Pulsar Detection

Francesco Scalera s292432, Riccardo Sepe s287760

July 13, 2022

Abstract

In this article, we discuss and evaluate various methods for performing binary classification tasks on the HTRU2 dataset. In particular, we first carry out PCA and feature transformation towards a Gaussian distribution, then we analyze the output from various classifiers, including Multivariate Gaussian Classifier, Linear Logistic Regression, Support Vector Machines (with various kernels), and Gaussian Mixture Models. We use the detection cost function as a metric for comparing results.

Keywords: Machine Learning, Binary Classification, Pulsar Detection, Multivariate Gaussian Classifier, Logistic Regression, Support Vector Machine, Gaussian Mixture Model

1 Introduction

The HTRU2 dataset describes a sample of pulsar candidates collected during the High Time Resolution Universe Survey (South) [1].

Pulsars are a rare type of Neutron star that produce radio emission detectable here on Earth. They are of considerable scientific interest as probes of space-time, the inter-stellar medium, and states of matter. As pulsars rotate, their emission beam sweeps across the sky, and when this crosses our line of sight, produces a detectable pattern of broadband radio emission. As pulsars rotate rapidly, this pattern repeats periodically. Thus pulsar search involves looking for periodic radio signals with large radio telescopes.

2 Dataset description

The dataset is composed by 17,898 total examples: 1,639 (9.2%) are positive (labeled as class 1) and 16,259 (90.8%) are negative (labeled as class 0). To perform our classification tasks, this dataset has been split into training and test set, with the latter pretending to be unseen data even if it is labeled in order to allow us to assess the performance of the classifier on held-out data. Each candidate (sample) is described by 8 continuous variables and a label that specifies its class.
The features represent respectively:

1. Mean of the integrated profile.

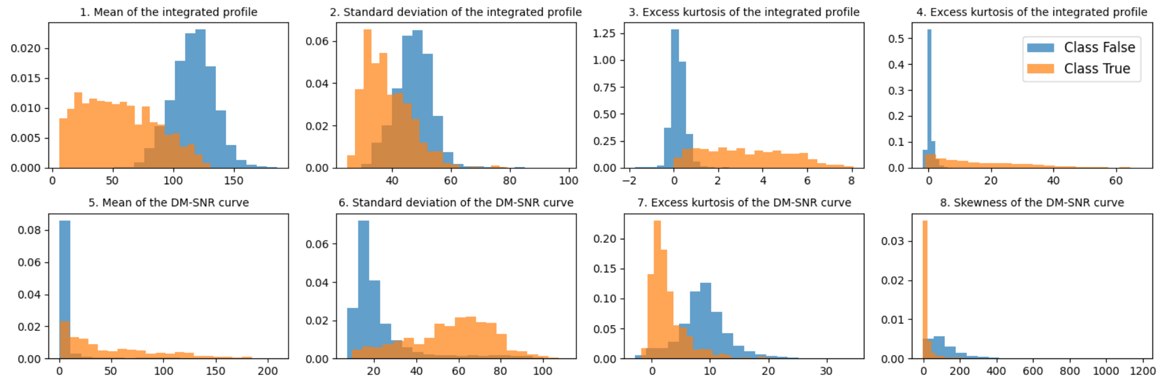

Figure 1: Histograms of the raw features of the training partition of the HTRU2 dataset.

2. Standard deviation of the integrated profile.
3. Excess kurtosis of the integrated profile.
4. Skewness of the integrated profile.
5. Mean of the DM-SNR curve.
6. Standard deviation of the DM-SNR curve.
7. Excess kurtosis of the DM-SNR curve.
8. Skewness of the DM-SNR curve.
9. Class

We decided to perform a distribution transformation towards a Gaussian one for the features of the samples in order to avoid sub-optimal results. In fact, we make use of the "gaussianization" procedure, which entails mapping the features to a uniform distribution and then transforming the mapped features using the inverse of the Gaussian cumulative distribution function. Let x be the feature we want to transform. We compute its rank over the training dataset:

$$r(x) = \frac{\sum_{i=1}^{N} \mathbb{I}[x<x_i]+1}{N+2}$$

x_i is the value of the considered feature for the $i-th$ training sample. We add 1 to the numerator and 2 to the denominator to avoid numerical issues. Then, we compute the transformed feature as

$$y = \Phi^{-1}(r(x))$$

where Φ is the inverse of the cumulative distribution function (percent point function, p.p.f) of the standard normal distribution.

In Figure 2 we can observe that, even if the gaussianization acts on the whole dataset, independently from the sample classes, the overall data distribution of both classes is anyway "more gaussian" than the original one.

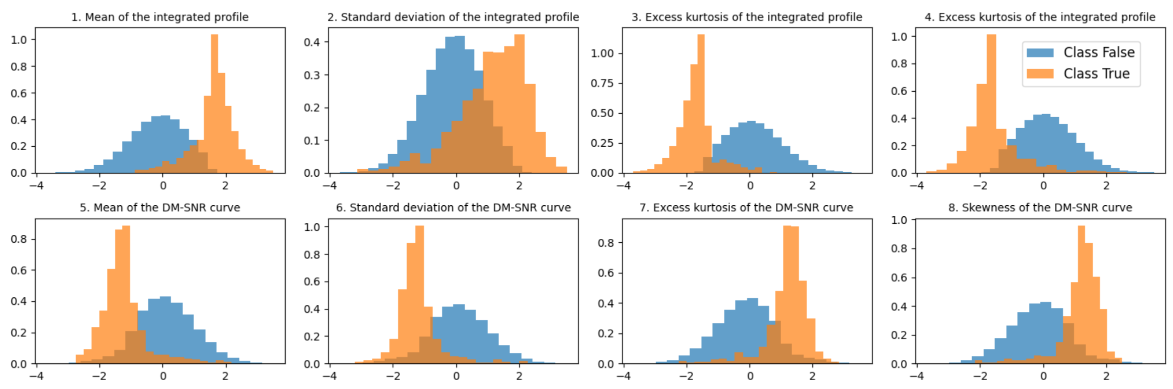

Figure 2: Histograms of the gaussianized features of the training partition of the HTRU2 dataset.

As a next step, we want to check whether any features are correlated with one another. For this purpose, we exploit a heat map that shows the absolute value (we are not interested in whether the correlation is negative or positive) of the Pearson correlation coefficient ρ between two features:

$$\rho = \left| \frac{Cov(X,Y)}{\sqrt{Var(X)}\sqrt{Var(Y)}} \right|$$

The cells with darker color tell us that the features corresponding to the indices of these cells are highly correlated. Furthermore, we can notice that on the diagonal of this map we have only 1 because it represents the correlation of each feature with itself. Comparing the top and bottom plots from Figure 3, it is possible to see that after the gaussianization, the already correlated features become more correlated among themselves and less correlated with the remaining ones, potentially making PCA more effective.

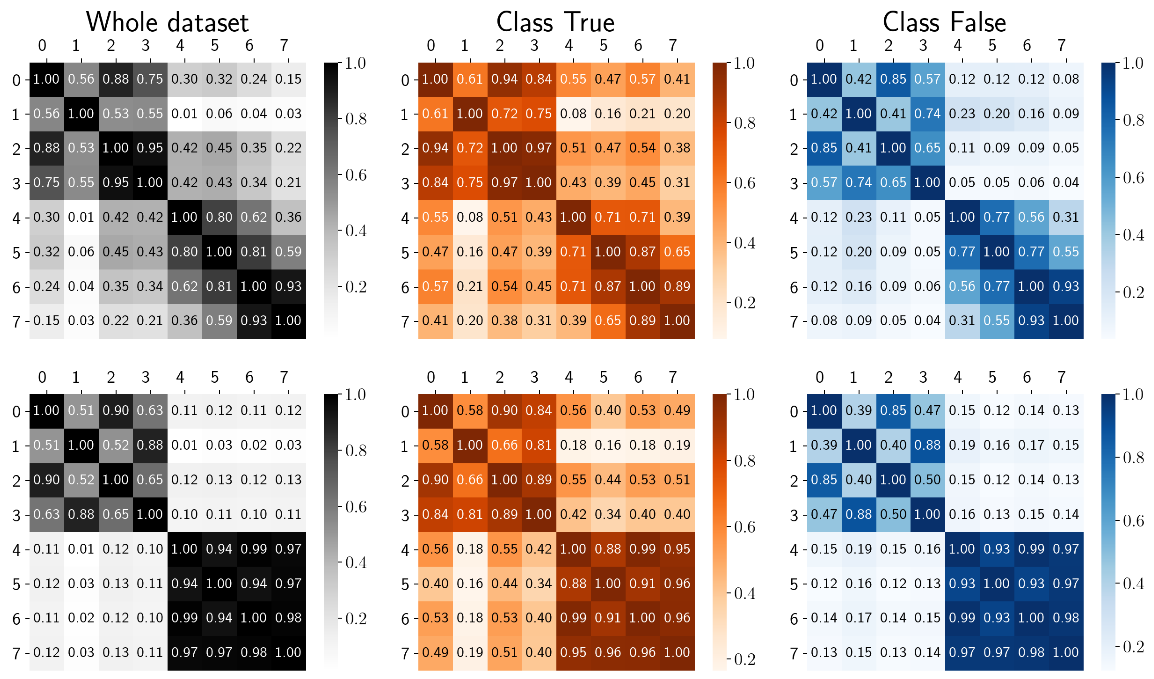

Figure 3: Correlation of the raw features (top) and gaussianized features (bottom) of the training partition of the HTRU2 dataset.

3 Project description

The core of our project is the `classifiers` folder: it contains the classes that model the classifiers we used. The base abstract class `ClassifierClass` is contained in `Classifier.py`: its subclasses can be constructed by providing the training data and labels and optionally some `**kwargs` that will vary based on which is the subclass. It exposes three abstract methods: `train_model()`, `classify()` and `get_scores()`. The latter is used to return the classifiers scores generated inside `classify()`, so it is not directly necessary for classification, but it's useful for our purpose of evaluating the performance of our model itself in the Optimal Bayes Decision (DCF computation) framework.
The `preprocessing` folder contains code useful for the pre-processing steps we considered and the `utils` folder contains some utility functions grouped by purpose: `matrix_utils`, `plot_utils` and `misc_utils`.
The `data`, `results` and `simulations` folders contain respectively the data, the results in term of $DCFs$ for each classifier and for each possible configuration and various collections of `.npy` files containing the values for the plots.
In file `main.py` is loaded the data and are called the tuning functions and simulations functions.

4 Classification of HTRU2 features

4.1 Multivariate Gaussian Classifiers

In this section, we examine Multivariate Gaussian classifiers to identify the most promising model and assess the impact of feature reduction methods (in this case, PCA) on our dataset. To perform this analysis, we employed the k-fold cross validation using 5 folds. We opted for this technique over single split validation because it provides us a larger amount of training and validation data, thus leading to more precise estimates. In Table 1 are reported the results of classification in terms of *minimum Detection Cost Function* (*min* DCF), that represents the cost we would pay to make optimal decisions on test set using the recognizer scores. We take into account three different applications, each of which is described by the triple (π_T, C_{fp}, C_{fn}) (the prior class $True$ probability, the cost of predicting as positive a sample that is actually negative, and vice versa), but which can be reduced to the single parameter $\widetilde{\pi}$, the *effective prior class True probability*. The reference application is denoted by $\widetilde{\pi} = 0.5$ while the other two are denoted by $\widetilde{\pi} = 0.1$ and $\widetilde{\pi} = 0.9$ respectively. These two values for $\widetilde{\pi}$ correspond respectively to a $\frac{C_{fp}}{C_{fn}}$ of $\frac{1}{9}$ and of 9 times the same ratio for the reference application. Classification has been performed with and without pre-processing steps (i.e. gaussianization and/or PCA) to assess whether and how these processes improve the results in terms of *min* DCF.

We can observe that the performance of the Tied Full-Cov implementation outperforms the other two. This indicates that the sample distributions for classes $True$ (the sample is actually a pulsar) and $False$ are comparable. Furthermore, we can see that on the gaussianized dataset with PCA7 and even with PCA5 the classifier performs nearly identically with the same dataset without the PCA

	Raw features			Gaussianized features		
	$\widetilde{\pi}$=0.5	$\widetilde{\pi}$=0.1	$\widetilde{\pi} = 0.9$	$\widetilde{\pi}$=0.5	$\widetilde{\pi}$=0.1	$\widetilde{\pi} = 0.9$
no PCA						
Full–Cov	0.141	0.286	0.672	0.154	0.247	0.706
Diag–Cov	0.193	0.315	0.747	0.153	0.278	0.605
Tied Full–Cov	0.112	0.224	0.573	0.131	0.235	0.537
PCA ($m = 7$)						
Full–Cov	0.171	0.300	0.714	0.154	0.244	0.691
Diag–Cov	0.219	0.632	0.821	0.162	0.248	0.662
Tied Full–Cov	0.152	0.280	0.609	0.133	0.245	0.538
PCA ($m = 5$)						
Full–Cov	0.180	0.428	0.731	0.152	0.246	0.696
Diag–Cov	0.201	0.607	0.784	0.155	0.240	0.634
Tied Full–Cov	0.178	0.313	0.600	0.136	0.249	0.542

Table 1: Results on MVG classifiers.

step. This supports our theory, which was based on the information in Figure 3. On the contrary, on the raw dataset the PCA step is quite disruptive, especially on our main application described by $\widetilde{\pi} = 0.5$. We showed that gaussianization generally makes our classifiers perform slightly worse, but it also significantly increases the model's PCA robustness.

4.2 Linear Logistic Regression

In this section we analyze Linear Logistic Regression behavior on the HTRU2 dataset. Since classes are not balanced, we resort to the prior-weighted form of the objective function:

$$J(\boldsymbol{w}, b) = \frac{\lambda}{2}||\boldsymbol{w}||^2 + \frac{\pi_T}{n_T}\sum_{i=1|c_i=1}^{n}\log(1+e^{-z_i(\boldsymbol{w}^T\boldsymbol{x}_i+b)}) + \frac{1-\pi_T}{n_F}\sum_{i=1|c_i=0}^{n}\log(1+e^{-z_i(\boldsymbol{w}^T\boldsymbol{x}_i+b)})$$

The first term's goal is to maintain control over $||\boldsymbol{w}||$, in fact if λ is too small, the risk of overfitting rises; conversely, if it is too high, the risk of underfitting rises.
Hence, we want to find a good value of λ and we want it to work well independently from the applications. In order to accomplish this, we once more turn to k-fold cross-validation, again with 5 folds, on various applications, and using various preprocessing techniques. Figure 4 displays the results.

The behavior of the model is significantly influenced by the gaussianization, as can be seen by comparing the top-left image with the other ones. In particular, this process makes the *minDCFs* outcomes significantly less dependent on the value of λ (apart from very high values). Additionally, it is possible to see that, with gaussianized features, the *minDCF* value oscillates remarkably for $10^{-6} \leq \lambda \leq 10^{-3}$ and that for $\lambda > 10^{-3}$, the metric is consistently higher than for $\lambda < 10^{-6}$. We thus decided to pick a value of $\lambda = 10^{-7}$ as it is sufficiently far from the oscillations.

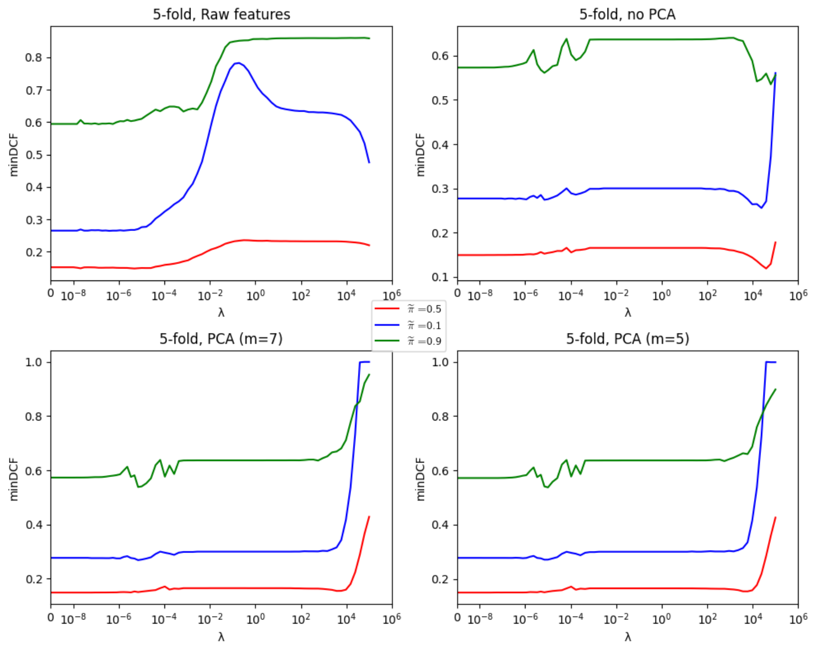

Figure 4: Tuning of LR hyperparameter λ

	$\widetilde{\pi}$=0.5	$\widetilde{\pi}$=0.1	$\widetilde{\pi}=0.9$
Raw features – no PCA			
Log Reg($\pi_T = 0.5$)	0.152	0.266	0.596
Log Reg($\pi_T = 0.1$)	0.158	0.298	0.609
Log Reg($\pi_T = 0.9$)	0.163	0.320	0.629
Gaussianized features – no PCA			
Log Reg($\pi_T = 0.5$)	0.149	0.575	0.637
Log Reg($\pi_T = 0.1$)	0.150	0.306	0.633
Log Reg($\pi_T = 0.9$)	0.165	0.299	0.636
Gaussianized features – PCA ($m = 7$)			
Log Reg($\pi_T = 0.5$)	0.149	0.276	0.575
Log Reg($\pi_T = 0.1$)	0.150	0.306	0.633
Log Reg($\pi_T = 0.9$)	0.165	0.299	0.636
Gaussianized features – PCA ($m = 5$)			
Log Reg($\pi_T = 0.5$)	0.150	0.572	0.637
Log Reg($\pi_T = 0.1$)	0.150	0.308	0.634
Log Reg($\pi_T = 0.9$)	0.165	0.299	0.636

Table 2: Results on LR classifiers.

4.3 Support Vector Machines

Now we may focus on SVM classifiers. We can again rely on k-fold cross-validation because we need to tune the hyperparameters. We test the SVM formulation both with and without class balancing to determine whether it is appropriate for our scenario. The classes in the training set are quite uneven, as was already mentioned in Section 2, thus we expect it to be helpful.
The complexity of the solution of the primal formulation of the problem is dependent on the shape of the decision boundary we are looking for, while the dual one is quadratic in its variable $\boldsymbol{\alpha}$ in any case. We thus decided to always solve the latter, modifying its formulation, that otherwise could not be solved by the L-BGFS-B algorithm. Since the variable $\boldsymbol{x}$ (one training sample) is mapped to $\widehat{\boldsymbol{x}} = [\boldsymbol{x}, K]^T$, where K is a hyperparameter of the model, the dual formulation becomes:

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \quad & \widehat{J}^D(\boldsymbol{\alpha}) = -\frac{1}{2}\boldsymbol{\alpha}^T\widehat{\boldsymbol{H}}\boldsymbol{\alpha} + \boldsymbol{\alpha}^T\mathbf{1} \\ \text{s.t.} \quad & 0 \leq \boldsymbol{\alpha}_i \leq C \quad \forall i \in \{0, ..., n\} \\ & \sum_{i=1}^{n} z_i\alpha_i = 0 \end{aligned} \tag{4.1}$$

where $\widehat{\boldsymbol{H}}$ is a matrix such that $\widehat{\boldsymbol{H}}_{ij} = z_i z_j k(\widehat{\boldsymbol{x}}_i, \widehat{\boldsymbol{x}}_j)$. The kernel function k generalizes the dot product of the linear formulation of the dual problem and embeds the shape of the decision boundary inside its definition. In the following, we analyze the behavior of three different kernels on the HTRU2 dataset:

1. Linear kernel:
$$k(\widehat{\boldsymbol{x_i}}, \widehat{\boldsymbol{x_j}}) = \widehat{\boldsymbol{x_i}}^T\widehat{\boldsymbol{x_j}} \tag{4.2}$$

2. Polynomial kernel:
$$k(\widehat{\boldsymbol{x_i}}, \widehat{\boldsymbol{x_j}}) = (\widehat{\boldsymbol{x_i}}^T\widehat{\boldsymbol{x_j}} + c)^d \tag{4.3}$$

3. Radial Basis function kernel (RBF):
$$k(\widehat{\boldsymbol{x_i}}, \widehat{\boldsymbol{x_j}}) = e^{-\gamma||\widehat{\boldsymbol{x_i}} - \widehat{\boldsymbol{x_j}}||^2} \tag{4.4}$$

The hyperparameters of the model are the aforementioned K and the parameter C from (4.1) first constraint. However, based on which kernel to use, there can also be other hyperparameters.

4.3.1 Linear kernel

Since MVG, which provides a quadratic decision rule, outperforms linear LR, we expect the model to perform better with a quadratic kernel rather than a linear one. Nonetheless, we still try both. We trained our classifier using several dataset configurations, both with unbalanced and balanced constraints, in order to determine whether the prediction was accurate. The latter consists of applying, in place of the first constraint in (4.1), ad-hoc constraints for each class:

$$0 \leq \boldsymbol{\alpha}_i \leq C_i \ \forall i \in \{0, ..., n\} \tag{4.5}$$

$$\text{where } C_i = \begin{cases} C\dfrac{\pi_T}{\pi_T^{emp}}, & \text{if } \boldsymbol{x}_i \text{ in class } True \\ C\dfrac{\pi_F}{\pi_F^{emp}}, & \text{if } \boldsymbol{x}_i \text{ in class } False \end{cases}$$

In the following we show the results for unbalanced constraints and for balanced ones; in the second case, different values of π_T have been considered.

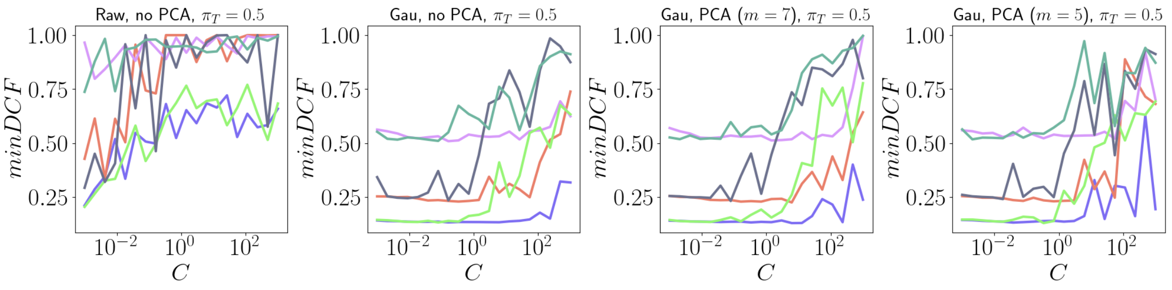

Figure 5: Results for unbalanced constraints with a linear kernel. Legend in Figure 8

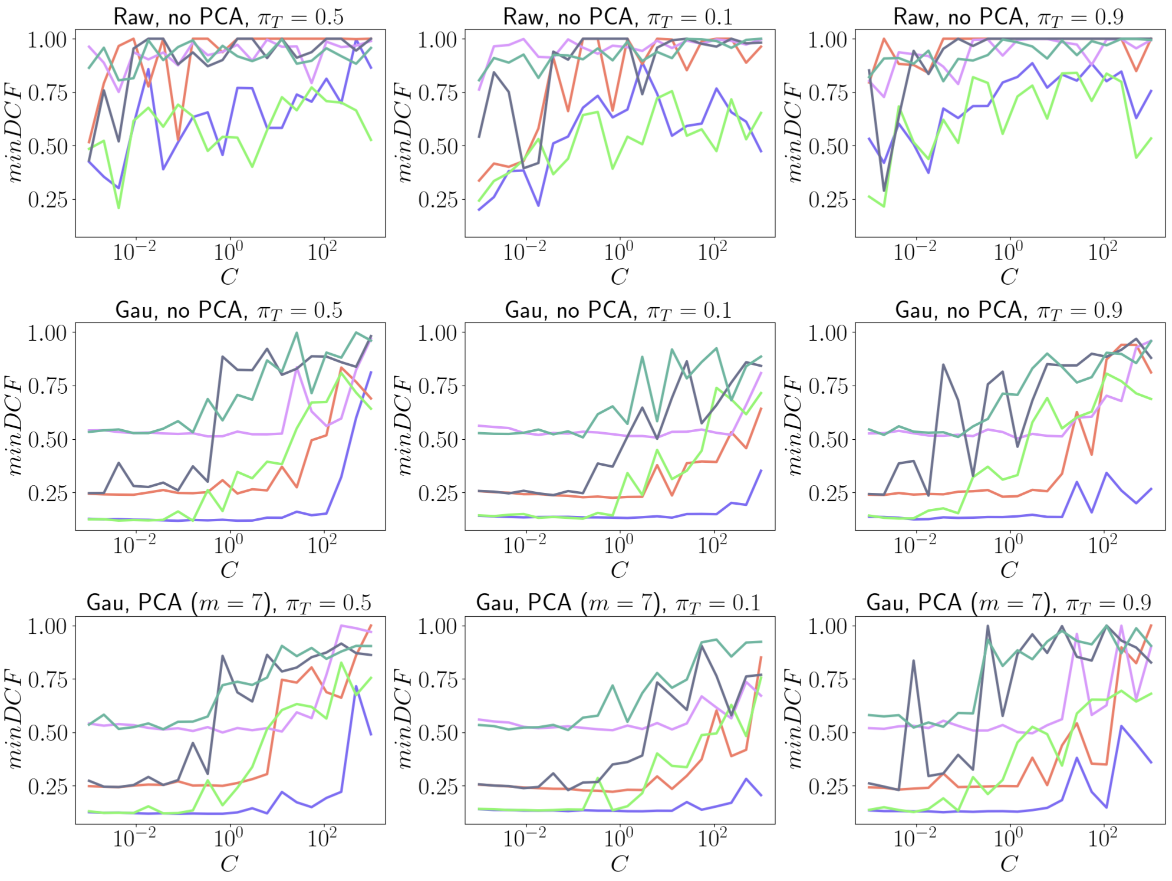

Figure 6: Results for balanced constraints with a linear kernel (1)

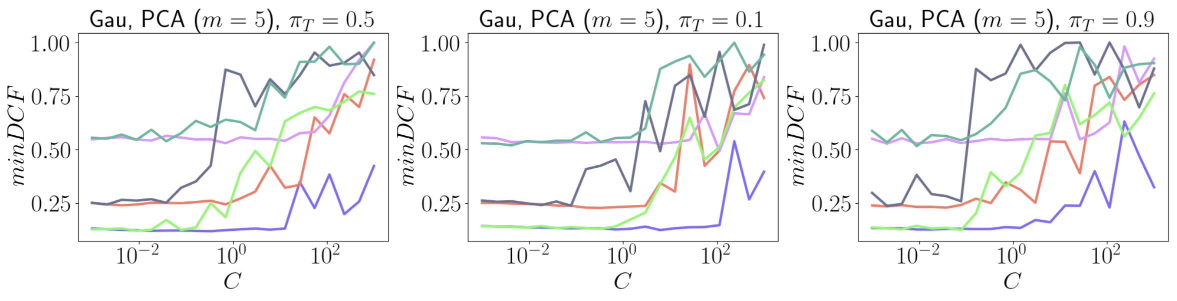

Figure 7: Results for balanced constraints with a linear kernel (2)

As expected, with gaussianized features we got significantly different (and better) results with respect to raw features. This is because SVM is not invariant under affine transformations and provides better results with centered data.
It's also possible to notice that our applications behave similarly to the unbalanced application when performing the balancing with $\pi_T = 0.1$, because in this case $\frac{\pi_c}{\pi_c^{emp}} \approx 1$, with $c \in \{T, F\}$, hence $C_i \approx C\ \forall i$.

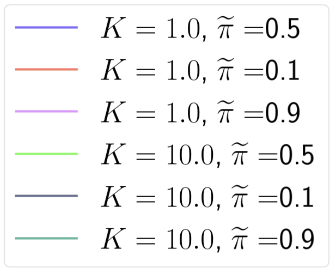
$K = 1.0$, $\widetilde{\pi} =$0.5
$K = 1.0$, $\widetilde{\pi} =$0.1
$K = 1.0$, $\widetilde{\pi} =$0.9
$K = 10.0$, $\widetilde{\pi} =$0.5
$K = 10.0$, $\widetilde{\pi} =$0.1
$K = 10.0$, $\widetilde{\pi} =$0.9

Figure 8: Legend for Figure 5 and Figures 6 and 7

	$\widetilde{\pi}$=0.5	$\widetilde{\pi}$=0.1	$\widetilde{\pi} = 0.9$
Raw features – no PCA			
Linear SVM($unbalanced, C = 1$)	0.385	0.671	0.901
Linear SVM($balanced, \pi_T = 0.5, C = 2 * 10^{-2}$)	0.297	0.614	0.850
Linear SVM($balanced, \pi_T = 0.1, C = 2 * 10^{-2}$)	0.444	0.889	0.926
Linear SVM($balanced, \pi_T = 0.9, C = 2 * 10^{-2}$)	0.532	0.934	0.962
Gaussianized features – no PCA			
Linear SVM($unbalanced, C = 1$)	0.135	0.229	0.518
Linear SVM($balanced, \pi_T = 0.5, C = 2 * 10^{-2}$)	0.123	0.243	0.539
Linear SVM($balanced, \pi_T = 0.1, C = 2 * 10^{-2}$)	0.137	0.251	0.548
Linear SVM($balanced, \pi_T = 0.9, C = 2 * 10^{-2}$)	0.132	0.240	0.523
Gaussianized features – PCA $(m = 7)$			
Linear SVM($unbalanced, C = 1$)	0.132	0.231	0.516
Linear SVM($balanced, \pi_T = 0.5, C = 2 * 10^{-2}$)	0.123	0.244	0.538
Linear SVM($balanced, \pi_T = 0.1, C = 2 * 10^{-2}$)	0.135	0.251	0.553
Linear SVM($balanced, \pi_T = 0.9, C = 2 * 10^{-2}$)	0.132	0.244	0.525
Gaussianized features – PCA $(m = 5)$			
Linear SVM($unbalanced, C = 1$)	0.131	0.238	0.522
Linear SVM($balanced, \pi_T = 0.5, C = 2 * 10^{-2}$)	0.127	0.245	0.554
Linear SVM($balanced, \pi_T = 0.1, C = 2 * 10^{-2}$)	0.141	0.246	0.540
Linear SVM($balanced, \pi_T = 0.9, C = 2 * 10^{-2}$)	0.133	0.235	0.530

Table 3: Results on Linear SVM classifiers.

Since the balancing, as expected, provided slightly better results for linear kernel, in the following subsections we only perform the validation with class balancing.

4.3.2 Polynomial kernel

We now turn our attention on polynomial kernel. In particular, we decided to test only quadratic kernel (hence $d = 2$), so we need to estimate only parameter c (see (4.3)). Here are the plots for different values of K, C and c. Unexpectedly, even if the results for quadratic and linear kernels are comparable, the latter provides slightly better results. In the raw features case, the $minDCF$ value remarkably drops down around values of $C = 10^0$. Anyways, the cost function values for $C \geq 1$ are still worse than the $minDCF$ values for the gaussianized dataset, thus confirming the effectiveness of this preprocessing step for SVM classifiers.

	$\widetilde{\pi}$=0.5	$\widetilde{\pi}$=0.1	$\widetilde{\pi} = 0.9$
Raw features – no PCA			
Polynomial SVM($\pi_T = 0.5, C = 10^{-2}, c = 15, d = 2$)	0.334	1.000	0.861
Gaussianized features – no PCA			
Polynomial SVM($\pi_T = 0.5, C = 10^{-2}, c = 15, d = 2$)	0.124	0.252	0.568
Gaussianized features – PCA ($m = 7$)			
Polynomial SVM($\pi_T = 0.5, C = 10^{-2}, c = 15, d = 2$)	0.123	0.239	0.533
Gaussianized features – PCA ($m = 5$)			
Polynomial SVM($\pi_T = 0.5, C = 10^{-2}, c = 15, d = 2$)	0.123	0.237	0.574

Table 4: Results on Poly SVM classifiers.

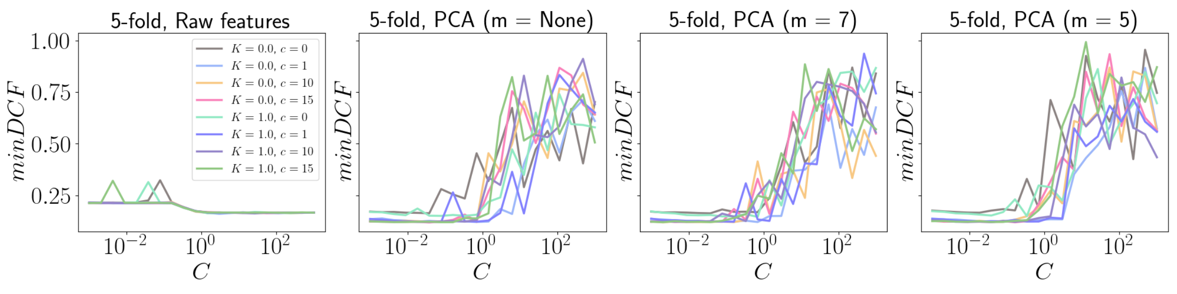

Figure 9: Results for balanced constraints with a quadratic kernel

4.3.3 Radial Basis Function kernel

We now test the performance of the SVM with the RBF kernel: in this case we need to estimate proper values of hyperparameter γ from (4.4).

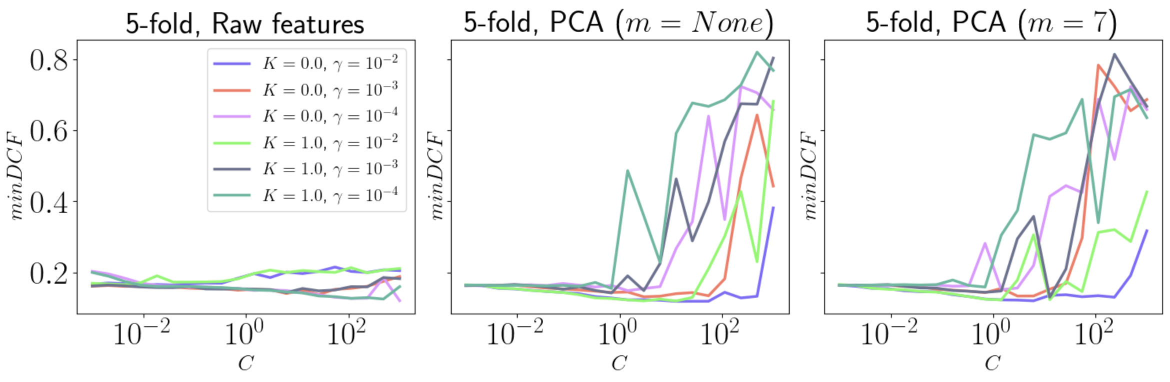

Figure 10: Results for balanced constraints with a RBF kernel

	$\widetilde{\pi}$=0.5	$\widetilde{\pi}$=0.1	$\widetilde{\pi} = 0.9$
Raw features – no PCA			
RBF SVM($\pi_T = 0.5, C = 10^{-1}, \gamma = 10^{-3}$)	0.151	0.282	0.603
Gaussianized features – no PCA			
RBF SVM($\pi_T = 0.5, C = 10^{-1}, \gamma = 10^{-3}$)	0.151	0.281	0.602
Gaussianized features – PCA $(m = 7)$			
RBF SVM($\pi_T = 0.5, C = 10^{-1}, \gamma = 10^{-3}$)	0.149	0.281	0.574

Table 5: Results on RBF SVM classifiers.

4.4 Gaussian Mixture Models

A GMM classifier is the last option we explore. We expect it to perform better than MVG, especially when working with the raw dataset, because it can model data with generic distribution. In order to model the class distributions, we must first identify a suitable number of components, possibly not just for our reference application but also for the other ones.

As we can see, the full-cov and diag models are more unstable than the tied full-cov model with respect to the number of components and, in particular, the latter is quite unaffected by PCA. Moreover, it is possible to observe that the *tied full-cov, no PCA* configuration is the only one in which the raw features outperform the pre-processed ones (bottom-left plot). The gaussianized features outperform the raw ones in the remaining cases and are generally more stable with regard to the number of components. This is actually unexpected, in fact, generally speaking, the gaussianization tends to reduce the separability of the clusters, making it more difficult to model the class distributions. There are two ways to interpret this result: either the data were already gaussian-like and the pre-processing had no impact on the data distribution (a hypothesis disproved by the histograms in section 2 and by the results with the other classifiers), or the raw data weren't distributed in well-separable clusters in the first place. The latter explanation seems more plausible. We chose a number of components equal to 16, since the plots from Figure 11 show that, using this value, the model provides good results independently from the applications. As can be seen in Table 4.4, the Full-Cov configuration outperforms the other two (meaning that the clusters of each class do not show similar within cluster covariances).

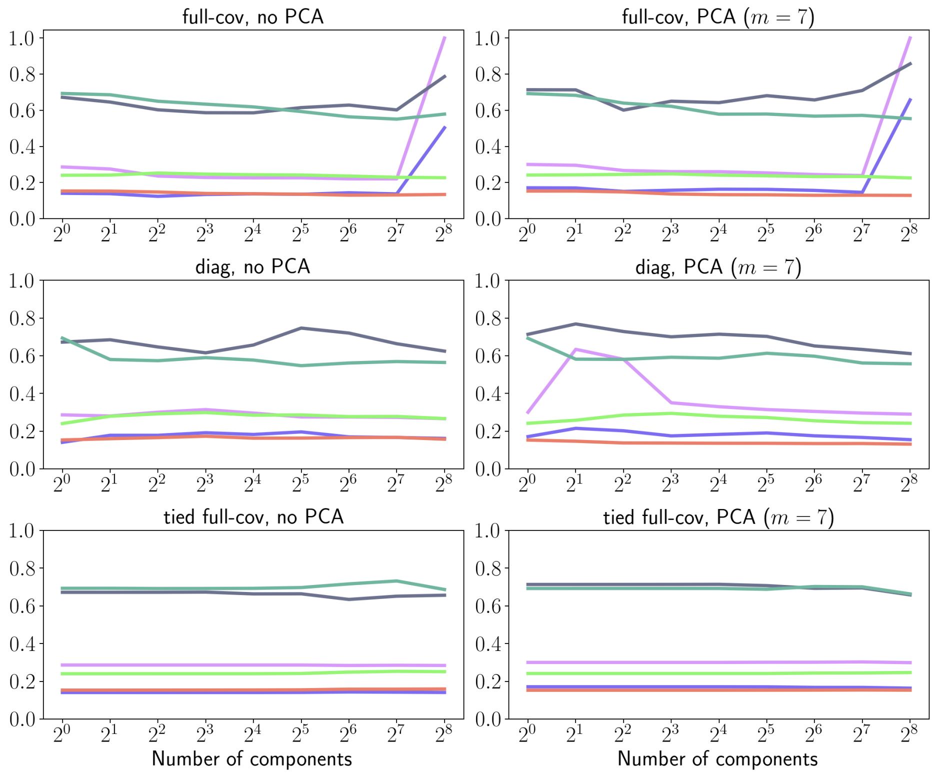

Figure 11: Tuning of number of components for a GMM classifier. Legend in figure 12

	$\widetilde{\pi}$=0.5	$\widetilde{\pi}$=0.1	$\widetilde{\pi} = 0.9$
Raw features – no PCA			
GMM, Full-Cov	0.137	0.226	0.586
GMM, Diag-Cov	0.182	0.296	0.657
GMM, Tied-Cov	0.141	0.286	0.664
Raw features – PCA $(m = 7)$			
GMM, Full-Cov	0.163	0.260	0.642
GMM, Diag-Cov	0.183	0.282	0.714
GMM, Tied-Cov	0.171	0.300	0.714
Gaussianized features – no PCA			
GMM, Full-Cov	0.138	0.243	0.619
GMM, Diag-Cov	0.163	0.330	0.577
GMM, Tied-Cov	0.154	0.240	0.693
Gaussianized features – PCA $(m = 7)$			
GMM, Full-Cov	0.133	0.241	0.578
GMM, Diag-Cov	0.136	0.279	0.587
GMM, Tied-Cov	0.153	0.242	0.693

Table 6: Results on GMM classifier.

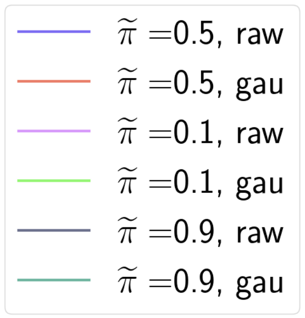
$\widetilde{\pi}$ =0.5, raw
$\widetilde{\pi}$ =0.5, gau
$\widetilde{\pi}$ =0.1, raw
$\widetilde{\pi}$ =0.1, gau
$\widetilde{\pi}$ =0.9, raw
$\widetilde{\pi}$ =0.9, gau

Figure 12: Legend for figure 11

4.5 Score calibration

	$\widetilde{\pi}$=0.5		$\widetilde{\pi}$=0.1		$\widetilde{\pi}$=0.9	
	minDCF	*actDCF*	*minDCF*	*actDCF*	*minDCF*	*actDCF*
MVG, Tied Full-Cov	0.133	0.143	0.245	0.251	0.538	0.606
Log Reg ($\pi_T = 0.5$)	0.149	0.184	0.276	0.382	0.575	0.699
Linear SVM(*balanced*)	0.120	0.123	0.250	0.244	0.521	0.538
GMM, Full-Cov, 16-G	0.133	0.140	0.242	0.286	0.578	0.658

Gaussianized features – PCA ($m = 7$)

Table 7: Comparison between most promising models.

In this subsection, we address the issue of selecting an appropriate classification threshold based on the scores provided by our recognizers. In fact, up to now, we have used the $minDCF$ as a metric for evaluating the quality of our classifiers' performance, but it only measures the cost we would pay if we made the best decisions possible based solely on the scores, whereas the actual cost we would pay depends on the comparison between the scores and the threshold. For this reason, starting from this subsection we will compute the $actualDCF$ metric. This metric will make us understand whether our recognizer's scores are well calibrated, i.e. if they provide the correct probabilistic interpretation: if this is true, the optimal threshold will be the one obtained by the mathematical formulas, i.e. the negative of the prior log-odds $t_{opt} = -\log \frac{\widetilde{\pi}}{1-\widetilde{\pi}}$.
We firstly report the results obtained using this threshold, i.e. the results obtaining on the hypothesis (untrue in general) that our scores are actually well-calibrated; then we calibrate them and compare the results.

As we can see from the plots, while generative models (i.e. MVG and GMM) produce nearly flawless results and LR (discriminative model) produces good results, the scores produced by SVM do not behave as well as the other ones, as expected. This is because the first three models produce scores with a correct probabilistic interpretation, while SVM ones have a geometric interpretation. To address this problem, we can either

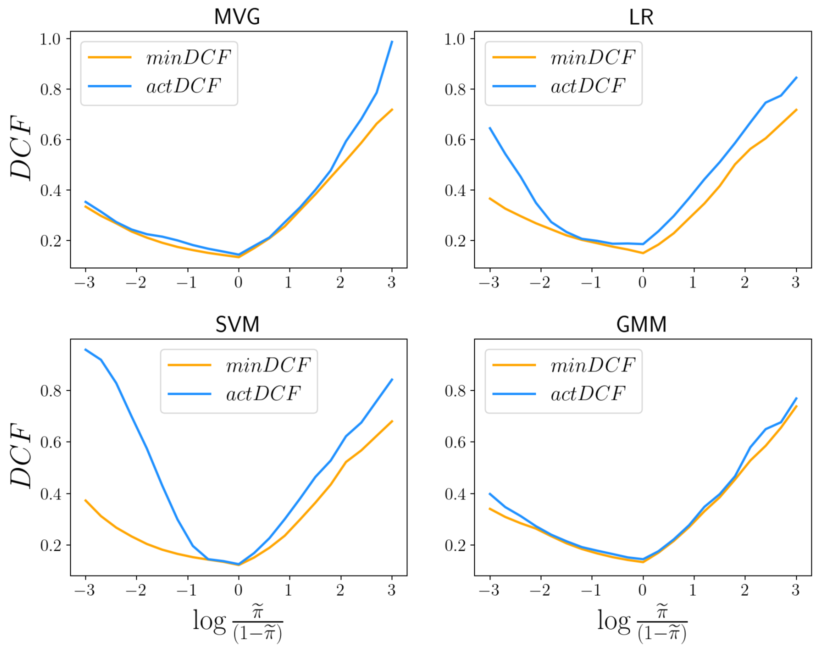

Figure 13: Minimum DCF vs actual DCF before score calibration.

resort to cross-validation to find a good threshold (i.e. different from t_{opt}, in general), or apply some function to our scores in order to transform them. Unfortunately, the first method requires to apply the CV for any application, because, with the "raw" scores produced by our classifiers, the threshold search is highly application-specific. The second method, instead, even though it's still mathematically dependent on the application (i.e. on the effective prior probability $\widetilde{\pi}$), turned out to work better with $\widetilde{\pi} = 0.5$ about 55% of the cases (not showed here): for this reason, we performed the calibration using this value for the effective prior. So now our problem becomes finding a function $f(s)$ which, applied on the scores s, produces well-calibrated scores $\widehat{s}$. We want f to be monotone, so that low values and high values of the score still correspond to different classes for the samples which produced those scores. For simplicity, we assume that f is an affine function, which means that our scores will just be scaled and/or shifted.

$$\widehat{s} = f(s) = \alpha s + \beta$$

Therefore, we must estimate the parameters α and β, and we may accomplish this by using linear logistic regression. Thus, we divided the 1-dimensional dataset of the scores into two partitions using a single 2:1 split, trained an LR on the training partition (in this context, the calibration set), and then applied the transformation to the remaining partition.

The results are shown in the bayes error plots with the *actual DCF* after score calibration (dashed line) in Figure 14.

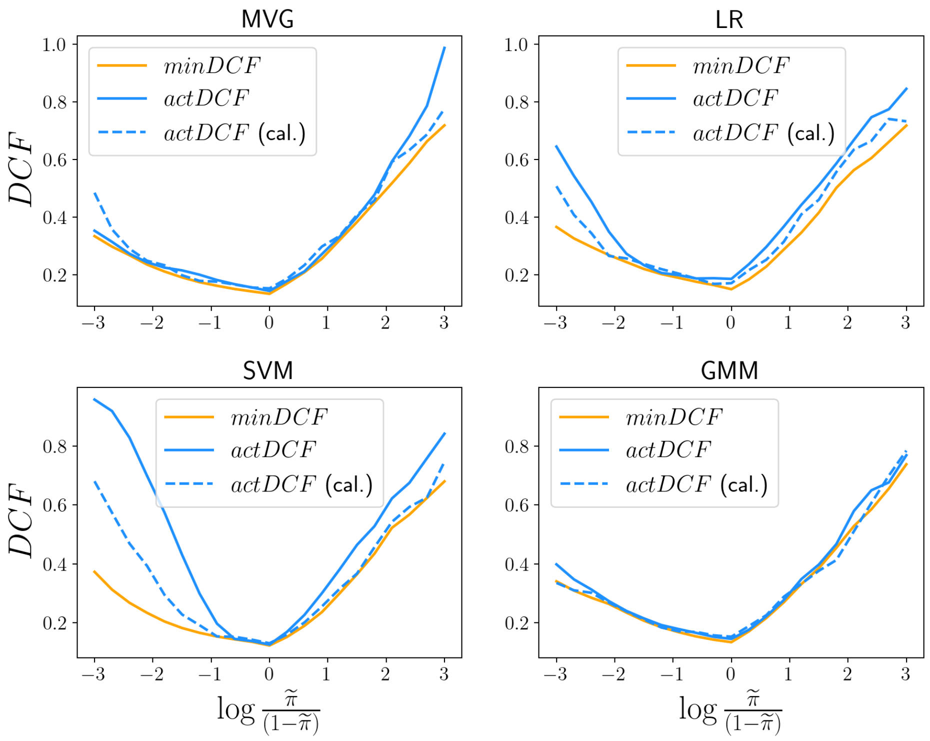

Figure 14: Minimum DCF vs actual DCF before score calibration.

5 Experimental results

Finally, in this section, we can evaluate how well our classifier performed on unseen data. The goal is to determine whether the classification task was performed successfully or instead its inference ability cannot be generalized well. To accomplish this, we will now train all the models using the hyperparameters tuned during the training phase on the entire training dataset and then evaluate the performance using the labeled samples gathered in the test set.

	$\widetilde{\pi}$=0.5	$\widetilde{\pi}$=0.1	$\widetilde{\pi} = 0.9$
Raw features – no PCA			
Full-Cov	0.140	0.283	0.646
Diag-Cov	0.185	0.330	0.621
Tied-Full Cov	0.110	0.207	0.591
Gaussianized features – no PCA			
Full-Cov	0.151	0.242	0.620
Diag-Cov	0.154	0.285	0.574
Tied-Full Cov	0.126	0.222	0.540
Gaussianized features – PCA $(m = 7)$			
Full-Cov	0.151	0.246	0.642
Diag-Cov	0.154	0.249	0.610
Tied-Full Cov	0.126	0.229	0.522
Gaussianized features – PCA $(m = 5)$			
Full-Cov	0.149	0.255	0.627
Diag-Cov	0.148	0.254	0.588
Tied-Full Cov	0.129	0.241	0.522

Table 8: Results on MVG classifiers (evaluation).

	$\widetilde{\pi}$=0.5	$\widetilde{\pi}$=0.1	$\widetilde{\pi} = 0.9$
Raw features – no PCA			
Log Reg($\pi_T = 0.5$)	0.144	0.252	0.597
Log Reg($\pi_T = 0.1$)	0.143	0.272	0.643
Log Reg($\pi_T = 0.9$)	0.145	0.318	0.690
Gaussianized features – no PCA			
Log Reg($\pi_T = 0.5$)	0.143	0.272	0.524
Log Reg($\pi_T = 0.1$)	0.153	0.298	0.572
Log Reg($\pi_T = 0.9$)	0.163	0.308	0.581
Gaussianized features – PCA $(m = 7)$			
Log Reg($\pi_T = 0.5$)	0.143	0.272	0.524
Log Reg($\pi_T = 0.1$)	0.153	0.298	0.572
Log Reg($\pi_T = 0.9$)	0.163	0.308	0.581
Gaussianized features – PCA $(m = 5)$			
Log Reg($\pi_T = 0.5$)	0.144	0.273	0.524
Log Reg($\pi_T = 0.1$)	0.153	0.298	0.571
Log Reg($\pi_T = 0.9$)	0.163	0.308	0.582

Table 9: Results on LR classifiers (evaluation).

The results on the test set are in line with what we expected. However, we performed also a new estimation of the hyperparameters to observe if we made a good choice during the validation phase by making a comparison with values obtained during the validation. The results for each classifier are shown in the following tables and plots.

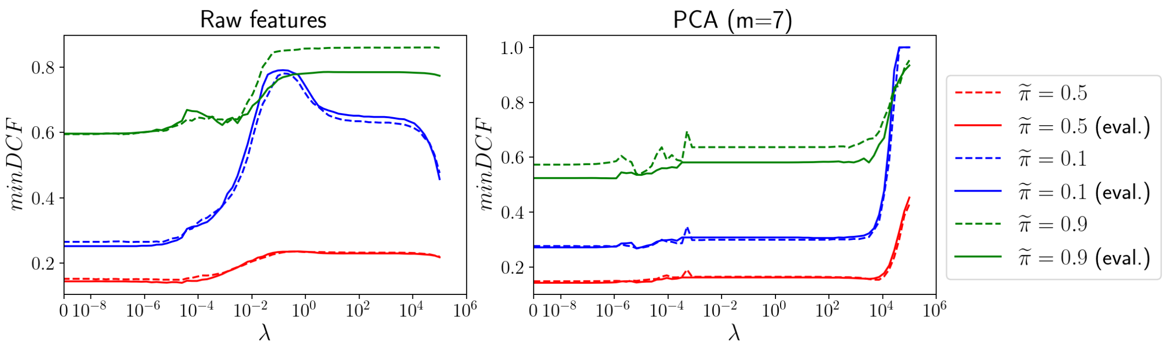

Figure 15: Comparison between $minDCF$ for different values of λ on validation and evaluation set.

	$\widetilde{\pi}$=0.5	$\widetilde{\pi}$=0.1	$\widetilde{\pi} = 0.9$
Raw features – no PCA			
Linear SVM($unbalanced, C = 1$)	0.347	0.519	0.980
Linear SVM($balanced,\ \pi_T = 0.5,\ C = 2 * 10^{-2}$)	0.224	0.374	0.841
Linear SVM($balanced,\ \pi_T = 0.1,\ C = 2 * 10^{-2}$)	0.131	0.251	0.653
Linear SVM($balanced,\ \pi_T = 0.9,\ C = 2 * 10^{-2}$)	0.230	0.340	0.994
Gaussianized features – no PCA			
Linear SVM($unbalanced, C = 1$)	0.125	0.208	0.496
Linear SVM($balanced,\ \pi_T = 0.5,\ C = 2 * 10^{-2}$)	0.115	0.233	0.479
Linear SVM($balanced,\ \pi_T = 0.1,\ C = 2 * 10^{-2}$)	0.132	0.235	0.510
Linear SVM($balanced,\ \pi_T = 0.9,\ C = 2 * 10^{-2}$)	0.123	0.208	0.473
Gaussianized features – PCA ($m = 7$)			
Linear SVM($unbalanced, C = 1$)	0.122	0.208	0.476
Linear SVM($balanced,\ \pi_T = 0.5,\ C = 8 * 10^{-2}$)	0.115	0.231	0.478
Linear SVM($balanced,\ \pi_T = 0.1,\ C = 8 * 10^{-2}$)	0.132	0.235	0.513
Linear SVM($balanced,\ \pi_T = 0.9,\ C = 8 * 10^{-2}$)	0.123	0.208	0.473
Gaussianized features – PCA ($m = 5$)			
Linear SVM($unbalanced, C = 1$)	0.127	0.216	0.492
Linear SVM($balanced, \pi_T = 0.5, C = 8 * 10^{-2}$)	0.120	0.240	0.487
Linear SVM($balanced, \pi_T = 0.1, C = 8 * 10^{-2}$)	0.135	0.233	0.509
Linear SVM($balanced, \pi_T = 0.9, C = 8 * 10^{-2}$)	0.127	0.223	0.494

Table 10: Results on Linear SVM classifiers.

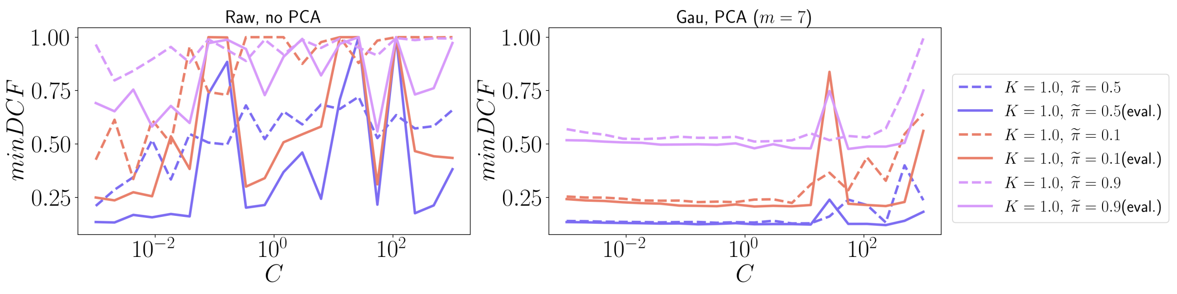

Figure 16: Comparison between $minDCF$ for different different values of C on evaluation and validation set for the linear unbalanced SVM.

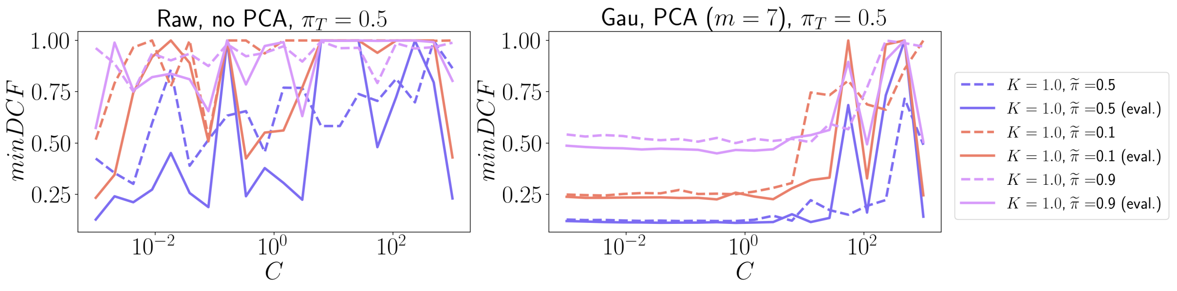

Figure 17: Comparison between $minDCF$ for different different values of C on evaluation and validation set for the linear balanced SVM.

	$\widetilde{\pi}$=0.5	$\widetilde{\pi}$=0.1	$\widetilde{\pi} = 0.9$
Raw features – no PCA			
Quadratic SVM$(\pi_T = 0.5, C = 10^{-2}, c = 15)$	0.217	0.599	0.777
Gaussianized features – no PCA			
Quadratic SVM$(\pi_T = 0.5, C = 10^{-2}, c = 15)$	0.113	0.222	0.491
Gaussianized features – PCA $(m = 7)$			
Quadratic SVM$(\pi_T = 0.5, C = 10^{-2}, c = 15)$	0.112	0.228	0.471
Gaussianized features – PCA $(m = 5)$			
Quadratic SVM$(\pi_T = 0.5, C = 10^{-2}, c = 15)$	0.113	0.231	0.509

Table 11: Results on Poly SVM classifiers (evaluation).

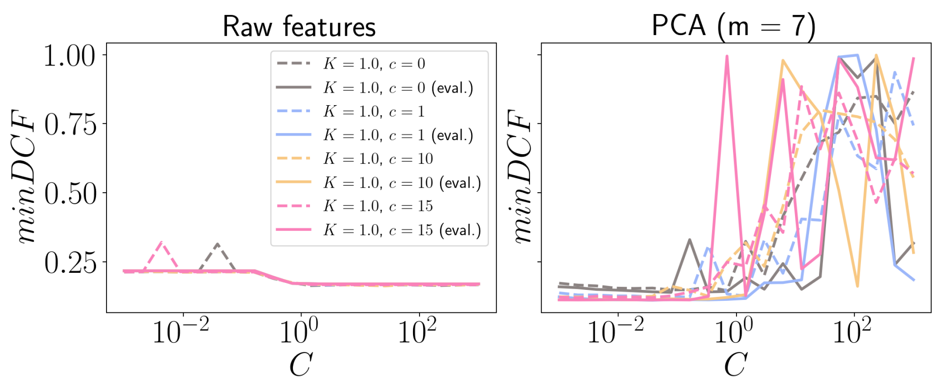

Figure 18: Comparison between $minDCF$ for different values of c and C on validation and evaluation set.

	$\widetilde{\pi}$=0.5	$\widetilde{\pi}$=0.1	$\widetilde{\pi} = 0.9$
Raw features – no PCA			
RBF SVM($\pi_T = 0.5, C = 10^{-1}, \gamma = 10^{-3}$)	0.146	0.278	0.532
Gaussianized features – no PCA			
RBF SVM($\pi_T = 0.5, C = 10^{-1}, \gamma = 10^{-3}$)	0.146	0.278	0.532
Gaussianized features – PCA ($m = 7$)			
RBF SVM($\pi_T = 0.5, C = 10^{-1}, \gamma = 10^{-3}$)	0.146	0.279	0.532

Table 12: Results on RBF SVM classifiers.

Also in this phase we analyze our result in terms of minimum DCFs, that measures the potential capabilities of our systems to produce good decisions. Thus, as we did previously, we're now introducing the *actual* DCF, that allows us to understand if our scores are well-calibrated to make a decision based on an optimal score treshold, obtained by $t_{opt} = -\log \frac{\widetilde{\pi}}{1-\widetilde{\pi}}$.

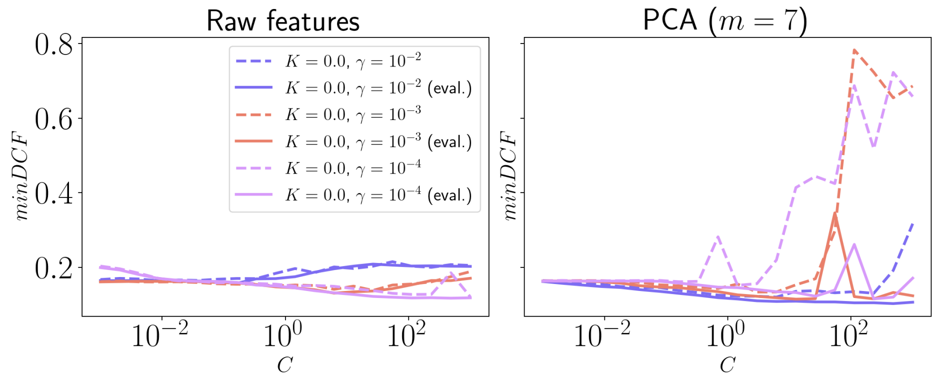

Figure 19: Comparison between $minDCF$ for different values of γ and C on validation and evaluation set.

	Raw features - no PCA			Raw features - PCA $(m = 7)$		
	$\widetilde{\pi}$=0.5	$\widetilde{\pi}$=0.1	$\widetilde{\pi} = 0.9$	$\widetilde{\pi}$=0.5	$\widetilde{\pi}$=0.1	$\widetilde{\pi} = 0.9$
GMM, Full-Cov	0.125	0.242	0.591	0.149	0.257	0.609
GMM, Diag-Cov	0.188	0.284	0.654	0.188	0.338	0.661
GMM, Tied-Cov	0.140	0.283	0.646	0.161	0.301	0.664
	Gaussianized features - no PCA			Gaussianized features - PCA $(m = 7)$		
GMM, Full-Cov	0.131	0.237	0.592	0.127	0.233	0.544
GMM, Diag-Cov	0.140	0.281	0.534	0.127	0.264	0.485
GMM, Tied-Cov	0.146	0.244	0.606	0.148	0.246	0.615

Table 13: Results on GMM classifier.

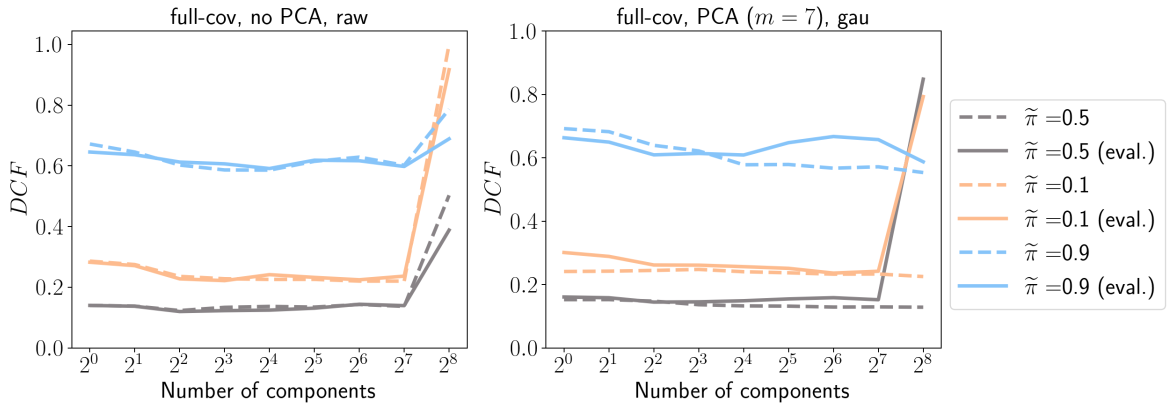

Figure 20: Comparison between *minDCF* for different GMM number of components on validation and evaluation set.

	Gaussianized features – PCA $(m = 7)$					
	$\widetilde{\pi}$=0.5		$\widetilde{\pi}$=0.1		$\widetilde{\pi}$=0.9	
	minDCF	*actDCF*	*minDCF*	*actDCF*	*minDCF*	*actDCF*
MVG, Tied Full-Cov	0.126	0.140	0.229	0.231	0.522	0.553
Log Reg $(\pi_T = 0.5)$	0.143	0.172	0.272	0.281	0.524	0.480
Linear SVM(*balanced*)	0.115	0.126	0.231	0.763	0.478	0.668
GMM, Full-Cov, 16-G	0.127	0.138	0.233	0.269	0.544	0.584

Table 14: Comparison between *minDCF* and *actDCF* for the most promising models.

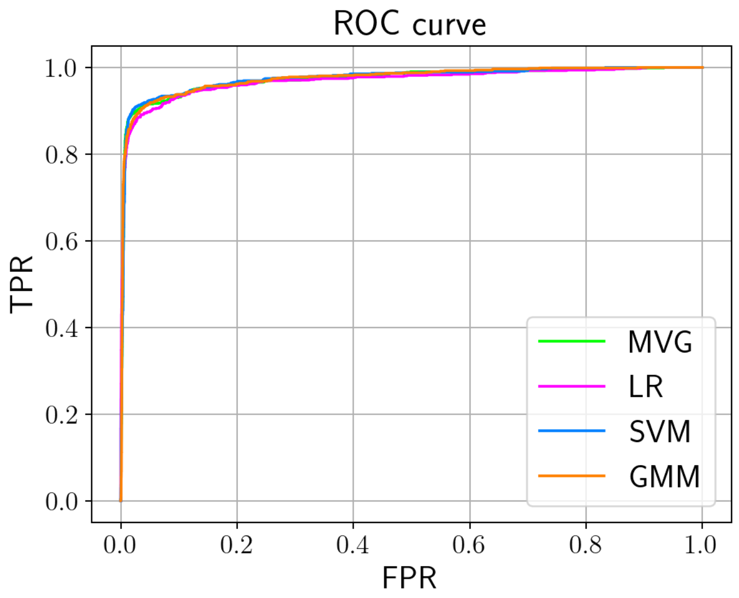

Figure 21: ROC curve for the best models.

Now we want to find again a calibration function f, so we resort again to Logistic Regression. According to the Bayes Error Plot, the *act DCF* calculated using well-calibrated scores is more similar to the *min DCF* rather than the *act DCF* calculated with "raw" scores, meaning that the decisions taken after the calibration are nearly optimal.

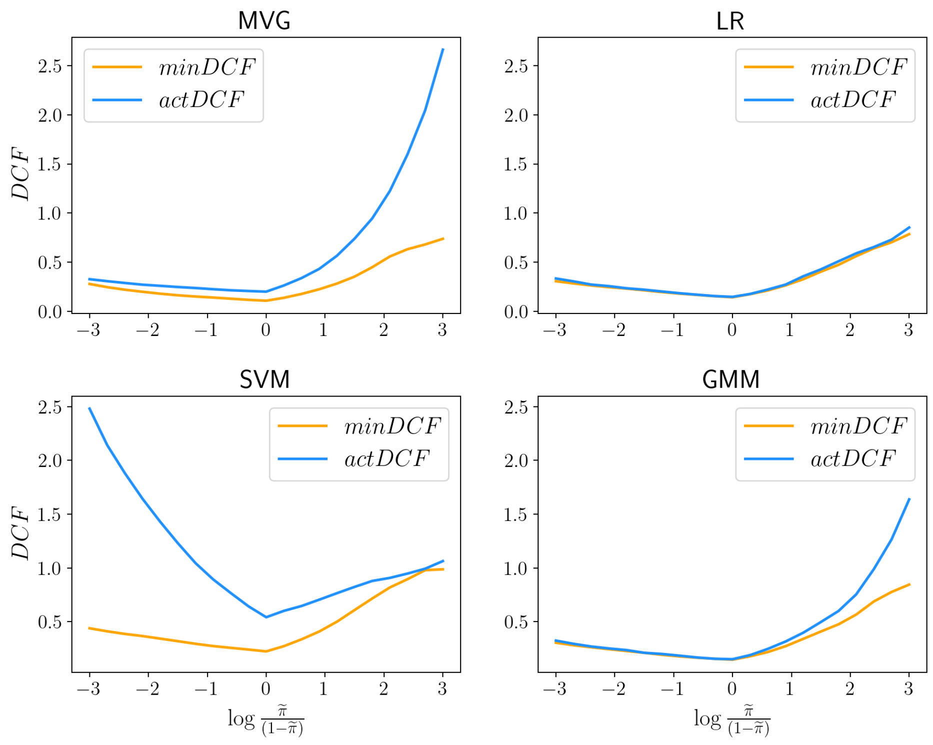

Figure 22: Minimum DCF vs actual DCF before score calibration.

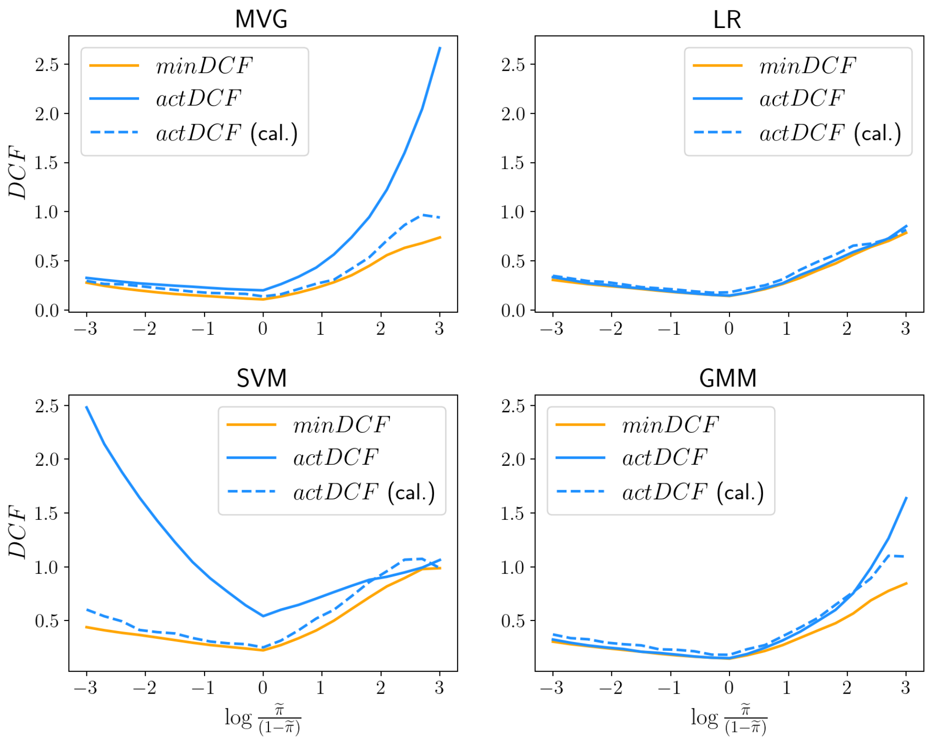

Figure 23: Minimum DCF vs actual DCF after score calibration.

6 Conclusions

Our analysis tells us that, in conjunction with gaussianization pre-processing and PCA7, the best performing model is MVG with tied covariance matrix, which provides for the reference application the best result in terms of DCF (0.140), followed by Linear LR and GMM, which for the reference application provide comparable results (respectively 0.178 and 0.183). This result can be explained by the gaussianization step: as previously mentioned in section 2, even though this technique doesn't guarantee a gaussian distribution for each class, for our specific scenario it still did. As a consequence, the gaussian assumption for the classes distribution made from MVG is quite realistic. The worst model by far is Linear SVM, with a DCF equal to 0.252 for $\tilde{\pi} = 0.5$. This finding can be explained by the fact that, even though the SVM classifier performed best with the linear kernel, the other classifiers showed that generally speaking the classes are not linearly separable. Concluding, the results showed in section 5 confirm the good quality of our decisions performed during the training/validation phase.

References

[1] M. J. K. et al. The High Time Resolution Universe Pulsar Survey - I. System Cconfiguration and Initial Discoveries. *Monthly Notices of the Royal Astronomical Society*, 409:619–627, 2010.