totP	totF	thd	dist_ax	algorithm	dif
1	0.745	0.059	1407	3464	3.371	0.485	Z2_2000_0.02	0.69
2	0.751	0.052	1407	3464	2.444	0.494	Z2_3000_0.02	0.70
3	0.817	0.115	1407	3464	0.969	0.496	Z2_3500_0.02	0.70
4	0.823	0.116	1407	3464	0.791	0.500	Z2_4000_0.02	0.71
5	0.817	0.130	1407	3464	0.624	0.486	Z2_4500_0.02	0.69
6	0.767	0.081	562	1387	3.548	0.485	Z2_2000_0.05	0.69
7	0.765	0.061	562	1387	2.765	0.498	Z2_3000_0.05	0.70
8	0.762	0.057	562	1387	2.581	0.498	Z2_3500_0.05	0.70
9	0.751	0.053	562	1387	2.358	0.494	Z2_4000_0.05	0.70
10	0.867	0.185	562	1387	0.549	0.482	Z2_4500_0.05	0.68
11	0.762	0.094	282	693	3.875	0.473	Z2_2000_0.1	0.67
12	0.759	0.062	282	693	3.210	0.493	Z2_3000_0.1	0.70
13	0.762	0.069	282	693	2.620	0.490	Z2_3500_0.1	0.69
14	0.741	0.046	282	693	2.568	0.491	Z2_4000_0.1	0.69
15	0.876	0.212	282	693	0.611	0.469	Z2_4500_0.1	0.66

Table 4: Best thresholds for the fifteen algorithms obtained while considering author PHF

3.2 ROC plot for other authors

In the following figures, the ROC plot for each of the four authors of section ?? are depicted.

```

1 rocplotesr <- ggplot(bestesr, aes(bestesr$FPR,bestesr$TPR,color=bestesr$algorithm,
   title="Algorithms - esr"))+ geom_segment(aes(x =0, y = 0, xend = 1, yend = 1),
   colour="#a5a5a5")+

```


	TPR	FPR	totP	totF	thd	dist_ax	algorithm	dif
1	0.704	0.208	4539	9211	2.351	0.351	Z2_2000_0.02	0.50
2	0.801	0.271	4539	9211	0.922	0.375	Z2_3000_0.02	0.53
3	0.798	0.278	4539	9211	0.725	0.368	Z2_3500_0.02	0.52
4	0.811	0.298	4539	9211	0.532	0.363	Z2_4000_0.02	0.51
5	0.811	0.298	4539	9211	0.439	0.363	Z2_4500_0.02	0.51
6	0.738	0.222	1816	3683	2.552	0.365	Z2_2000_0.05	0.52
7	0.827	0.277	1816	3683	1.079	0.389	Z2_3000_0.05	0.55
8	0.828	0.288	1816	3683	0.840	0.382	Z2_3500_0.05	0.54
9	0.794	0.262	1816	3683	0.814	0.376	Z2_4000_0.05	0.53
10	0.785	0.259	1816	3683	0.693	0.372	Z2_4500_0.05	0.53
11	0.783	0.257	908	1843	2.355	0.372	Z2_2000_0.1	0.53
12	0.852	0.296	908	1843	1.066	0.393	Z2_3000_0.1	0.56
13	0.833	0.291	908	1843	0.917	0.383	Z2_3500_0.1	0.54
14	0.825	0.286	908	1843	0.793	0.381	Z2_4000_0.1	0.54
15	0.818	0.285	908	1843	0.675	0.377	Z2_4500_0.1	0.53

Table 5: Best thresholds for the fifteen algorithms obtained while considering author luc

	TPR	FPR	totP	totF	thd	dist_ax	algorithm	dif
1	0.602	0.194	3432	1989	1.331	0.289	Z2_2000_0.02	0.41
2	0.716	0.368	3432	1989	0.339	0.246	Z2_3000_0.02	0.35
3	0.412	0.046	3432	1989	1.192	0.259	Z2_3500_0.02	0.37
4	0.796	0.401	3432	1989	0.145	0.280	Z2_4000_0.02	0.40
5	0.747	0.323	3432	1989	0.140	0.300	Z2_4500_0.02	0.42
6	0.488	0.105	1369	799	2.120	0.271	Z2_2000_0.05	0.38
7	0.653	0.275	1369	799	0.545	0.267	Z2_3000_0.05	0.38
8	0.738	0.343	1369	799	0.337	0.280	Z2_3500_0.05	0.40
9	0.726	0.299	1369	799	0.260	0.302	Z2_4000_0.05	0.43
10	0.756	0.315	1369	799	0.186	0.312	Z2_4500_0.05	0.44
11	0.520	0.115	685	400	2.085	0.286	Z2_2000_0.1	0.40
12	0.724	0.345	685	400	0.500	0.268	Z2_3000_0.1	0.38
13	0.756	0.343	685	400	0.375	0.293	Z2_3500_0.1	0.41
14	0.813	0.365	685	400	0.242	0.317	Z2_4000_0.1	0.45
15	0.755	0.297	685	400	0.227	0.323	Z2_4500_0.1	0.46

Table 6: Best thresholds for the fifteen algorithms obtained while considering author PhC

```

2 scale_colour_manual(values=rep(brewer.pal(12,"Paired"),times=2),name="Algorithms")+
3 scale_shape_manual(values=rep(c(15,16,3,17,18,19),each=3),name="Algorithms")+
4 geom_point(aes(shape=bestesr$algorithm,label=bestesr$algorithm),size=2)+
5 xlab("False Positive Rate") + ylab("True Positive Rate")+
6 labs("ROC")
7 rocplotesr

```

```

1 Error in eval(expr, envir, enclos): objet 'bestLuc' introuvable

```

3.3 Structure of the Graph Database

We detail here the characteristics of our Graph Database. It is constructed with *Neo4j*, its query language *Cypher* and its python interface *Py2Neo*. The

3.3.1 Nodes

We have structured our database to contain the following Node Types – Properties names are in *italic*:

- **Algorithms**

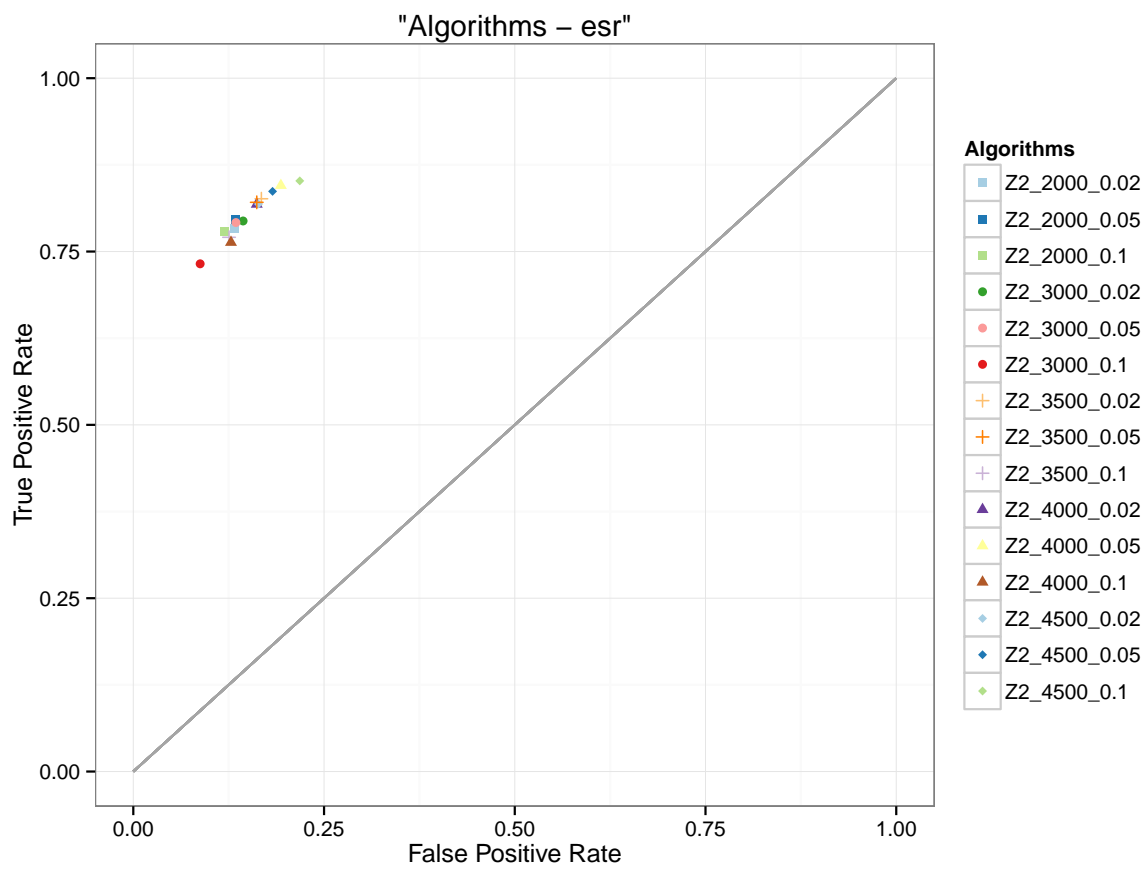


Figure 3: ROC plot of the algorithms with their best threshold, tested on author esr. The plot represents the True Positive Rate over the False Positive Rate.

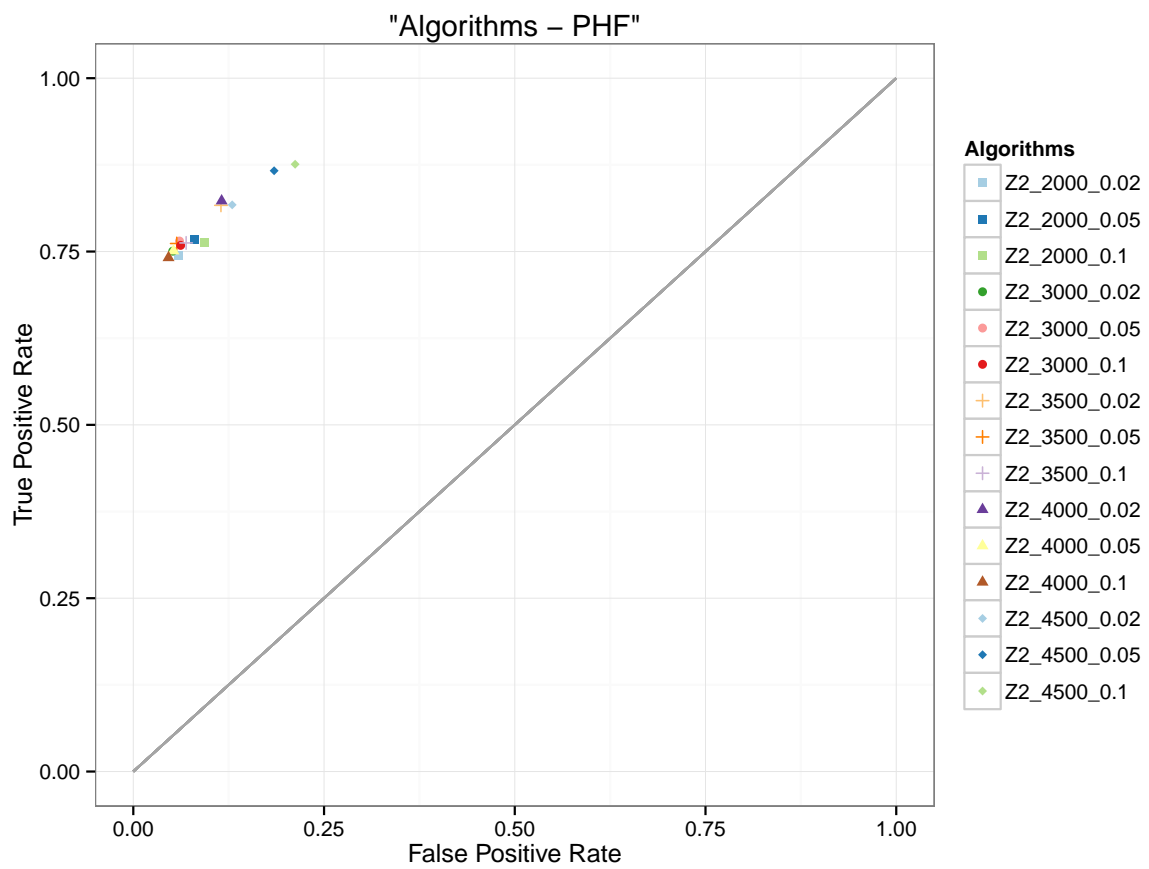


Figure 4: ROC plot of the algorithms with their best threshold, tested on author PHF. The plot represents the True Positive Rate over the False Positive Rate.

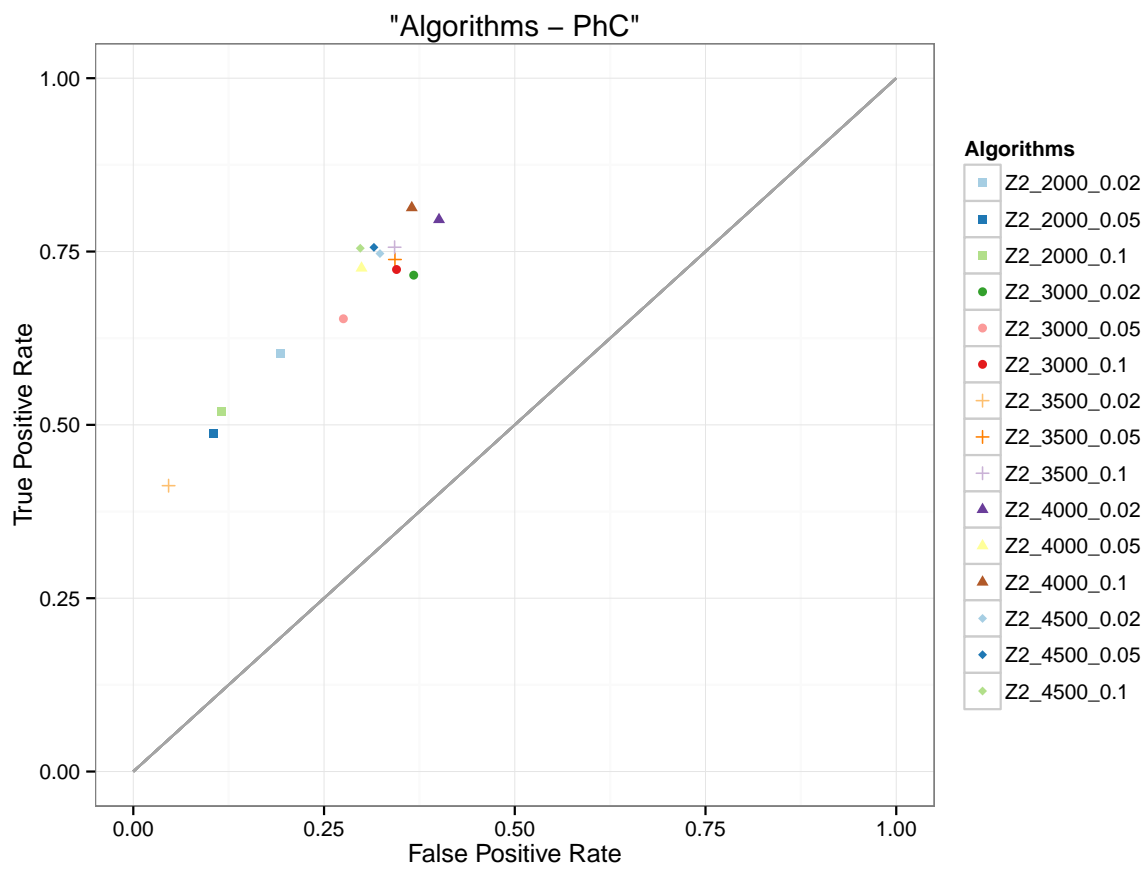


Figure 5: ROC plot of the algorithms with their best threshold, tested on author PhC. The plot represents the True Positive Rate over the False Positive Rate.

- *Name*
- *class*
- *dt* for delta time used in discretisation
- *overlap* used in the STFT
- *fmin* lower bound for low band
- *fmax* upper bound for high band
- *fc* threshold between low and high bands
- *noiseType* (detected by the algorithm)
- *threshold* applied to low/high band ratio
- **Passings** (i.e. measurement characteristics of one train passage)
 - *Name*
 - *Measurement*
 - *Temperature*
 - *Humidity*
 - *direction*
 - *rain*
 - *Wind*
 - *axleProLength*
 - *Location*
 - *v1*
 - *v2*
- **MicMes** (i.e. the measurements values recorded by one microphone at one passing)
 - *Name*
 - *micN* number of this microphone
 - *dt*
 - *Tb* start time
 - *Te* end time
 - *Tp_ b* start time masked
 - *Tp_ e* end time masked
 - *tNoise* Period in s. where noise was detected
 - *tNoisemasked* Period in s. where noise was detected (masked between *Tp_ b* and *Tp_ e*)
 - *TEL*
 - *Location*
- **TrainType**
 - *Name*
- **Location**
 - *Name*

3.3.2 Relationships

The database contains the following Relationships – the example is the canonical use of the relationship :

- **IN** (a Passing took place IN a Location)
 - *Time*
 - *Date*
 - *Track*
- **SAW** (a Passing SAW a TrainType)
 - *Time*
 - *Date*
 - *TrainLength*
 - *Track*
- **ISANEVALOF** (a MicMes is an evaluation of a Passing)
 - *tEvalmasked* total length of the masked recording
 - *tEval* total length of the recording
- **WASEVALWITH** (a MicMes was evaluated with an Algorithm)
- **NEAR** (two Locations are NEAR, e.g. Biel and Biel2)