WINE QUALITY DETECTION

MACHINE LEARNING REPORT

Author: Giuseppe Ruggeri, s292459

Introduction to the task

In this report we are going to conduct an analysis of different Machine Learning methods that can be applied to solve classification problems; in particular, we focus on the binary classification task.

The selected dataset is called Wine Quality and it contains wine data including red and white variants of the Portuguese "Vinho Verde" wine. The original dataset is composed of two datasets (red and white variants) and it contains 4898 wine samples represented by 11 physicochemical features and one output feature representing the wine quality from 0 to 10. The dataset is available from the UCI machine learning repository (UCI, 2015).

This dataset can be employed either for classification or regression task; moreover, the 11 features may be correlated, thus it could be interesting experimenting feature selection methods to remove redundant or harmful dimensions.

As we are interested in solving a binary classification problem, the output quality classes have been binarized. Specifically, the output feature has been transformed according to the following:

$$\begin{cases} 0\ (Low\ Quality) & if\ Quality \leq 5 \\ 1\ (High\ Quality) & if\ Quality \geq 7 \end{cases}$$

The two datasets have been merged in a single dataset and the wine samples with quality equal to 6 have been removed to simplify the problem.

The 11 + 1 features of the dataset are:

1. fixed acidity
2. volatile acidity
3. citric acid
4. residual sugar
5. chlorides
6. free sulfur dioxide
7. total sulfur dioxide
8. density
9. pH
10. sulphates
11. alcohol
12. quality (0 or 1)

Dataset Analysis

In this study, the dataset has been split in two datasets, a training dataset and a held-out test dataset to perform evaluation.

To guide the study, we first conduct a dataset analysis to understand which methods could be useful to try. To do a fair study, all the analyses have been done on the training dataset only, keeping the test dataset out and only for the final evaluation.

From now on, the Low-quality class will be referred as False Hypothesis (Hf) while the High-quality class will be the True Hypothesis (Ht).

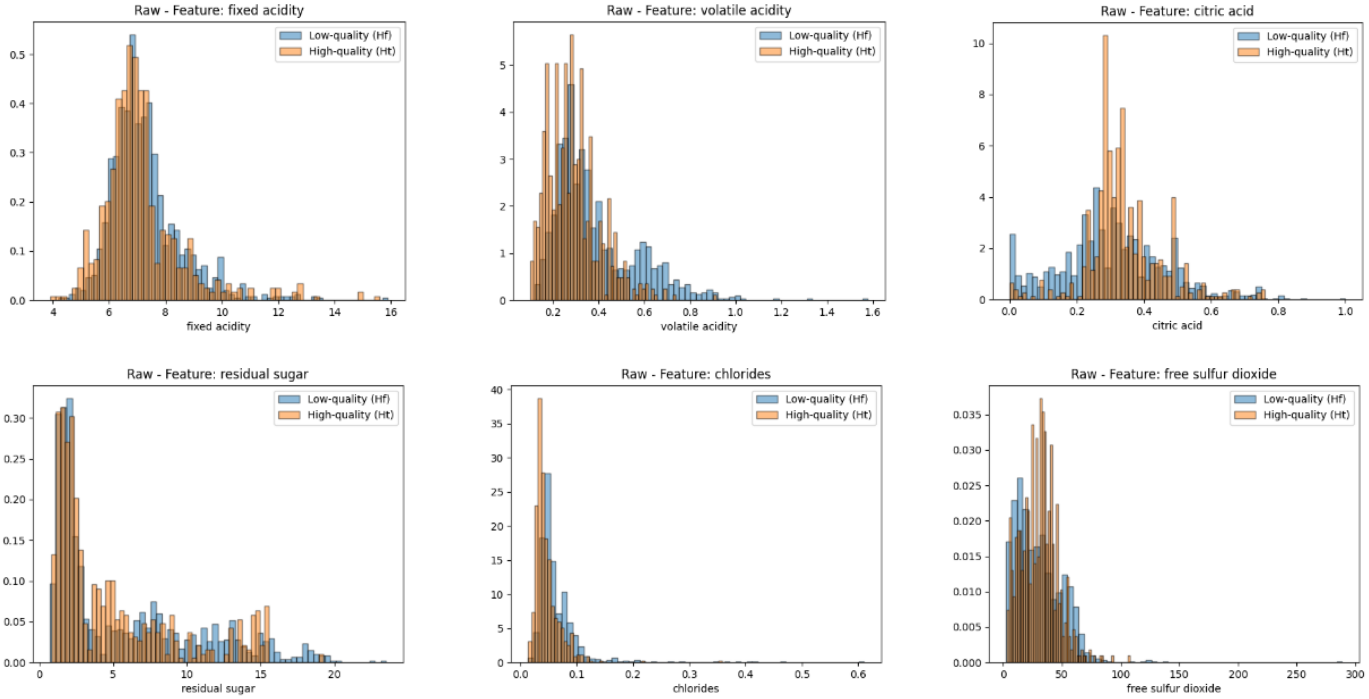
The training dataset is composed of 1839 samples, in particular 1226 are low quality wine samples, and 613 are high quality ones, showing that the empirical true class prior probability is 0.33.

The test dataset is composed of 1822 samples, 1158 are low quality wine samples while 664 are high quality ones.

The ranges of the features on the training dataset are reported in the table, furthermore, we visualize the histograms of the features to have a first approximation of the data distribution.

Feature	Min	Max	Mean	Var
Fixed acidity (FA)	3.900	15.90	7.258	1.833
Volatile acidity (VA)	0.100	1.58	0.355	0.0289
Citric acid (CA)	0.000	1.00	0.316	0.0220
Residual sugar (RS)	0.700	23.5	5.44	22.5
Chlorides (CH)	0.0130	0.611	0.0576	0.00137
Free sulfur dioxide (FSD)	2.000	289.0	30.15	369.5
Total sulfur dioxide (TSD)	7.0000	440.00	116.30	3374.4
Density (DE)	0.9874	1.003	0.9949	0.00000865
pH	2.79	3.90	3.21	0.0254
Sulphates (SU)	0.2500	1.360	0.5284	0.02042
Alcohol (AL)	8.000	14.90	10.38	1.501

Table 1 Training dataset features ranges.
The units are: FA: g(tartaric acid)/dm3; VA: g(acetic acid)/dm3; CA: g/dm3; RS: g/dm3; CH: g(sodium chloride)/dm3; FSD: mg/dm3; TSD: mg/dm3; DE: g/dm3; SU: g(potassium sulphate)/dm3; AL: %vol.



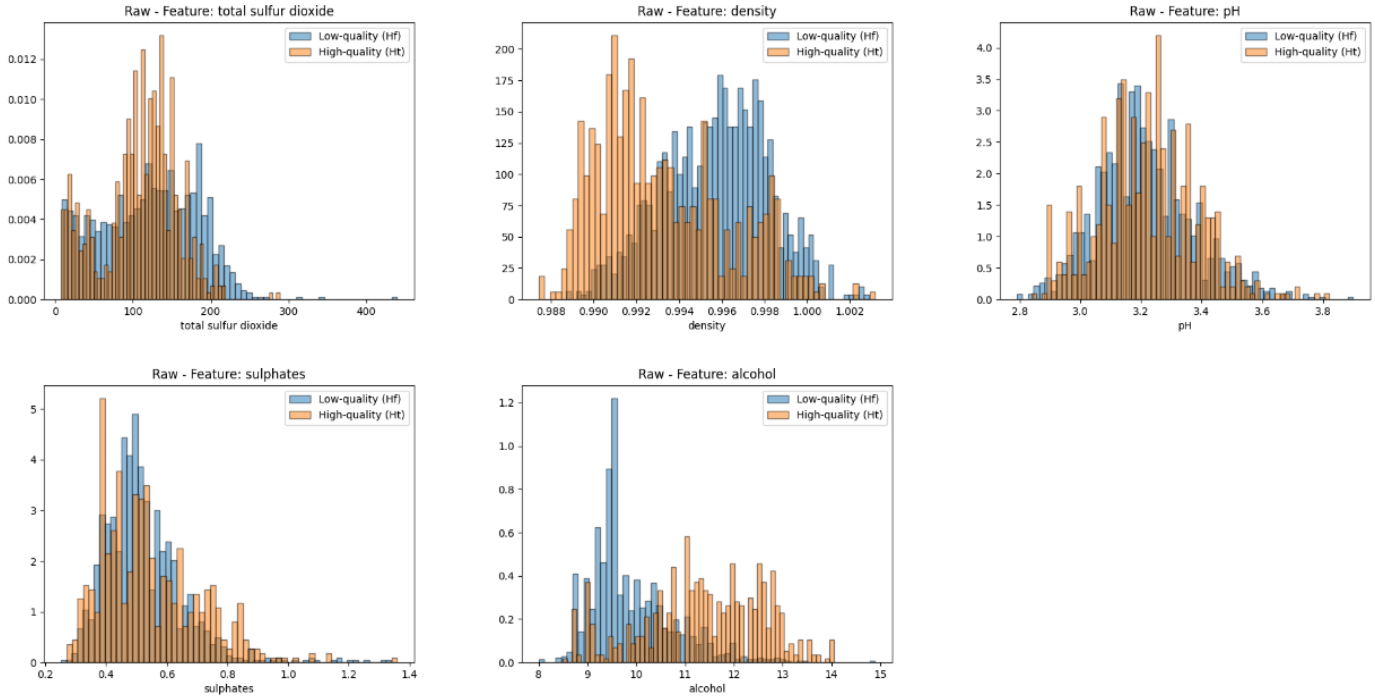


Figure 1 - Histograms of raw features for each dimension.

As the histograms show, the distributions of the data along the different dimensions are in many cases irregular and there are in some cases large outliers. This may be an unneglectable signal that all the ML methods which rely on the assumption of Gaussian class-conditional distributions may perform poorly.

Gaussianization

We now try to tackle this problem by Gaussianizing the data along the different dimensions. The Gaussianization is a procedure that allows mapping a set of features to values whose empirical cumulative distribution function is well approximated by a Gaussian c.d.f. The method consists in mapping the features to a uniform distribution and then transforming the mapped features through the inverse of Gaussian c.d.f.

As the histograms of the features after Gaussianization show, they are now better approximated by a Gaussian distribution.

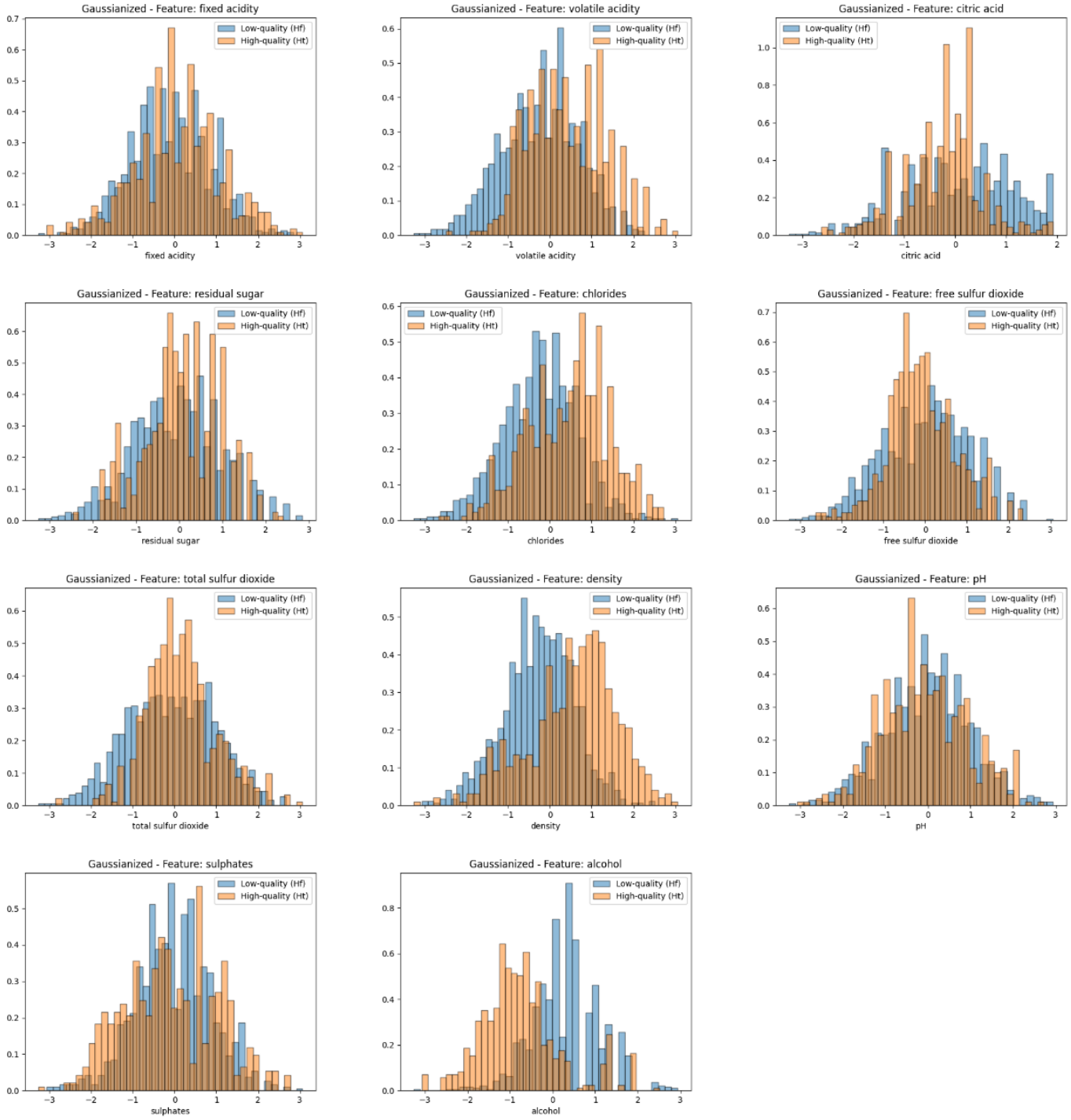


Figure 2 - Histograms of the features after the Gaussianization. NB: they are visualized as diverse plots for each class but remember that the Gaussianization takes place at dimension-level, without considering the classes separately.

However, it is worth noting that the Gaussianization does not take into account classes, therefore, even if the features may be Gaussian distributed, class-conditional distributions may be much more different than a Gaussian distribution.

Correlation analysis

We are now interested in the correlation between different features. Knowing if some features are highly correlated may suggest that feature selection methods could be effective to improve classification.

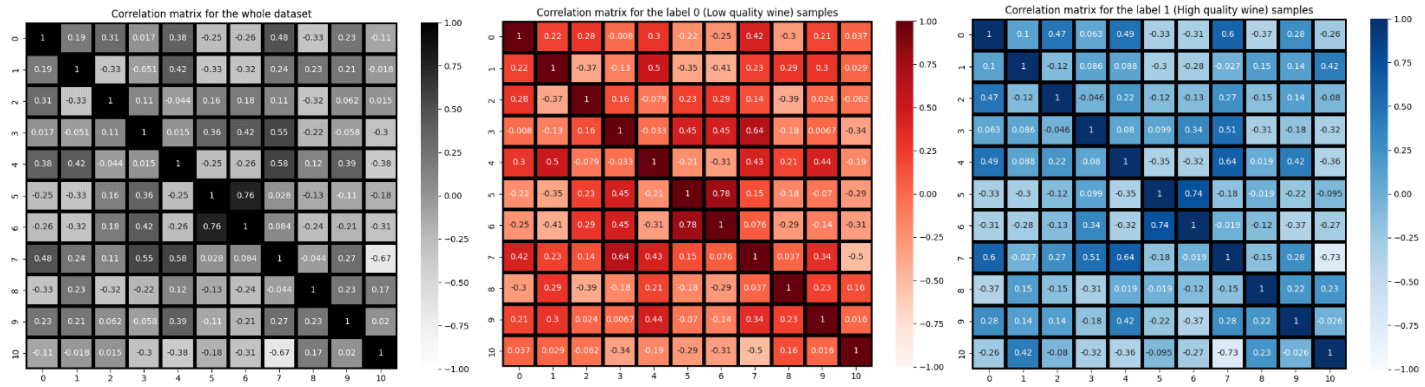


Figure 3 - Correlation matrices. Grey: all data; Red: only low-quality class data; Blue: only high-quality class data.

The correlation matrices show that the couple of features (5, 6) is highly correlated. Moreover, the couples of features (4, 7), (3, 7) are fairly correlated too. This may suggest that applying a PCA with m = 10 or m = 9 may be effective. Furthermore, in general, the correlations are high enough to state that a Naïve Bayes assumption should perform poorly.

Model selection & hyperparameters optimization

We now want to build an effective classifier for the given task; thus, we employ a model selection procedure and a hyperparameters optimization method.

Setup and process

In this stage we use only the training dataset to avoid biased results at the end. We recall that we have 1839 samples at our disposal, 1226 belonging to low quality class and 613 to the high quality one.

We proceed to employ a bunch of different ML methods; we first perform hyperparameters optimization and then we select the best model as our primary system. To this end, since we have very few samples to train our classifier on, we employ a K-Fold cross-validation approach to exploit as much samples as we can for our training. We choose K=5 for our cross-validation method, achieving a good tradeoff: we need to train only 5 classifiers to try a single configuration and we can employ 1468 training samples and 367 validation samples in each iteration. After the cross-validation step, we retrain our models on the whole training dataset to exploit as much samples as we can. We thus do not consider fixed-split training-validation sets because we would not have enough validation samples to have meaningful comparing results. It is worth noting that K=10 may be preferrable because, with K=5, we may not have enough samples to train a classifier which requires a lot of training samples; for example, if we have a simple Multivariate Gaussian Classifier and a complex kernel SVM, the validation step may lead us to choose the MVG classifier simply because the SVM would have required more samples to be effective. We keep K=5 merely to keep a fast enough validation process and to be able to try different configurations and hyperparameters. Notice that we use a frequentist approach to estimate our model parameters; in particular, we use a Maximum Likelihood estimator for probabilistic models.

Preprocessing

Along with model selection and hyperparameters optimization, we employ different preprocessing stages to improve classification performance. Each classifier may require different preprocessing stages in order to perform well for this task; therefore, we use different combinations of the different preprocessing stages depending on the type of the classifier we want to optimize our pipeline with respect to.

Our preprocessing stages are:

- Gaussianization: already described in the previous chapter
- Centering: simply centering each dimension
- Z-Normalization: making a zero-mean and unit-variance dataset - $Z = \frac{X - \mu}{\sigma}$

- L2-Normalization: simply making unit-norm vectors
- Whitening Covariance Matrix: making the covariance matrix the Identity matrix
- Whitening Within-class Covariance Matrix: making the within covariance matrix the Identity matrix
- PCA
- LDA

Performance comparison

To be able to compare different combinations of hyperparameters or preprocessing stages, together with different types of classifiers, we need to choose a performance metric and an evaluation pipeline.

As we want to build models as much independent from the target application as we can, we try to model class-conditional distributions without embedding any information about the target application whenever possible. As we will see, some discriminative non-probabilistic models do not allow to recover class-conditional log-likelihood ratios and so, they will need some additional steps to calibrate the scores.

As just introduced, we are going to use class-conditional log-likelihood ratios to perform classification. Since we need to fix a threshold to perform the actual classification, we are going to use the optimal threshold which minimizes the bayes risk. In some cases, if the scores are not well-calibrated, we will need some calibration methods to be able to recover a probabilistic interpretation.

For comparison purpose, we are only interested in choosing the best classifier or the best configuration for the selected classifier, therefore, we choose to use the minimum detection cost function as performance metric. We compare different classifiers on different target applications, so we compute the minimum detection cost function for the selected target application.

The target applications we are going to consider are three and they are described by the effective target application prior probability for the true class (High quality wine class):

- $\tilde{\pi} = 0.5$ - Which represents a balanced target application scenario
- $\tilde{\pi} = 0.1$ - Which represents a scenario where high-quality wines are more rare
- $\tilde{\pi} = 0.9$ - Which represents a scenario where high-quality wines are more common

NB: the descriptions are only an example, since these are effective target priors, they could correspond to different actual tuples (π, C_{fp}, C_{fn})*, which of course have different real interpretations.*

Thus, we conduct three different analyses on these three different target applications. In some cases, some classifiers need to embed the target application in the training process, therefore we consider these cases and we see how they perform if we embed the target application in the training process.

Multivariate Gaussian Classifier

We start by considering the family of Gaussian classifiers, in particular, we focus on approximating our class-conditional distributions with a multivariate Gaussian distribution and we finally compute the log-likelihood ratios to compute the minimum detection cost function for the different target applications. Since this is a generative probabilistic model, the whole training process is fully independent from the chosen target application.

Since this type of classifier has a very fast training process, we choose to consider the Naïve Bayes and the tied covariance variants too, even though we expect that they will perform worse than the full covariance MVG. We also consider different preprocessing strategies; we consider both the raw and the Gaussianized features along with PCA applied with a target dimensionality of 10 or 9.

In the table, we report the most meaningful results.

Model	$\tilde{\pi} = 0.5$	$\tilde{\pi} = 0.1$	$\tilde{\pi} = 0.9$
Raw Features – No PCA			
Full-Cov	0.310	0.792	0.854
Tied Full-Cov	0.334	0.829	0.738
Diag-Cov	0.420	0.852	0.932
Tied Diag-Cov	0.408	0.871	0.954
Raw Features – PCA = 10			
Full-Cov	0.326	0.803	0.856
Tied Full-Cov	0.340	0.842	0.718
Diag-Cov	0.395	0.864	0.893
Tied Diag-Cov	0.353	0.831	0.714
Raw Features – PCA = 9			
Full-Cov	0.312	0.826	0.837
Tied Full-Cov	0.340	0.834	**0.712**
Diag-Cov	0.390	0.883	0.870
Tied Diag-Cov	0.350	0.825	**0.709**
Gaussianized Features – No PCA			
Full-Cov	**0.304**	**0.783**	0.776
Tied Full-Cov	0.349	0.791	0.853
Diag-Cov	0.443	0.831	0.885
Tied Diag-Cov	0.443	0.867	0.996
Gaussianized Features – PCA = 10			
Full-Cov	**0.299**	**0.769**	0.753
Tied Full-Cov	0.352	0.790	0.800
Diag-Cov	0.401	0.828	0.900
Tied Diag-Cov	0.354	0.793	0.819
Gaussianized Features – PCA = 9			
Full-Cov	**0.304**	0.794	0.726
Tied Full-Cov	0.352	0.817	0.800
Diag-Cov	0.398	0.845	0.853
Tied Diag-Cov	0.354	0.813	0.813

Table 2 - MVG models cross-validation on the training dataset to perform hyperparameters optimization and model selection. The table reports the minDCF on the validation set (built from the K validation folds).

Comments

We first consider the main target application ($\tilde{\pi}$=0.5).

- As expected, MVG with full covariance matrices obtains the best performance since the covariance matrices are not diagonal at all and we have enough samples to estimate two covariance matrices.
- Gaussianization is also effective as expected since the raw features were, in some cases, irregularly distributed. However, the improvement is not so large as one could expect.
- PCA does not significantly improve the classification for the full covariance and the tied covariance models; however, it improves the performance of the diagonal variants. PCA with m=10 can reduce the model parameters and achieves slightly better results in the case of Gaussianized features, this may happen because reducing the model parameters allows the MVG with full covariance matrices to do a better estimation with the available samples.
- The diagonal variants perform in general worse than everyone; this is an expected behavior since the covariance matrices are not diagonal at all as we have seen in the previous chapter.
- The tied variants perform worse than the full covariance version but better than the diagonal one; this may be reasonable since we have seen that the covariance matrices of the two classes are fairly similar in some part and different in other parts.

Regarding the other two target applications, they follow a similar trend, but they provide way worse results. One exception is the case in which the tied models perform the best over the raw features projected with PCA=9 over the target application $\tilde{\pi}$=0.9. It is reasonable to think that this may be due to the fact that reducing drastically the number of parameters helps to obtain a more reliable estimate of the shared covariance matrix. As we have seen, we have fewer samples for the high quality class, plus, with a K-Fold approach we reduce of one fifth the training samples at our disposal, therefore, in a target application where we either face a lot of more samples from the high quality class or we penalize a lot the False positive errors, it is crucial to provide a good estimate for the distribution of the class Ht ($P(X \mid C = Ht)$ and thus we need either to decrease the number of parameters or to increase the number of samples.

Overall, the current best candidate is MVG with full covariance matrices applied on Gaussianized features and projected on a 10-dimensional space (discarding one dimension).

Logistic Regression

Now we want to focus our attention to discriminative models. We start considering a discriminative probabilistic approach, the logistic regression model applied on our binary classification task.

Regularized Linear Logistic Regression

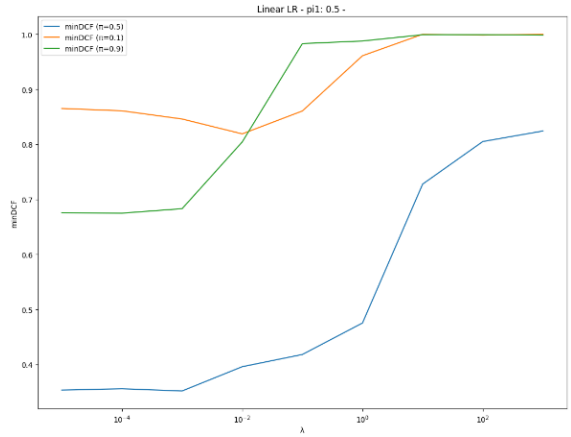
The first model we want to experiment is a regularized linear logistic regression. Since the classes are not balanced, we train our LR model modifying the objective to rebalance the cost according to our target application.

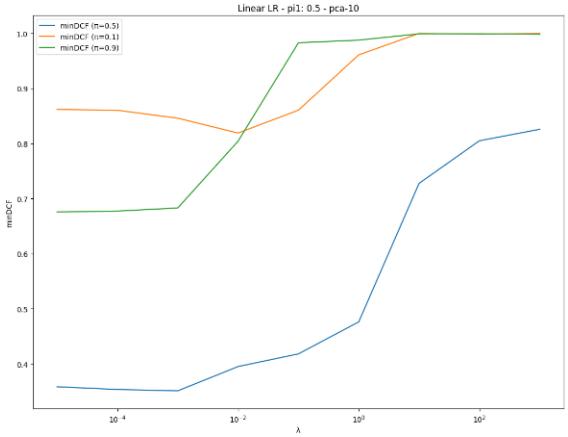
$$J(\boldsymbol{w}, b) = \frac{\lambda}{2}\|\boldsymbol{w}\|^2 + \frac{\pi_T}{n_T}\sum_{i=1|c_i=1}^{n}\log\left(1+e^{-z_i(\boldsymbol{w}^T\boldsymbol{x}_i+b)}\right) + \frac{1-\pi_T}{n_F}\sum_{i=1|c_i=0}^{n}\log\left(1+e^{-z_i(\boldsymbol{w}^T\boldsymbol{x}_i+b)}\right)$$

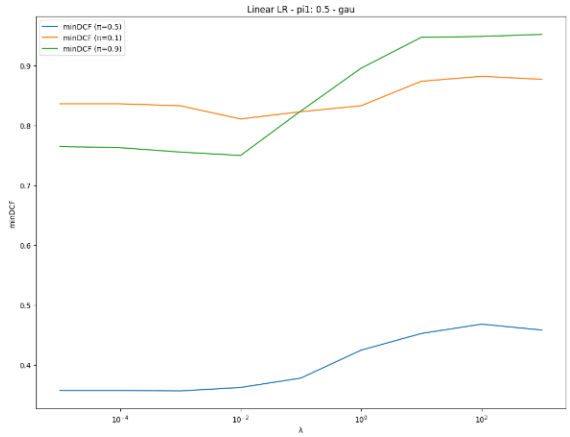
We first consider our main target application ($\tilde{\pi}$=0.5), as our regularized logistic regression model is not invariant to linear transformations, it may be worth trying different preprocessing configurations to improve classification. Therefore, we try different preprocess configurations that may be effective for our classifier and then we try, for each configuration, different values of lambda in order to optimize the hyperparameter depending on the selected preprocess configuration.

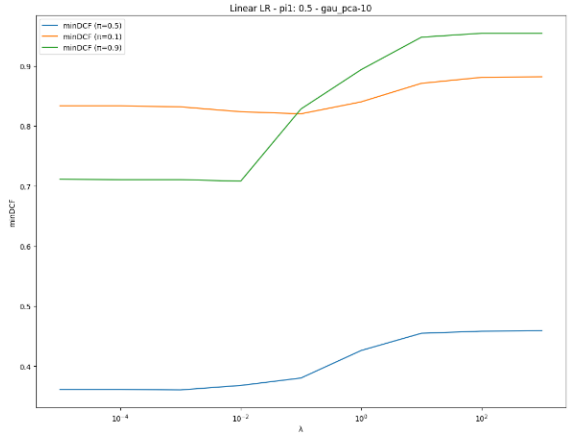
As the number of configurations / hyperparameters increases, we are no more able to perform an exhaustive grid search, so we try only a fixed number of preprocess configurations. In the following, we will try these configurations:

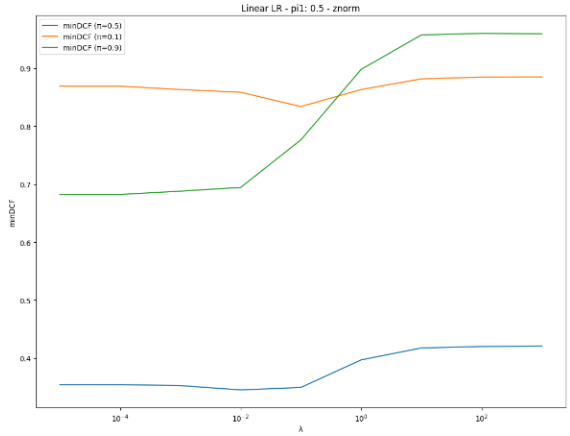
- No preprocessing
- Pca-10
- Gaussianization
- Gaussianization, PCA-10
- Z-Normalization: it may be worth for each dimension centering and making unit-variance to have equal ranges across different dimensions
- Centering, Whitening Covariance Matrix, L2 Normalization
- Centering, Whitening Within Covariance Matrix, L2 Normalization

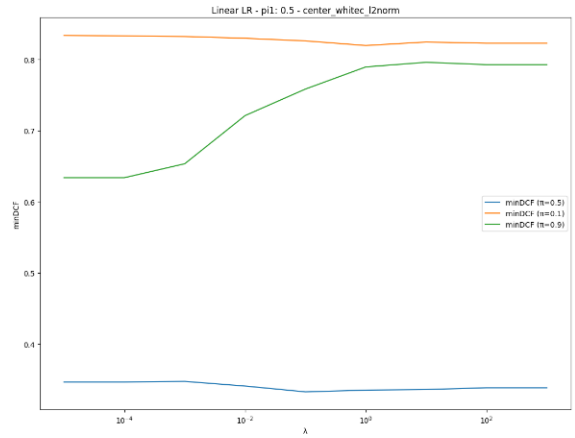












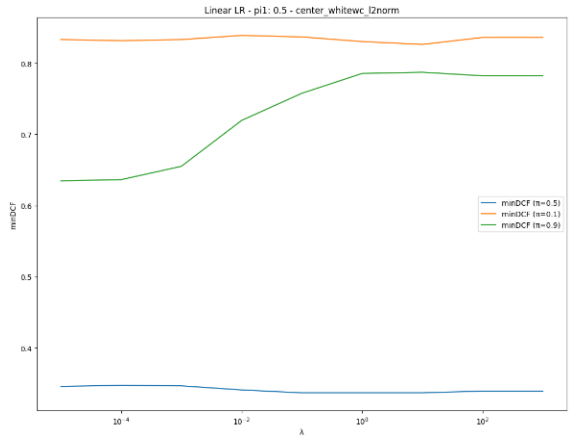


Figure 4 - Plots of performance of different Regularized Linear Logistic Regression models. Each model refers to a different preprocess configuration and shows the performance as we change the value of λ. X-axis: λ; Y-axis: minDCF on the validation set

Comments

- Overall, it is evident that Gaussianization degrades the performance obtaining the worst results.
- All the other preprocess configurations obtain similar results for small values of lambda, obtaining the best performance with the preprocessing configuration consisting of centering the dataset, whitening the covariance matrix and applying L2-normalization, together with a lambda value of 0.1, for our main target application.
- The lambda regularization does not affect a lot the performance, we observe a similar upward trend as we increase the value of lambda regardless of the chosen preprocess configuration and the target application in the most cases.
- Moreover, we observe that our model is not able to obtain good performance for the target application $\tilde{\pi}$=0.1, while it provides some improvements with respect to the full-cov MVG for the target application $\tilde{\pi}$=0.9; however, they are still poor performance.

Since the model obtains better performance for the target application $\tilde{\pi}$=0.9, we now want to try the best preprocess configuration for the specific target application embedding the prior in the training process, therefore, we are going to experiment with the following configurations:

- Target application embedded in the training process: $\tilde{\pi}$=0.1
  - Preprocess configuration: Gaussianization
  - $\lambda = 0.01$
- Target application embedded in the training process: $\tilde{\pi}$=0.9
  - Preprocess configuration: Centering, Whitening Covariance Matrix, L2-Normalization
  - $\lambda = 0.00001$

Finally, we report the best results in a table to compare with the previous full-cov MVG.

Model	$\tilde{\pi} = 0.5$	$\tilde{\pi} = 0.1$	$\tilde{\pi} = 0.9$
MVG (Full-Cov) (Quadratic) (Gaussianized features + PCA-10)	**0.299**	**0.769**	0.753
MVG (Tied Full-Cov) (Linear) (Raw features + PCA-10)	0.340	0.834	**0.712**
Linear LR ($\lambda = 0.1, \pi_T = 0.5$) (Centering, White Cov, L2-Norm)	0.333	0.826	0.758
Linear LR ($\lambda = 0.01, \pi_T = 0.5$) (Gaussianized features)	0.362	*0.811*	0.750
Linear LR ($\lambda = 0.01, \pi_T = 0.1$) (Gaussianized features)	0.346	*0.789*	0.893
Linear LR ($\lambda = 0.00001, \pi_T = 0.5$) (Centering, White Cov, L2-Norm)	0.347	0.834	*0.634*
Linear LR ($\lambda = 0.00001, \pi_T = 0.9$) (Centering, White Cov, L2-Norm)	0.369	0.863	***0.631***

Table 3 - Performance comparison of Linear Logistic Regression and MVG. The table shows the minDCF calculated on the validation set. Green: best Linear LR for our main target app. Blue: Linear LR embedding app prior in the objective and corresponding to the best configuration for the other target applications.

As the table shows, embedding the prior in the training process does not significantly improve the classification. Overall, MVG with full-cov still remains the best candidate for our main target application.

Regularized Quadratic Logistic Regression

Up to now we have employed a linear logistic regression model obtaining worse results than the MVG with the full covariance, which in contrast uses quadratic separation surfaces to perform the classification. This may suggest that linear separation surfaces may be inappropriate to separate the dataset; therefore, we proceed analyzing a quadratic logistic regression model, obtained by projecting the original features into an expanded feature space (quadratic expansion). As a result, we have a linear logistic regression model working on the expanded feature space which

translates into having quadratic separation surfaces in the original feature space. We repeat the exact same analysis for this model and we report the results in the following.

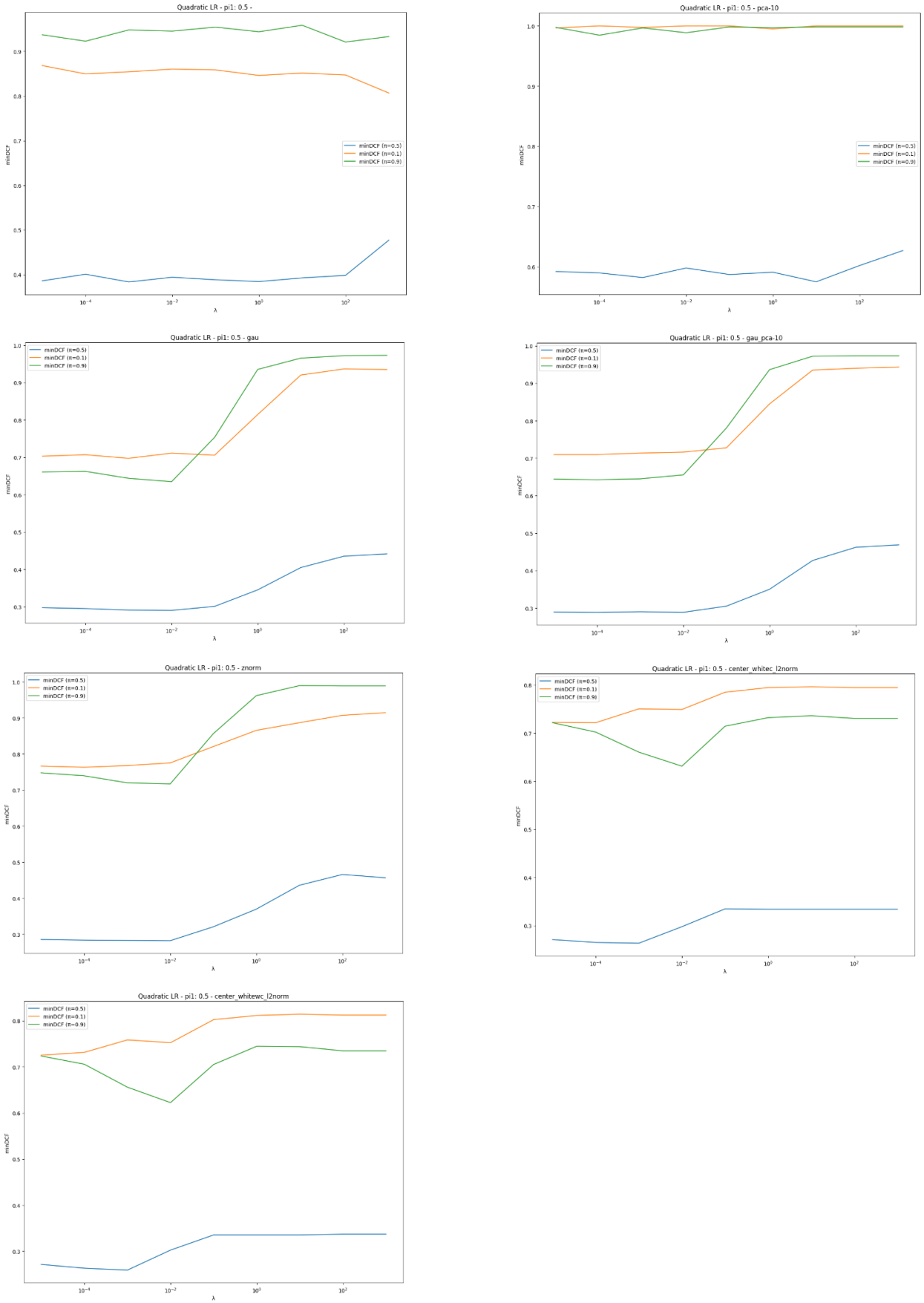


Figure 5 - Plots of performance of different Regularized Quadratic Logistic Regression models. Each model refers to a different preprocess configuration and shows the performance as we change the value of λ. X-axis: λ; Y-axis: minDCF on the validation set

Comments

- Regardless of the preprocess configuration or the target application, similarly to the linear case the parameter lambda does not significantly affect the performance, in general, greater values of lambda degrade the performance. In some cases, we observe a slight improvement for small lambdas up to 10^{-1}.
- We can observe that Gaussianization provides a huge performance boost compared to the model trained on the raw features; however, the best results are achieved again with the combo Centering, Whitening Covariance/Within Covariance matrix and a lambda value of 0.001.

We observe some slight improvement in the other target applications too; therefore, we try to embed the specific prior in the training process and we compare the overall results.

- Target application embedded in the training process: $\tilde{\pi}$=0.1
  - Preprocess configuration: Gaussianization
  - $\lambda = 0.001$
- Target application embedded in the training process: $\tilde{\pi}$=0.9
  - Preprocess configuration: Centering, Whitening Within Covariance Matrix, L2-Normalization
  - $\lambda = 0.01$

Model	$\tilde{\pi} = 0.5$	$\tilde{\pi} = 0.1$	$\tilde{\pi} = 0.9$
MVG (Full-Cov) (Gaussianized features + PCA-10)	**0.299**	**0.769**	0.753
Linear LR ($\lambda = 0.00001, \pi_T = 0.9$) (Centering, White Cov, L2-Norm)	0.369	0.863	**0.631**
Quadratic LR ($\lambda = 0.001, \pi_T = 0.5$) (Centering, White Within Cov, L2-Norm)	**0.259**	0.759	0.656
Quadratic LR ($\lambda = 0.001, \pi_T = 0.5$) (Gaussianization)	0.291	*0.698*	0.644
Quadratic LR ($\lambda = 0.001, \pi_T = 0.1$) (Gaussianization)	0.299	***0.680***	0.665
Quadratic LR ($\lambda = 0.01, \pi_T = 0.5$) (Centering, White Within Cov, L2-Norm)	0.303	0.752	***0.622***
Quadratic LR ($\lambda = 0.01, \pi_T = 0.9$) (Centering, White Within Cov, L2-Norm)	0.326	0.788	*0.643*

Table 4 - Performance comparison of the best models up to now. There are all the best models for every target application. We also compare with the last four rows the models trained embedding the prior in the objective with the same models without embedding the prior.

The table shows that the Quadratic Regularized Logistic Regression model outperforms the full-cov MVG one. Also, embedding the prior in the training process provides an improvement in the target application $\tilde{\pi}$=0.1.

Support Vector Machine

Up to now we have considered generative probabilistic models (MVG) and discriminative probabilistic models (Logistic Regression), we now turn our attention to the analysis of a discriminative non-probabilistic model method, the support vector machine. As we have seen, linear models have demonstrated to be ineffective for the separation of this dataset, for completeness we first report the performance of a linear support vector machine, although we expect poor results. Next, we analyze non-linear separation surfaces leveraging different kernels, from polynomial to RBF kernels.

To keep the implementation simple, we expand the input feature vector and the vector w incorporating the bias term:

$$\hat{x}_i = \begin{bmatrix} x_i \\ K \end{bmatrix}, \quad \hat{w} = \begin{bmatrix} w \\ b \end{bmatrix}$$

Since with this method we are regularizing the bias term too, we can obtain sub-optimal results. To mitigate this problem, we can increase the value of K, keeping in mind that increasing the value of K slows down the objective

optimization algorithm calculation because it requires more iterations to converge. Therefore, in the following, sometimes we try K=10 to see if this affects the performance.

Linear SVM

We briefly report the most significant results with the most promising preprocess configuration. Note that we have experimented more preprocess configurations, but they are not reported here for conciseness. We need to tune the hyperparameter C, which weighs the error term (hinge loss), so we cross-validate for different values of C and we show some plots.

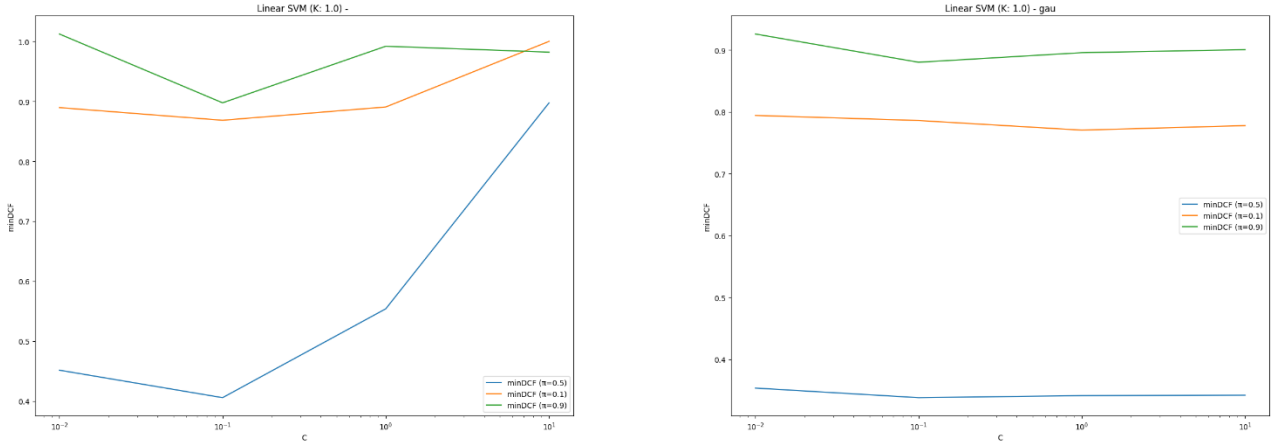


Figure 6 - Plots of Linear SVM representing the minDCF calculated for the target applications over the validation set as we change the value of C. Left: raw features. Right: Gaussianized features.

We can also rebalance the classes in the objective formulation to account for unbalanced classes. We therefore do it only for the Gaussianized features.

Model	$\tilde{\pi} = 0.5$	$\tilde{\pi} = 0.1$	$\tilde{\pi} = 0.9$
Tied Full-Cov MVG (raw features)	0.334	0.829	0.738
Linear LR ($\lambda = 0.1, \pi_T = 0.5$) (Centering, White Cov, L2-Norm)	0.333	0.826	0.758
Linear SVM (C=0.1, K = 1) Without class rebalancing (Gaussianization)	0.338	0.786	0.880
Linear SVM (C=0.1, K = 10) Without class rebalancing (Raw features)	0.371	0.840	0.824
Linear SVM (C=0.1, K = 10) Class rebalancing $\pi_T = 0.5$ (Gaussianization)	*0.353*	0.821	0.812
Linear SVM (C=100, K = 10) Class rebalancing $\pi_T = 0.1$ (Gaussianization)	0.613	*0.911*	0.986
Linear SVM (C=1, K = 10) Class rebalancing $\pi_T = 0.9$ (Gaussianization)	0.391	0.943	**0.645**

Table 5 - Performance comparison of all the linear models considered up to now. In blue we also consider a class-rebalanced version of the best Linear SVM for the different target applications. The table shows minDCF values calculated on the validation set.

Comments

- The linear svm performs similarly and slightly worse than the other linear models as expected.

- Class rebalancing degrades the performance except for the target app π=0.9.

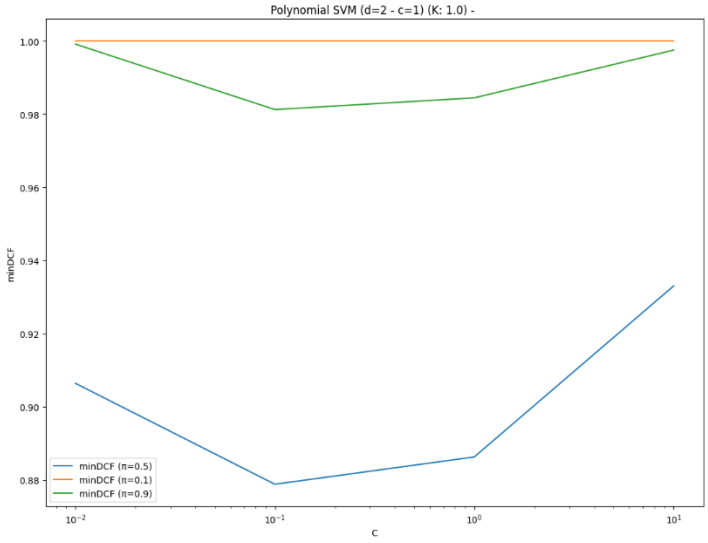
Polynomial SVM

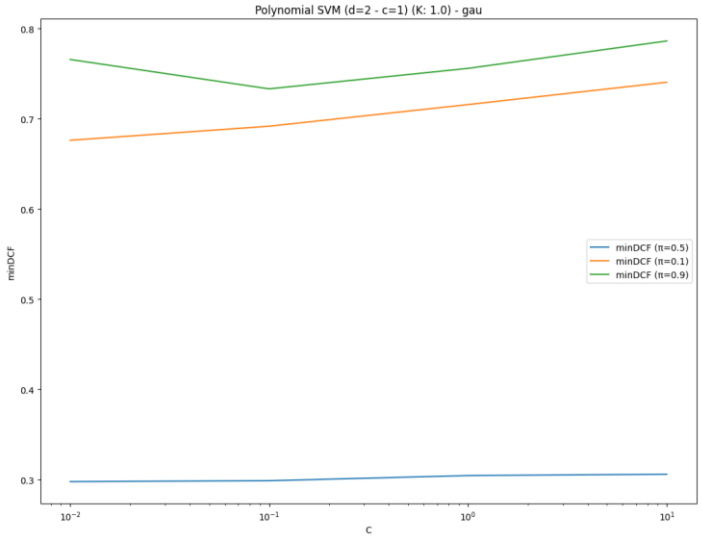
Since linear models have proven to be ineffective, we now experiment a polynomial svm, in particular, we see how quadratic and cubic polynomial kernels perform.

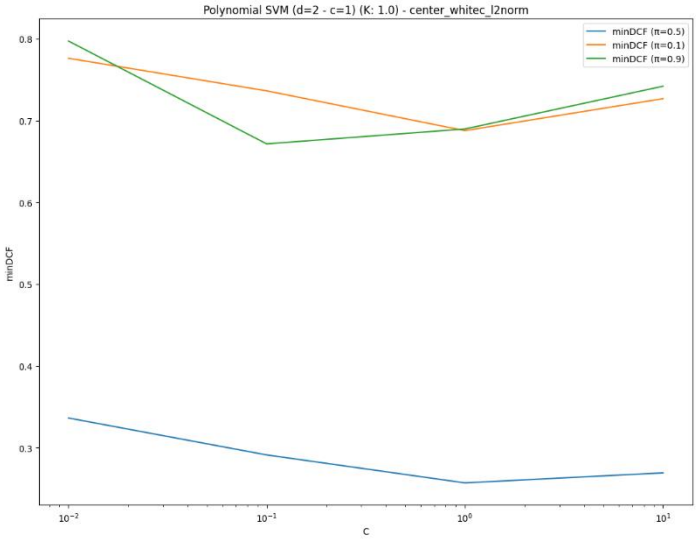
In the following, we will experiment with these preprocess configurations:

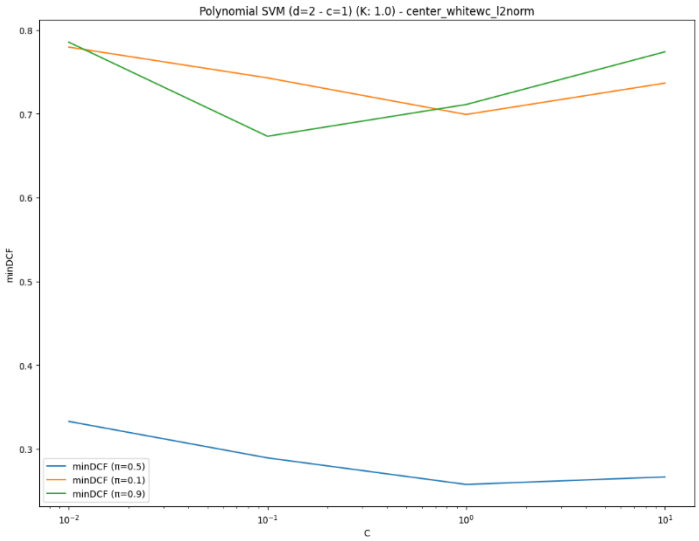
- No preprocessing
- Gaussianization
- Centering, Whiten Covariance Matrix, L2 Normalization
- Centering, Whiten Within Covariance Matrix, L2 Normalization

We need again to tune the hyperparameter C for the different configurations and classifier types.









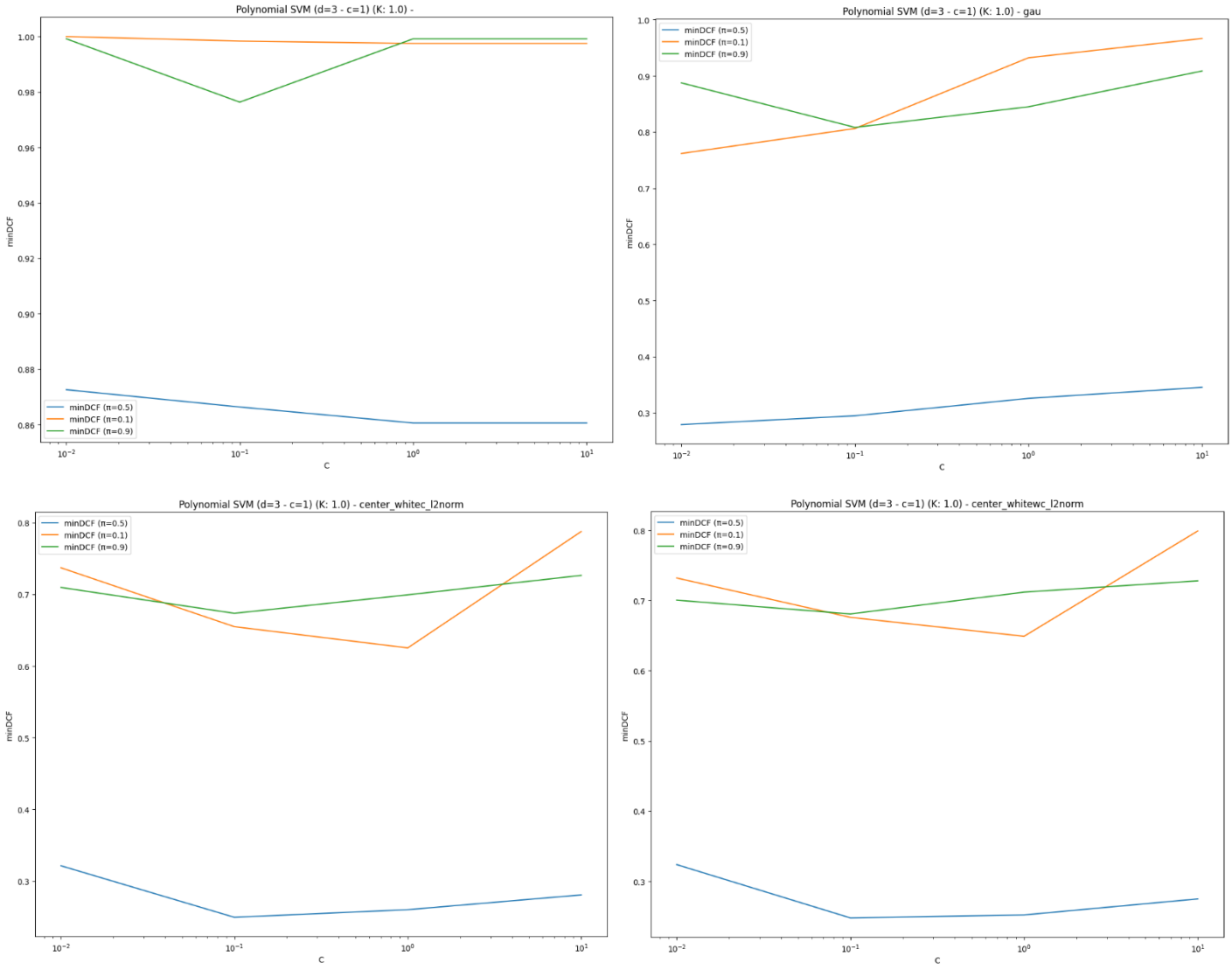


Figure 7 - Plots of the performance of different polynomial SVM models. Each plot shows a different preprocess configuration and represents the minDCF calculated on the validation set as we change the value of C. The first 4 plots are referring to a quadratic polynomial kernel while the last 4 plots are referring to a cubic polynomial kernel.

Comments

- Polynomial kernels perform very bad on raw features; this may happen because the ranges of the different features are very different.
- All the preprocess configurations perform well on average, achieving the top performance on the cubic kernel with a preprocess configuration composed of Centering, Whitening Within Class Covariance Matrix, L2-Normalization.
- Cubic kernel SVM outperforms Quadratic LR and MVG with full-cov.
- Class-rebalancing the best configuration does not improve the classification.

Overall, the current best non-linear performances for our main target application are summarized in the following table.

Model	$\tilde{\pi} = 0.5$	$\tilde{\pi} = 0.1$	$\tilde{\pi} = 0.9$
MVG (Full-Cov) **(Gaussianized features + PCA-10)**	0.299	0.769	0.753
Quadratic LR ($\lambda = 0.001, \pi_T = 0.5$) **(Centering, White Within Cov, L2-Norm)**	0.259	0.759	**0.656**
Polynomial SVM (d = 3, C=0.1) **(Centering, White Within Cov, L2-Norm)**	**0.248**	**0.676**	0.681

Table 6 - Performance comparison of the best non-linear models considered up to now for our main target application. The table shows the values of minDCF calculated on the validation set.

Radial Basis Function SVM

To finish our validation of SVM approaches, we lastly experiment with the Gaussian Radial Basis Function kernel, which projects the features in an infinite Hilbert space, through the kernel function: $\boldsymbol{k(x_1, x_2) = e^{-\gamma ||x_1 - x_2||^2}}$

The kernel depends on the distance of the points, the closer the points, the higher the kernel value will be. Therefore, we need to tune the hyperparameter γ, which represents the width of the kernel; a small value means a wide kernel, so support vectors influence a lot of other points, and a great value means vice versa. Thus, we need to tune jointly C and γ; we restrict our analysis on a large grid search with a coarse level of detail to have a first approximation, next, for the best configuration, we repeat a fine-grained search to reach a closer to local optimum result. It is worth noting that a random grid search could be more effective in this step.

Again, we do the first coarse level search on different preprocess configurations:

- No preprocessing
- Gaussianization
- Centering, Whiten Covariance Matrix, L2 Normalization
- Centering, Whiten Within Covariance Matrix, L2 Normalization

Note that we have experimented more values of gamma, but we report the top-three gamma values to have a clearer plot. We report the full plots only for our main target application ($\tilde{\pi}$=0.5), while for the others, we report only the best configuration plot.

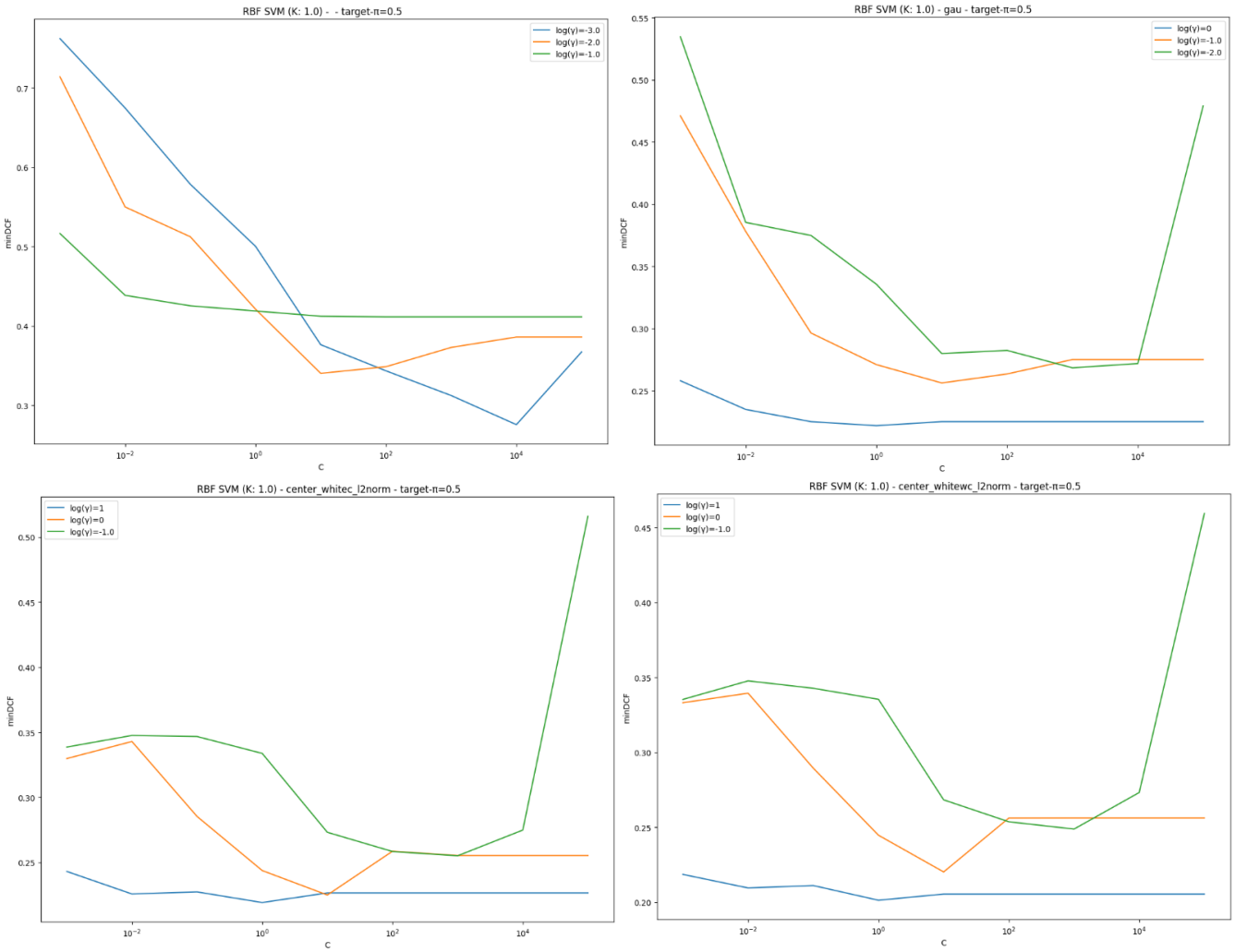


Figure 8 - Plots of the performance of RBF SVM models. The plots represent the values of minDCF calculated over the validation set for the main target application (π=0.5) over different values of C (coarse-level grid search). Each figure refers to a different preprocess configuration. Different lines refer to different values of γ (top-three gamma values).

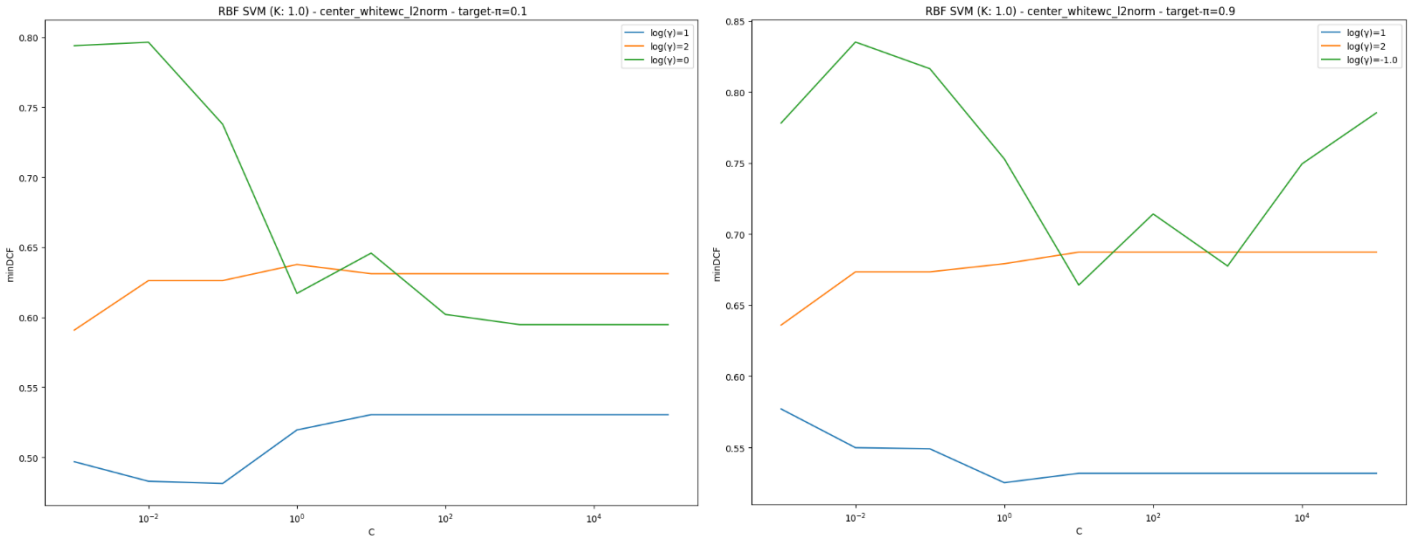


Figure 9 - Plots of the performance of RBF SVM models for the other two target applications. Left: π=0.1. Right: π=0.9. Both are RBF SVM models trained on preprocessed feature (Centering, Whitening Within Covariance Matrix, L2-Normalization).

Since the best results are obtained with the preprocessing configuration consisting of Centering, Whitening Within Covariance Matrix, L2-Normalization, we now repeat a fine-grained grid search only using that configuration and focusing on small ranges of C and gamma around the already found minimum. We try also to do class-rebalancing on the best configuration with the best hyperparameters.

We report the results in a table comparing the best models so far.

Model	$\widetilde{\boldsymbol{\pi}} = \mathbf{0.5}$	$\widetilde{\boldsymbol{\pi}} = \mathbf{0.1}$	$\widetilde{\boldsymbol{\pi}} = \mathbf{0.9}$
MVG (Full-Cov) **(Gaussianized features + PCA-10)**	0.299	0.769	0.753
Quadratic LR ($\boldsymbol{\lambda = 0.001, \pi_T = 0.5}$) **(Centering, White Within Cov, L2-Norm)**	0.259	0.759	0.656
Quadratic LR ($\boldsymbol{\lambda = 0.01, \pi_T = 0.5}$) **(Centering, White Within Cov, L2-Norm)**	0.303	0.752	**0.622**
Polynomial SVM (d = 3, C=0.1) **(Centering, White Within Cov, L2-Norm)**	**0.248**	**0.676**	0.681
RBF SVM (γ = 8, C=0.5) **Without class-rebalancing** **(Centering, White Within Cov, L2-Norm)**	**0.195**	0.499	0.544
RBF SVM (γ = 10, C=0.1) **With class-rebalancing to $\boldsymbol{\pi_T = 0.1}$** **(Centering, White Within Cov, L2-Norm)**	0.211	**0.472**	0.549
RBF SVM (γ = 10, C=1) **With class-rebalancing to $\boldsymbol{\pi_T = 0.9}$** **(Centering, White Within Cov, L2-Norm)**	0.201	0.520	**0.525**

Table 7 - Performances of the current best models for the different target applications. Blue: previous best model scores. Red: best new model scores. The table contains the minDCF calculated on the validation set.

Comments

- The RBF SVM with proper configuration and hyperparameters outperforms with a large margin all the previous best models on all the target applications.
- Again, the combination of centering, whitening the within class covariance, l2-normalization, is the most effective preprocessing strategy.
- RBF SVM on raw features provides the worst results, even worse than models trained on Gaussianized features. This may be due to the intrinsic distribution of this specific dataset since RBF SVM takes into account distances.

- We observe in all the plots a similar trend, a decrease of minDCF as we increase the value of C up to a certain point, notice that the behavior depends also on the chosen gamma. Moreover, the best value of gamma depends on the chosen preprocess configuration.
- Class-rebalancing does not improve classification except for the target application π_T=0.1.
- NB: RBF SVM models depend on the number of available samples, therefore we can expect a performance improvement as we re-train the model on the whole training dataset.
- Up to now, our best model for our main target application is the RBF SVM model without class-balancing and trained on preprocessed feature (centering, whitening within class covariance matrix, l2-norm).

Gaussian Mixture Model

To conclude, the last model we are going to experiment with is again a generative probabilistic model, the Gaussian Mixture Model. Starting with the idea that the data of each class can be partitioned in N subclasses, we are going to model the data of each class with a GMM, where each component will represent each sub-class. As the MVG model has performed not so greatly, we can safely assume that class-conditional data are not well Gaussian distributed, even after the Gaussianization process (remember that the Gaussianization processes the data in a class-independent way). Therefore, since GMM can model theoretically any distribution, we can expect that it will outperform the MVG one.

In the following analysis we are going to try different preprocess configurations, along with different types of GMM models, including GMM with tied covariance matrices or GMM with diagonal covariance matrices, or even both. Leveraging the LBG algorithm, we are going to experiment different numbers of components, from 1 component (MVG) to 256 components. To provide an explanation on these numbers, we recall that, with 5-folds cross-validation, we have only 1468 training samples on each iteration. Since each component requires estimating in the worst case a covariance matrix 11x11, a mean vector (11) and a weight, and we must do it for both classes, we have $\left(\frac{11x(11+1)}{2} + 11 + 1\right) \; x \; components \; x \; 2$ parameters to estimate; therefore, since we have only 490 samples for the true class, increasing too much the number of components will not make any sense.

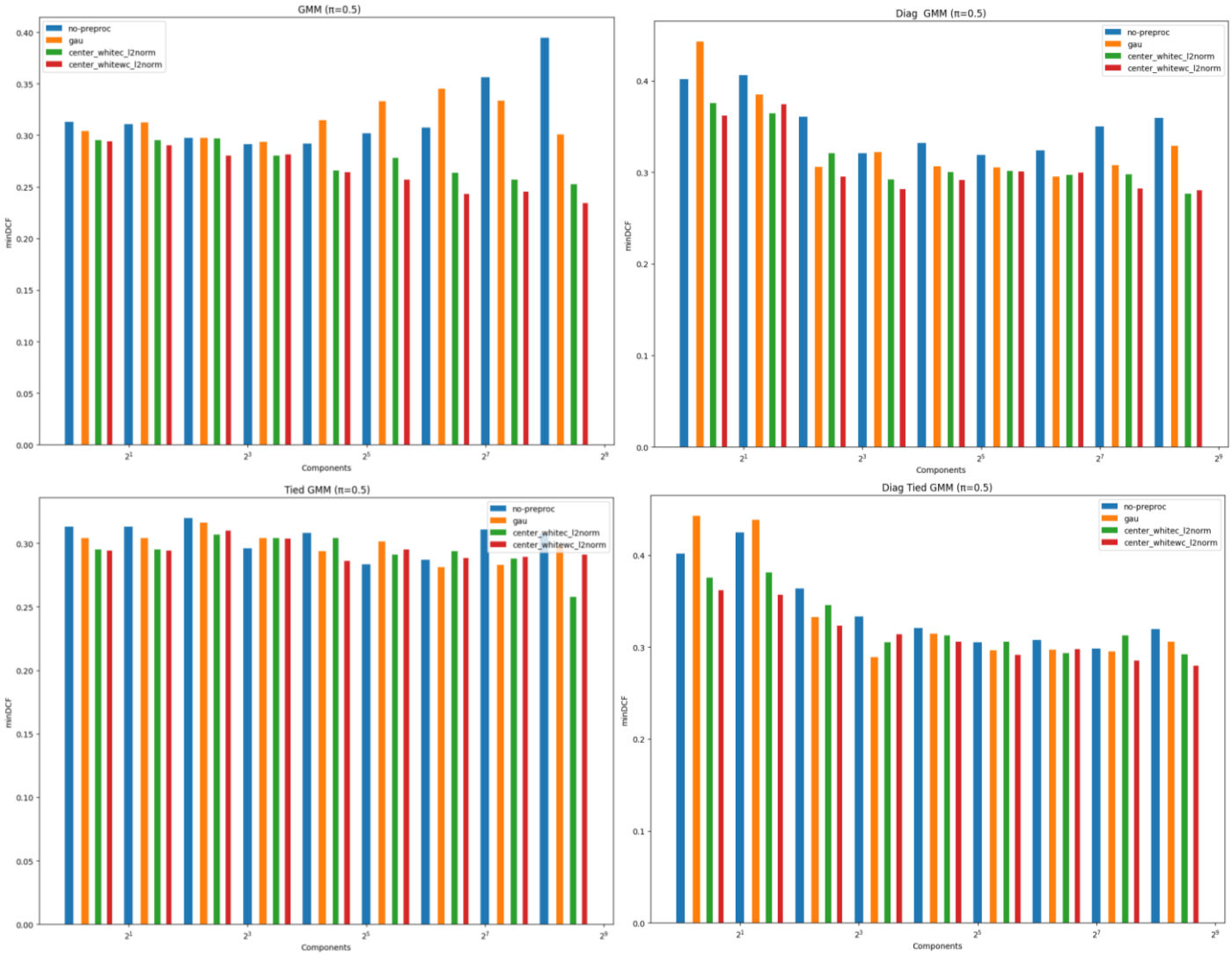


Figure 10 - Plots of performances of different GMM models as we increase the number of components, only for the main target application. The height indicates the minDCF value calculated on the validation set. Top-left: full-cov; Top-right: diag-cov; Bottom-left: tied-cov; Bottom-right: diag&tied-cov.

Model	$\tilde{\pi} = 0.5$	$\tilde{\pi} = 0.1$	$\tilde{\pi} = 0.9$
MVG (Full-Cov) **(Gaussianized features + PCA-10)**	0.299	0.769	0.753
Quadratic LR ($\lambda = 0.001, \pi_T = 0.5$) **(Centering, White Within Cov, L2-Norm)**	0.259	0.759	0.656
Polynomial SVM (d = 3, C=0.1) **(Centering, White Within Cov, L2-Norm)**	0.248	0.676	0.681
RBF SVM (γ = 8, C=0.5) **Without class-rebalancing** **(Centering, White Within Cov, L2-Norm)**	**0.195**	0.499	0.544
RBF SVM (γ = 10, C=0.1) **With class-rebalancing to $\pi_T = 0.1$** **(Centering, White Within Cov, L2-Norm)**	0.211	**0.472**	0.549
RBF SVM (γ = 10, C=1) **Without class-rebalancing** **(Centering, White Within Cov, L2-Norm)**	0.201	0.520	**0.525**
GMM (Full-Cov, 256 comps) **(Centering, White Within Cov, L2-Norm)**	**0.234**	**0.616**	**0.610**

Table 8 - Performances of the current best models for the different target applications. Blue: second best model. Red: best model. The table contains the minDCF calculated on the validation set.

Comments

- Regardless of the type of model, we can see that centering, whitening (within) cov matrix, l2-norm preprocess configuration outperforms all the other preprocess configurations with a large margin.
- Moreover, we can observe a general downward trend as we increase the number of components.
- The diagonal-covariance models are in general the worst. This is reasonable since we have seen that the covariance matrices are not close-to-diagonal.
- The tied-covariance models perform well on average, still worse than the full-covariance, we have tried components up to 1024 for tied model, but they are still worse. This could mean that the distribution is easier approximated by a full-covariance model or that increasing too much the number of components starts to overfit our data.
- Gaussianizing the features performs in the major of the cases worse than the raw features, this may due to the fact that reducing the dynamic ranges of the samples makes more complex to separate some clusters.
- As expected, the GMM classifier outperforms the MVG one.

GMM with full-covariance matrix performs better than every other model except the RBF SVM one. We thus select the RBF SVM with the best configuration as our primary system and the GMM with full-covariance as our secondary system.

- Primary system: **RBF SVM (γ = 8, C=0.5), Without class-rebalancing, (Centering, White Within Cov, L2-Norm)**
- Secondary system: **GMM (Full-Cov, 256 components), (Centering, White Within Cov, L2-Norm)**

From now on, the primary system will be referred only as RBF SVM and the secondary system as GMM.

Scores Calibration

Up to now, we have assessed the goodness of the different models, configurations and hyperparameters based on the minimum DCF metric. However, that metric allows assessing the capability of the model to separate the classes given a set of scores and a target application, which in the binary-class case corresponds to picking the best threshold on the specific validation set to perform optimal decision and achieving the minimum cost.

Therefore, the actual cost we would pay in a real case scenario depends on the goodness of the decision we make using those scores, that in the binary-class case corresponds to picking a good threshold value. If the scores have a probabilistic interpretation, in order to optimize the Bayes risk, we pick the theorical optimal threshold: $t = -\log\frac{\tilde{\pi}}{1-\tilde{\pi}}$

Thus, we are interested in understanding if the scores generated by our models are well-calibrated. So, we compute the actual DCF using the theoretical optimal threshold for our candidate models.

Model	$\tilde{\pi} = 0.5$		$\tilde{\pi} = 0.1$		$\tilde{\pi} = 0.9$	
	minDCF	*actDCF*	*minDCF*	*actDCF*	*minDCF*	*actDCF*
RBF SVM	0.195	0.560	0.499	1.000	0.544	1.000
GMM	0.234	0.248	0.616	0.774	0.610	1.175

The results show that the SVM model is providing totally uncalibrated scores as we could have expected since our SVM approach lacks of a probabilistic interpretation. The GMM provides scores almost calibrated for our main target application, with a relative loss of 5%. However, things are different for the other two target applications where the scores become completely uncalibrated. We recall that even if GMM is a generative probabilistic model and thus it should provide calibrated scores, things are actually more complex. In fact, wrong model assumptions or differences between train and test distributions may cause uncalibrated scores.

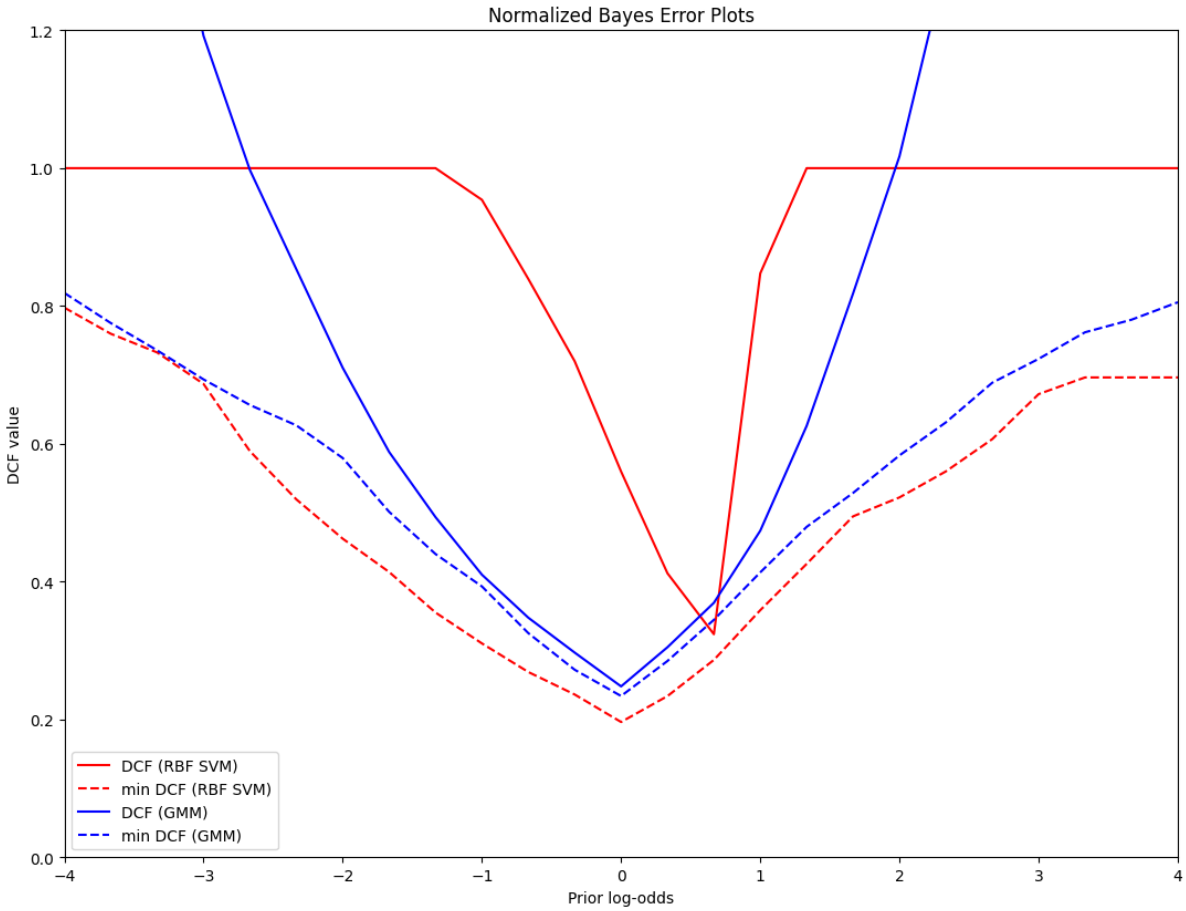


Figure 11 - Normalized Bayes Error Plot of the best svm model and the best gmm model

The statement is confirmed by a Bayes Error Plot which shows that the GMM model is almost calibrated around our main target application ($\tilde{\pi} = 0.5$) and becomes very uncalibrated as we move to more unbalanced target applications; furthermore, the SVM model provides totally uncalibrated scores.

As it is evident that we need a way to choose a good threshold, we can resort to two strategies:

- Picking the threshold which gives the minimum DCF over the validation set
- Re-calibrate the scores over the validation scores set so that the theoretical optimal threshold provides close to optimal results over a wide range of effective priors.

We will go for the second strategy. In particular, we split our validation scores in a training part (calibration set) and a validation part and then we use the training part to train a linear logistic regression model at score level.

f(s) can be interpreted as class-cond. log-likelihood ratio for the two class hypotheses: $f(s) = \log\frac{f_{S|C}(s \mid H_T)}{f_{S|C}(s \mid H_F)} = \alpha\, s + \beta$

The class posterior log-likelihood ratio, which is used to perform the classification, corresponds to:

$$\log\frac{P(C = H_T \mid s)}{P(C = H_F \mid s)} = \alpha\, s + \beta + \log\frac{\tilde{\pi}}{1 - \tilde{\pi}} = \alpha\, s + \beta'$$

Therefore, we only need to estimate α and β′ and then we can subtract the prior log-odds from the score to recover class-conditional log-likehood ratios. Note that we are effectively embedding a target-application in our re-calibration method; however, it should perform well also on other target applications.

RBF SVM	***minDCF ($\tilde{\pi} = 0.5$)***	***minDCF ($\tilde{\pi} = 0.1$)***	***minDCF ($\tilde{\pi} = 0.9$)***
	0.193	0.477	0.504
	actDCF ($\tilde{\pi} = 0.5$)	***actDCF ($\tilde{\pi} = 0.1$)***	***actDCF ($\tilde{\pi} = 0.9$)***
Uncalibrated	0.557	1.000	1.000
Log-Reg $\tilde{\pi}$=0.5	**0.199**	**0.554**	**0.624**
Log-Reg $\tilde{\pi}$=0.1	0.570	0.622	1.000
Log-Reg $\tilde{\pi}$=0.9	0.438	0.711	1.365

Table 9 - Performance comparison of RBF SVM after Linear Logistic Regression scores recalibration on a fixed-split calibration set. The results are minDCF calculated on the validation set obtained from the calibration-validation split of the original validation set.

As we can see, LR-recalibration provides good results for all the target applications. We notice a performance decreasing while recovering calibration scores embedding the prior $\tilde{\pi}$=0.9 for the target application $\tilde{\pi}$=0.9. However, since the original scores were completely uncalibrated, we notice a good performance even with the only training prior $\tilde{\pi}$=0.5.

GMM	***minDCF ($\tilde{\pi} = 0.5$)***	***minDCF ($\tilde{\pi} = 0.1$)***	***minDCF ($\tilde{\pi} = 0.9$)***
	0.239	0.568	0.567
	actDCF ($\tilde{\pi} = 0.5$)	***actDCF ($\tilde{\pi} = 0.1$)***	***actDCF ($\tilde{\pi} = 0.9$)***
Uncalibrated	0.247	0.718	1.124
Log-Reg $\tilde{\pi}$=0.5	**0.248**	**0.594**	**0.588**
Log-Reg $\tilde{\pi}$=0.1	0.479	1.340	0.754
Log-Reg $\tilde{\pi}$=0.9	0.479	0.895	1.143

Table 10 - Performance comparison of GMM after Linear Logistic Regression scores recalibration on a fixed-split calibration set. The results are minDCF calculated on the validation set obtained from the calibration-validation split of the original validation set.

Using a training prior of $\tilde{\pi}$=0.5 achieves very good performance and we can safely state that the achieved scores are well-calibrated.

We now show the Bayes Error Plot for our calibrated scores using the effective prior $\tilde{\pi}$=0.5 to see how it performs on different operating points.

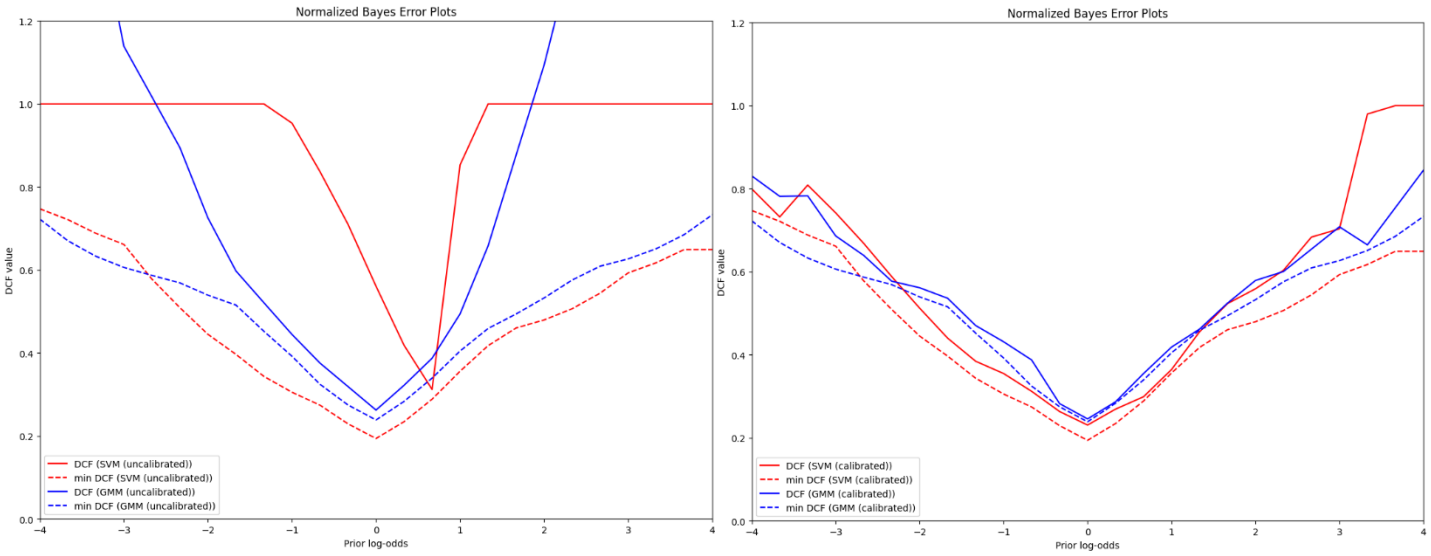


Figure 12 - Bayes Error Plots before and after LR scores re-calibration with training prior $\tilde{\pi}$=0.5. The results are obtained both on the validation set of the calibration-validation split. Left: without calibration. Right: with calibration.

The plot shows a huge improvement after the calibration for a wider range of operating points. There is however still a little gap between the minimum DCF and the actual DCF, especially for the SVM model.

Model fusion

This method allows easily to combine different classifiers in order to improve the final performance. An effective fusion approach consists of score-level fusion rather than decision-level fusion. We only need to train again a linear LR on the scores provided by the two classifiers and we have a fused score with the good property of being automatically re-calibrated.

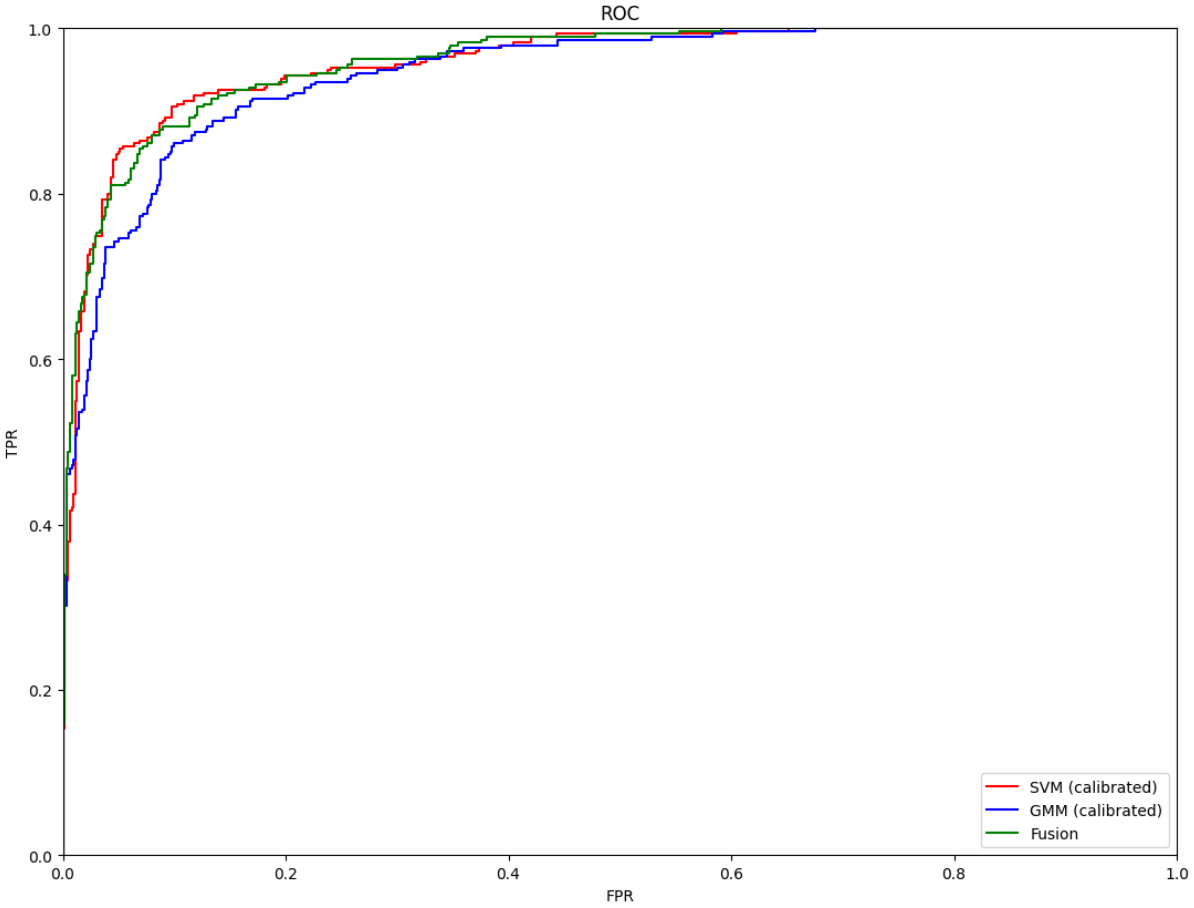


Figure 13 - ROC curve for performance comparison. The scores are calculated on the validation set obtained from the calibration-validation split.

As the ROC curve shows, the fusion outperforms the SVM on certain regions while being worse in another region.

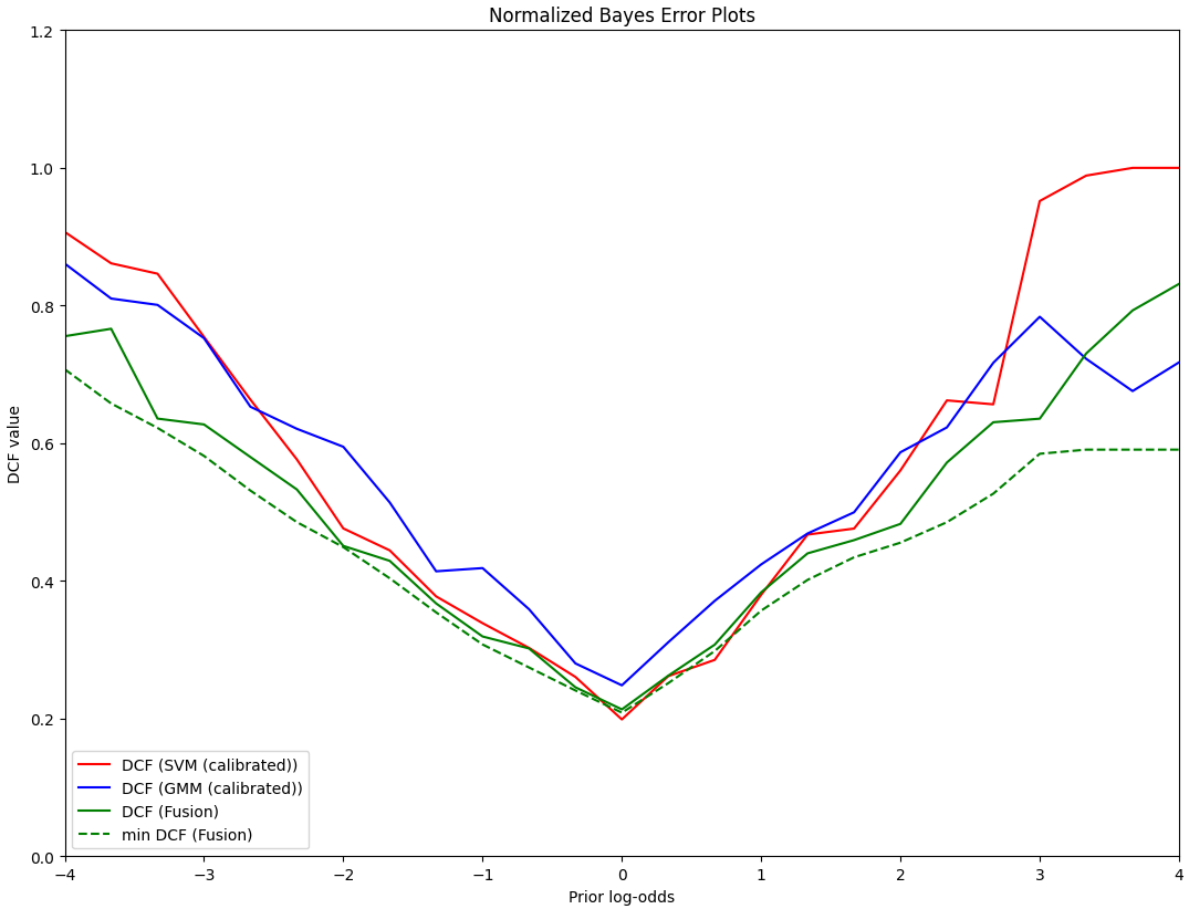


Figure 14 - Bayes Error Plots after LR scores re-calibration and fusion with training prior $\tilde{\pi}$=0.5. The results are both obtained on the validation set of the calibration-validation split.

Model	$\tilde{\pi} = 0.5$		$\tilde{\pi} = 0.1$		$\tilde{\pi} = 0.9$	
	minDCF	*actDCF*	*minDCF*	*actDCF*	*minDCF*	*actDCF*
RBF SVM (calibrated)	**0.193**	**0.199**	0.477	0.554	0.504	0.624
GMM (calibrated)	0.239	0.248	0.568	0.594	0.567	0.588
Fusion	0.209	0.213	**0.471**	**0.494**	**0.472**	**0.541**

Table 11 - minDCF and actual DCF of the calibrated models and the fused one. The values are calculated on the validation set obtained from the calibration-validation split.

As confirmed by a Normalized Bayes Error Plot, the Fused model achieves better performance on a wide range of operating points and still remains more calibrated than the SVM one.

With that in mind, we choose as our candidate model the Fusion, we will assess the goodness of our decision in the next chapter. We report for completeness all the details of our candidate model.

1. Train **RBF SVM (γ = 8, C=0.5), Without class-rebalancing, (Centering, White Within Cov, L2-Norm)** on 4/5 of the training dataset;
2. Train **GMM (Full-Cov, 256 components), (Centering, White Within Cov, L2-Norm)** on 4/5 of the training dataset;
3. Train a **Regularized Linear Logistic Regression ($\lambda=10^{-3}$) with a prior-weight objective (π_T=0.5)** on the scores generated by the two models obtained from the remaining 1/5 of the training dataset.

Experiments and results

After having performed model selection and hyperparameters optimization using the described protocol on the training dataset, we are now interested to contrast and assess the made choices and performances over held-out data. To this end, we have left out the evaluation dataset, composed of 1822 samples, 1158 of low quality class and 664 of high quality class.

As we have already seen in the previous chapter, we briefly introduce the setup with the goal of contrasting our decisions this time. As in our 5-folds cross-validation we have used 4/5 of the training dataset to train our models, we proceed to assess the quality of the classifiers on the evaluation set using both 4/5 and the full training dataset to see if our made choices were good for the evaluation data too and if adding more samples to the training process would change our performances drastically. As already described, we first use minDCF metric to perform classifier comparison.

Multivariate Gaussian Classifier

We start with the first analyzed classifier, the family of the Gaussian classifiers.

Model	***4/5 Train data***			***All train data***		
	$\tilde{\pi} = 0.5$	$\tilde{\pi} = 0.1$	$\tilde{\pi} = 0.9$	$\tilde{\pi} = 0.5$	$\tilde{\pi} = 0.1$	$\tilde{\pi} = 0.9$
Raw Features – No PCA						
Full-Cov	0.329	**0.645**	0.716	0.337	**0.656**	0.698
Tied Full-Cov	**0.319**	0.671	0.706	**0.315**	0.681	0.700
Diag-Cov	0.376	0.740	0.874	0.367	0.731	0.869
Tied Diag-Cov	0.372	0.744	0.907	0.369	0.738	0.930
Raw Features – PCA = 10						
Full-Cov	0.342	0.675	0.714	0.343	0.692	0.707
Tied Full-Cov	0.332	0.713	0.725	0.328	0.711	0.723
Diag-Cov	0.342	0.738	0.811	0.357	0.730	0.804
Tied Diag-Cov	0.330	0.727	0.708	0.331	0.726	0.716
Raw Features – PCA = 9						
Full-Cov	0.336	0.687	0.733	0.349	0.699	0.703
Tied Full-Cov	0.333	0.707	0.719	0.332	0.712	0.714
Diag-Cov	0.355	0.749	0.827	0.358	0.745	0.811
Tied Diag-Cov	0.331	0.723	0.705	0.330	0.723	0.719
Gaussianized Features – No PCA						
Full-Cov	0.333	0.691	0.758	0.334	0.675	0.724
Tied Full-Cov	0.326	0.699	0.789	0.319	0.698	0.800
Diag-Cov	0.389	0.758	0.876	0.377	0.760	0.913
Tied Diag-Cov	0.397	0.816	0.906	0.379	0.821	0.942
Gaussianized Features – PCA = 10						
Full-Cov	0.326	0.733	**0.697**	0.319	0.719	**0.687**
Tied Full-Cov	**0.313**	0.720	0.793	**0.313**	0.717	0.796
Diag-Cov	0.334	0.772	0.851	0.330	0.772	0.889
Tied Diag-Cov	0.320	0.721	0.794	0.311	0.720	0.806
Gaussianized Features – PCA = 9						
Full-Cov	0.326	0.757	0.709	0.324	0.732	0.710
Tied Full-Cov	0.325	0.734	0.809	0.322	0.731	0.827
Diag-Cov	0.355	0.815	0.777	0.339	0.787	0.790
Tied Diag-Cov	0.322	0.728	0.808	0.316	0.747	0.833

Table 12 - MVG models comparison training on the training dataset and evaluating minDCF on the evaluation dataset. Left: Training with 4/5 of data. Right: Training with full data. Red: best results here. Blue: other corresponding validation table best results to see the difference.

Comments

- The results obtained on the evaluation dataset using 4/5 or the full training dataset are very consistent, which shows that using a 5-folds cross-validation was a good idea, at least in this case.
- We notice that the tied variants here perform slightly better than the full-cov ones, it is worth noting however that since the differences are quite small in both cases, it is not a dramatic decision.
- Another point is that here the diagonal variants work very well after applying PCA, this may suggest the presence of a difference between the training and the evaluation distributions, specifically, the evaluation data may have within-class covariance matrices more similar and diagonal.
- Consistently, Gaussianization helps achieving slightly better results than the raw features also in this case.
- We would have chosen the tied variant if we were to perform our decision on the evaluation dataset, but again, the results are quite similar.

Logistic Regression

We consider again the Regularized Logistic Regression models.

Regularized Linear Logistic Regression

We start with a linear logistic regression model using the best estimated lambda values and the best preprocess configurations for our target applications.

Model	***4/5 Train data***			***All train data***		
	$\tilde{\pi} = 0.5$	$\tilde{\pi} = 0.1$	$\tilde{\pi} = 0.9$	$\tilde{\pi} = 0.5$	$\tilde{\pi} = 0.1$	$\tilde{\pi} = 0.9$
Linear LR ($\lambda = 0.1, \pi_T = 0.5$) (Centering, White Cov, L2-Norm)	0.318	**0.699**	0.679	0.317	**0.682**	0.772
Linear LR ($\lambda = 0.01, \pi_T = 0.1$) (Gaussianized features)	**0.313**	0.702	0.832	**0.304**	0.731	0.853
Linear LR ($\lambda = 0.00001, \pi_T = 0.9$) (Centering, White Cov, L2-Norm)	0.344	0.766	**0.647**	0.345	0.763	**0.644**

Table 13 - Performance comparison of Linear Logistic Regression models on the evaluation dataset. There are only the best configurations. The table shows the minDCF calculated on the evaluation dataset while training on 4/5 (left) or the full (right) dataset.

Results are again mostly consistent with the ones obtained from the validation set. We can see a small improvement for our main target application while changing the embedded prior to π_T=0.1, this difference becomes slightly larger as we increase the number of training samples, however, since the difference is relatively small, this can be a neglectable difference. Results are similar to the linear MVG models also in this case. The other preprocess configurations perform worse than the ones considered, consistently with the results on the validation set. We rapidly see how different lambda values perform on the evaluation dataset with our best preprocess configuration, considering 4/5 of train data only.

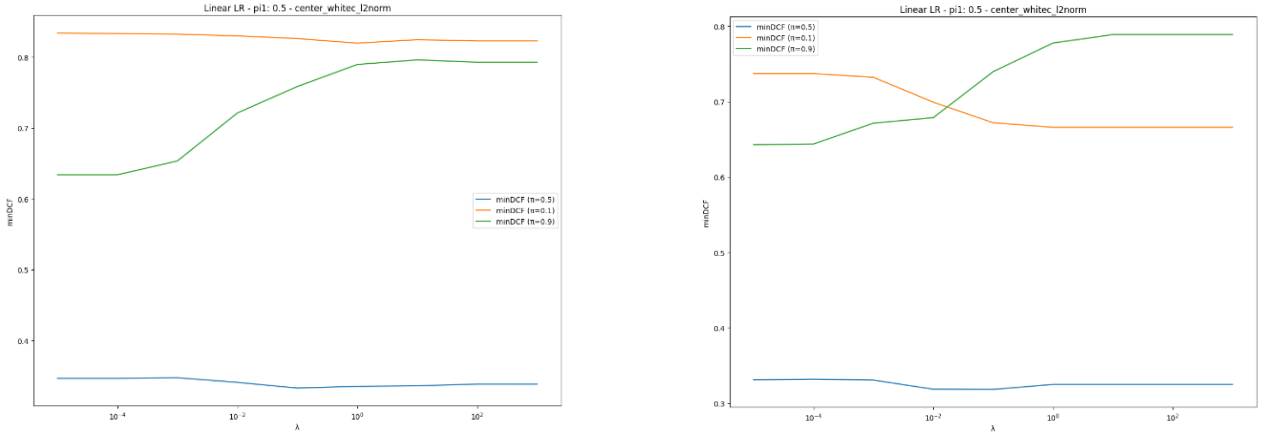


Figure 15 - Performance comparison using different values of lambda using the best preprocess configuration: centering, whitening the covariance matrix and then l2-normalization. (Left: validation set, Right: evaluation set)

Again, the results are very similar for our main target application, a different trend is shown for the target application with $\tilde{\pi}$=0.1. However, this plot would have led us to the same choice.

Regularized Quadratic Logistic Regression

We do the exact same analysis also for the quadratic logistic regression model. We use the best estimated lambda values and the best preprocess configurations for our target applications.

Model	***4/5 Train data***			***All train data***		
	$\tilde{\pi} = 0.5$	$\tilde{\pi} = 0.1$	$\tilde{\pi} = 0.9$	$\tilde{\pi} = 0.5$	$\tilde{\pi} = 0.1$	$\tilde{\pi} = 0.9$
Quadratic LR ($\lambda = 0.001, \pi_T = 0.5$) (Centering, White Within Cov, L2-Norm)	0.284	**0.619**	0.573	0.277	**0.593**	0.567
Quadratic LR ($\lambda = 0.001, \pi_T = 0.1$) (Gaussianization)	0.291	0.705	0.623	0.288	0.686	0.598
Quadratic LR ($\lambda = 0.01, \pi_T = 0.9$) (Centering, White Within Cov, L2-Norm)	0.302	0.645	0.578	0.298	0.638	0.591
Quadratic LR ($\lambda = 0.0001, \pi_T = 0.5$) (Centering, White Within Cov, L2-Norm)	0.269	0.701	**0.561**	0.261	0.643	0.568
Quadratic LR ($\lambda = 0.001, \pi_T = 0.5$) (Z-Normalization)	**0.263**	0.647	0.572	**0.256**	0.631	**0.539**

Table 14 - Performance comparison of Quadratic LR computing minDCF on the evaluation dataset after training with 4/5 (left) or the full (right) dataset.

The results are consistent with the ones on the validation set, we notice that, in this case, increasing the training data provides best results while maintaining the same relative trend; this is an expected behavior since quadratic separation surfaces can obtain more reliable estimates if fed with more non-disruptive samples. Moreover, the preprocess configuration consisting of applying z-normalization achieves convincing best results. This may be explainable by the difference on the covariance matrices of the evaluation data that we had a hunch from the previous MVG tied analysis. In fact, whitening the within covariance matrix can be less effective if the two covariance matrices are already very similar and diagonal. Z-Normalization, in contrast, makes data having a zero-mean and unit variance, making simpler for the quadratic separation surfaces to separate the classes since unbalanced dynamic ranges on the values of different dimensions are normalized. Embedding the different priors does not help in this case, even though also in the validation results we obtain small improvements from that. Lastly, we see that for our best estimated model for our main target application, we have a lower performance on the eval dataset (0.284) compared to the one estimated on the validation set (0.259), reflecting a distribution difference as we have already guessed from our previous considerations. We repeat the same analysis showing the performances over different values of lambda for our chosen preprocess configuration. We avoid showing the ones for the z-normalization model just because we have reported in the table the best lambda value, which happens to correspond to the best lambda for the other preprocess configuration.

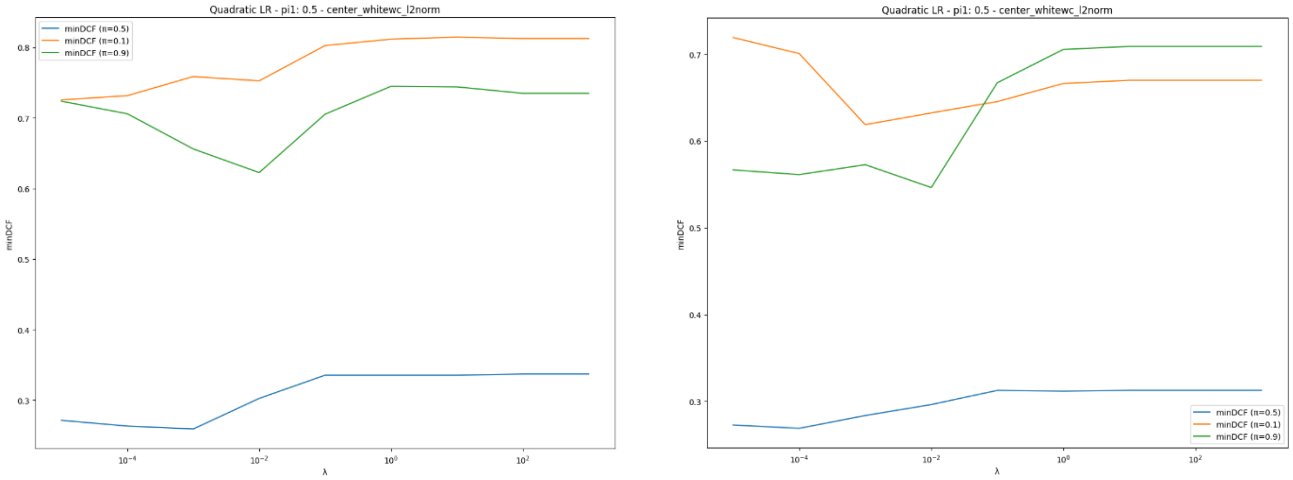


Figure 16 - Performance comparison using different values of lambda using the best preprocess configuration: centering, whitening the within-class covariance matrix and then l2-normalization. (Left: validation set, Right: evaluation set) (The models are trained over 4/5 of train data)

We can see from the plots and from the table that we would have obtained slightly better results if we had chosen a smaller value of lambda (10^{-4} rather than 10^{-3}). However, the difference is small, which means that our choice performs well also in this case.

Linear and Polynomial SVM

For conciseness, we show a comprehensive table reporting the results obtained on the evaluation set for the linear and polynomial SVM.

Model	***4/5 Train data***			***All train data***		
	$\tilde{\pi} = 0.5$	$\tilde{\pi} = 0.1$	$\tilde{\pi} = 0.9$	$\tilde{\pi} = 0.5$	$\tilde{\pi} = 0.1$	$\tilde{\pi} = 0.9$
Linear SVM (C=0.1, K = 1) Without class rebalancing (Gaussianization)	0.315	0.687	0.841	0.309	0.703	0.838
Linear SVM (C=0.1, K = 10) Class rebalancing $\pi_T = 0.5$ (Gaussianization)	0.322	0.696	0.773	0.321	0.694	0.790
Linear SVM (C=100, K = 10) Class rebalancing $\pi_T = 0.1$ (Gaussianization)	0.325	0.698	0.959	0.349	0.758	0.963
Linear SVM (C=1, K = 10) Class rebalancing $\pi_T = 0.9$ (Gaussianization)	0.395	0.911	0.656	0.399	0.936	0.648
Polynomial SVM (d = 3, C=0.1) (Centering, White Within Cov, L2-Norm)	0.273	0.629	0.570	0.282	0.587	0.573

Table 15 - Performance comparison of Linear and Polynomial SVM classifiers reporting only the best models chosen during the model selection phase, these are the minDCF results calculated on the evaluation dataset after training on 4/5 (left) or the full (right) dataset.

We also show the plots against C for our linear and polynomial (d=3) classifiers using our best preprocess configurations. We use only the training on 4/5 of the train dataset.

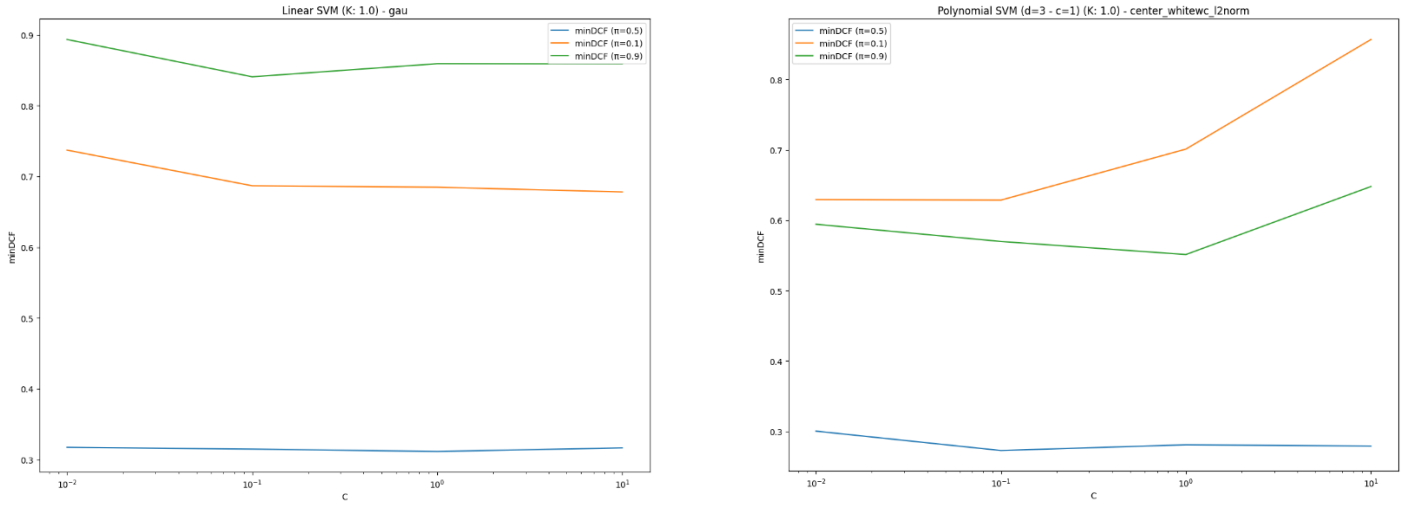


Figure 17 - Performance comparison over different values of C for the best linear SVM configuration and the best Polynomial svm configuration calculating minDCF on the evaluation dataset after training on 4/5 of train dataset.

Again, we see very consistent results with the ones referred to the validation set. Class-rebalancing does not affect the performances as we have seen also for the validation part. It is worth noting that we have done a complete analysis for all the configurations again and still the best hyperparameters are confirmed.

Radial Basis Function SVM

We now pass to analyze the two candidate models of our fusion, RBF SVM and GMM. We start with the RBF SVM by showing the performances over the evaluation dataset using the best estimated parameters.

Model	***4/5 Train data***			***All train data***		
	$\tilde{\pi} = 0.5$	$\tilde{\pi} = 0.1$	$\tilde{\pi} = 0.9$	$\tilde{\pi} = 0.5$	$\tilde{\pi} = 0.1$	$\tilde{\pi} = 0.9$
RBF SVM (γ = 8, C=0.5) **Without class-rebalancing** **(Centering, White Within Cov, L2-Norm)**	0.275	0.616	0.704	0.270	0.627	0.626
RBF SVM (γ = 10, C=0.1) **With class-rebalancing to $\pi_T = 0.1$** **(Centering, White Within Cov, L2-Norm)**	0.290	0.606	0.759	0.278	0.634	0.689
RBF SVM (γ = 10, C=1) **With class-rebalancing to $\pi_T = 0.9$** **(Centering, White Within Cov, L2-Norm)**	0.286	0.640	0.650	0.278	0.647	0.654
RBF SVM (γ = 0.1, C=1) **Without class-rebalancing** **(Gaussianization)**	**0.249**	0.628	**0.516**	**0.253**	0.642	**0.517**
RBF SVM (γ = 5, C=0.5) **Without class-rebalancing** **(Centering, White Within Cov, L2-Norm)**	0.255	**0.591**	0.563	0.258	**0.605**	0.527

Table 16 - Performances of the best RBF SVM models estimated during the cross-validation (black) and best models on the evaluation dataset (blue). The table contains minDCF calculated on the evaluation dataset after training on 4/5 (left) or the full (right) train dataset.

Surprisingly, the best hyperparameters estimated during the cross-validation on the training dataset perform very worse on the evaluation dataset. In particular, with the best hyperparameters for our main target application, we pass from a minDCF of 0.195 to 0.275, losing about 30% relative performance. We can see that in general all the obtained results on the evaluation dataset are worse, meaning that RBF SVM is not so good as our optimistic validation has shown on the training set during our optimization procedure. Moreover, from the conducted grid search, we can see that we pass from a minDCF of 0.195, the best result during our optimization, up to 0.249, the best result on the evaluation set, which shows a distribution difference that makes the evaluation data harder to be separated by an RBF SVM model. Lastly, we see that increasing the training data does not significantly improve the performances, an unexpected behavior since our RBF kernel should benefit from the presence of more non-disruptive data. For completeness, to compare the trends, we show some plots against C of our best estimated configurations. We use only the models trained on 4/5 of the train dataset to have a fair comparison and we consider only our main target application.

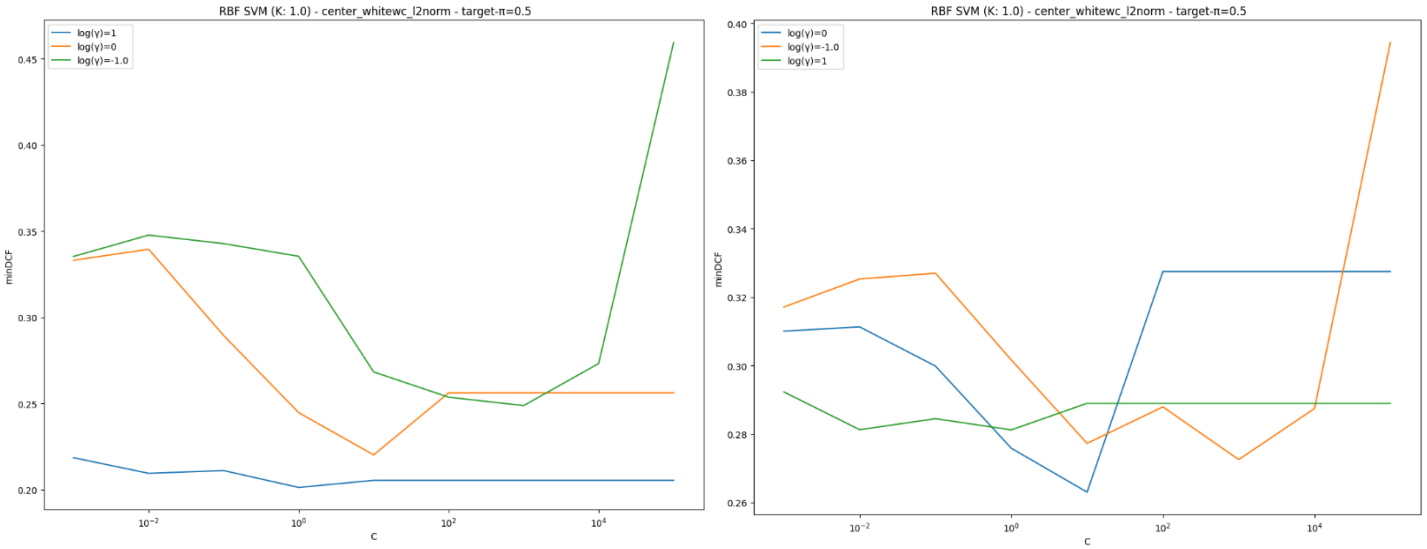


Figure 18 - Plot against C for our best estimated configuration and our main target application. Left: validation set. Right: evaluation set. (colors are different!)

Paying attention at the colors (they are different in the two plots), they show a similar trend for some values of gamma, with some differences for some other values of it. However, the evaluation plots are in general translated up towards worse performance. To have a better look at what is happening, we show the plots obtained by a fine-grained search.

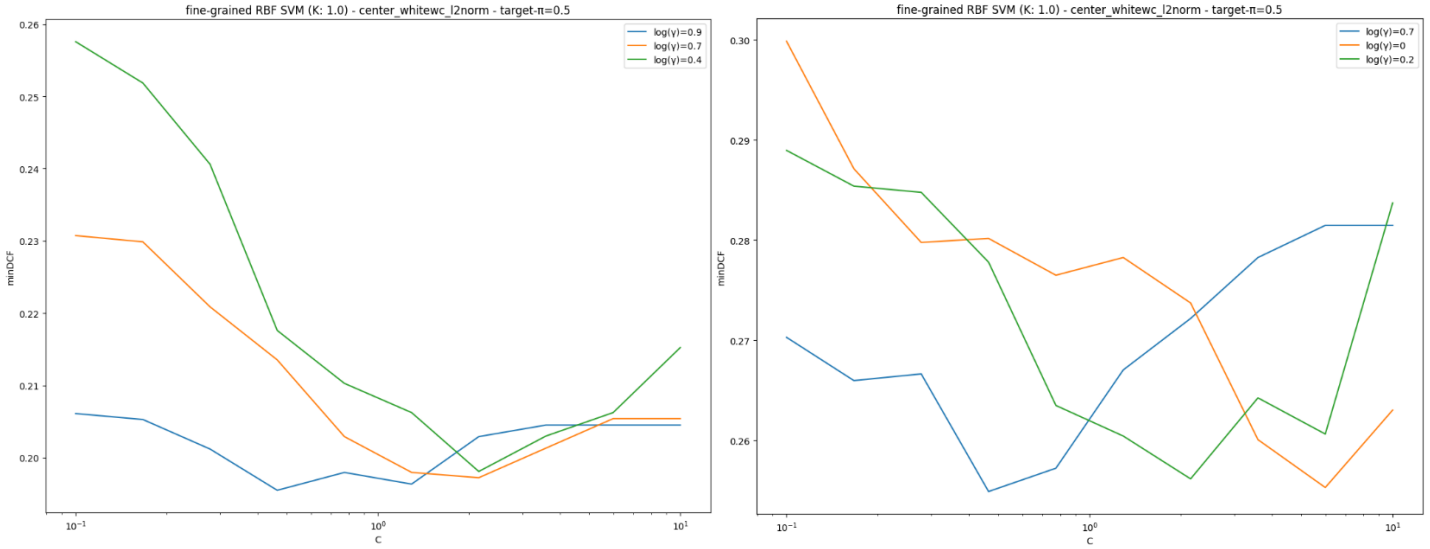


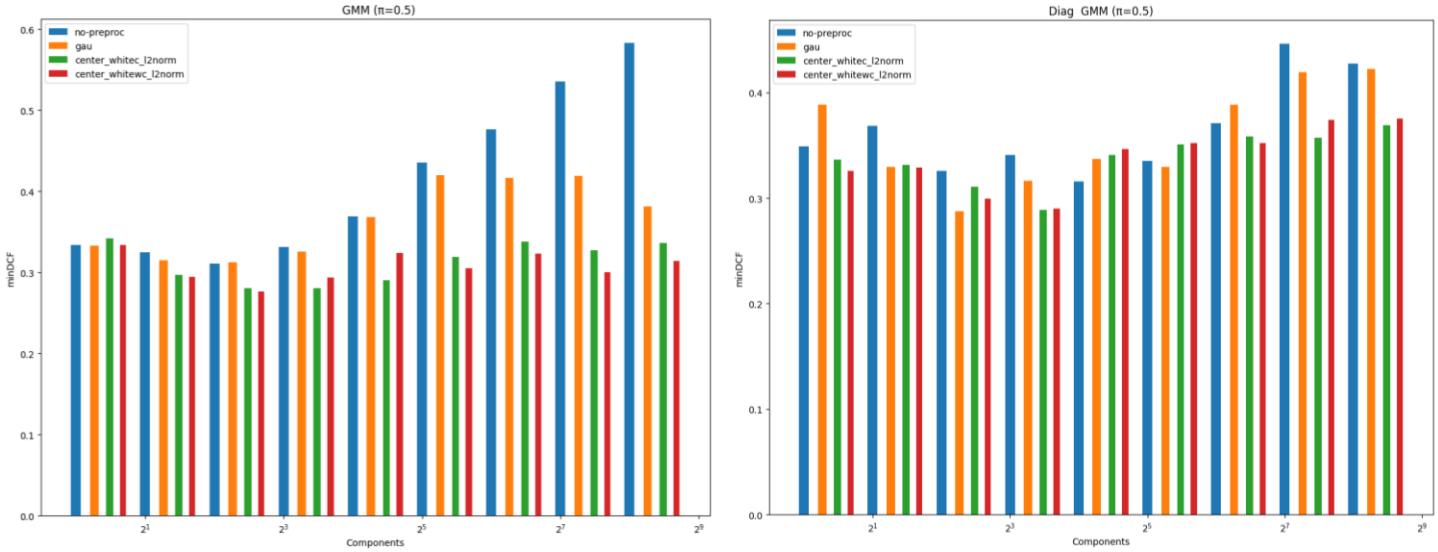
Figure 19 - Plot against C (fine-grained search) for our best estimated configuration and our main target application. Left: validation set. Right: evaluation set. (colors and gamma values are different!)

The plots report different values of gamma (top-three gamma values), however, the left orange and right blue plots correspond and show a significant difference on the slope for values of C greater than 1.

However, in terms of optimization, we could have chosen a different configuration of hyperparameters to improve the performance, still the gap remains small, making not a huge difference in the end. Overall, unlike in our optimization procedure, our model performs worse and similarly as our quadratic logistic regression.

Gaussian Mixture Model

We proceed with analyzing the gaussian mixture model family, our last individual model type. We first show the plots of the different model types, considering the combination of full-cov, tied and diagonal variants. Again, they are referred to a training on 4/5 of the train dataset. We consider only our main target application.



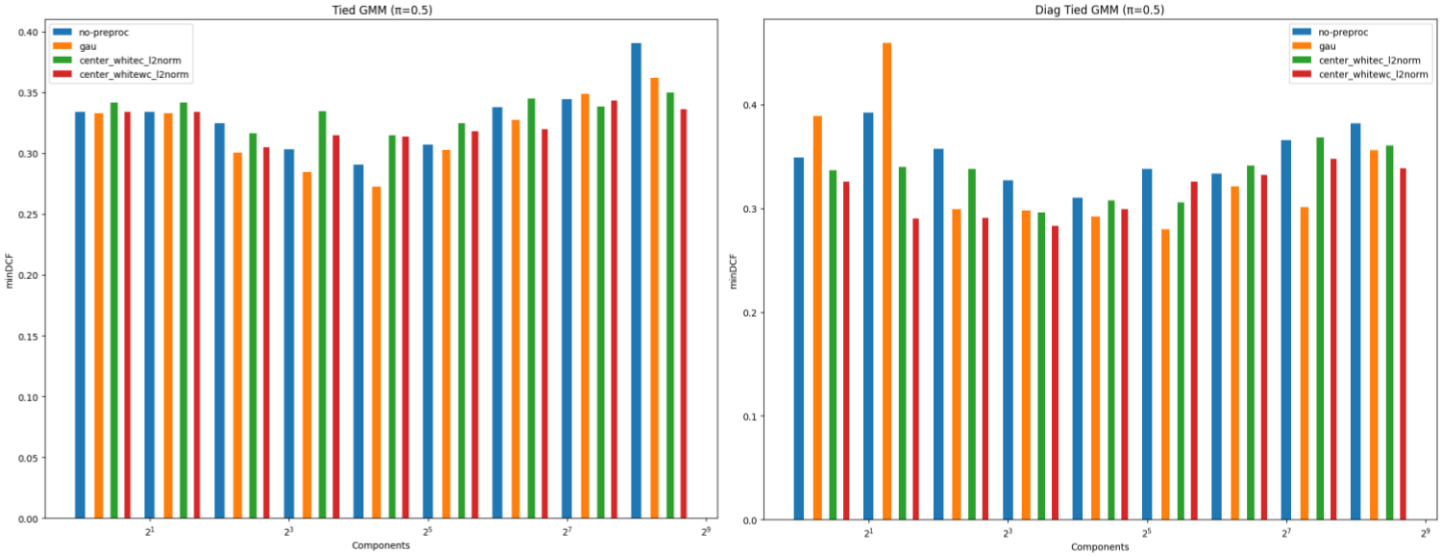


Figure 20 - Plots of performances of different GMM models as we increase the number of components, only for the main target application. The values are minDCF calculated on the evaluation dataset after training on 4/5 of the training dataset. Top-left: full-cov; Top-right: diag-cov; Bottom-left: tied-cov; Bottom-right: diag&tied-cov.

As we can see from these graphs compared with the Figure 10, there is again a significant difference both in absolute minDCF average value and in the trend. In fact, as we increase the number of components, we start losing performance, probably due to overfitting. We have seen that for our training dataset, during our optimization procedure, our preprocessing strategy is effective to reduce overfitting and improve the performance even with 256 components. On the other hand, this does not apply for the evaluation dataset, where the preprocessing strategy is not effective to reduce overfitting, having good results only with very small number of components (4-8). We also notice that tied and diagonal variants provide in proportion better results compared to the ones obtained on the validation set. Our hypothesis on the diversity of the two distributions seem to be confirmed by this analysis.

Model	***4/5 Train data***			***All train data***		
	$\tilde{\pi} = 0.5$	$\tilde{\pi} = 0.1$	$\tilde{\pi} = 0.9$	$\tilde{\pi} = 0.5$	$\tilde{\pi} = 0.1$	$\tilde{\pi} = 0.9$
GMM (Full-Cov, 256 comps) (Centering, White Within Cov, L2-Norm)	0.314	0.808	0.612	0.328	0.790	0.597
GMM (Full-Cov, 4 comps) (Centering, White Within Cov, L2-Norm)	0.277	0.647	0.662	0.282	0.627	0.643

Figure 21 - Performances of the best estimated GMM during optimization procedure and best GMM on evaluation dataset. The table reports minDCFs values obtained on the evaluation dataset after training with 4/5 (left) or the full (right) train dataset.

As we can see, we could have obtained a very better result if we had used 4 components instead of 256. It is worth noting that increasing the number of training samples degrades the performance, which is a good signal that our hypothesis on overfitting is true. The difference in the evaluation distribution does not allow to effectively preprocess the data in a space where the GMM can grow up in the number of components without starting to lose performance. This may suggest the presence of disruptive data in the training set that deviates our estimated class-conditional likelihood in a wrong direction. Similar considerations can be extended to the other two target applications, with the exception of the target application with $\tilde{\pi}$=0.9, which maintains the same performance.

Model fusion

Lastly, we consider our model fusion using the best models estimated on the optimization procedure, even if GMM is working bad for the evaluation dataset.

Model	*4/5 Train data*		
	$\tilde{\pi} = 0.5$	$\tilde{\pi} = 0.1$	$\tilde{\pi} = 0.9$
RBF SVM (γ = 8, C=0.5) **Without class-rebalancing** **(Centering, White Within Cov, L2-Norm)**	0.275	0.616	0.704
GMM (Full-Cov, 256 comps) **(Centering, White Within Cov, L2-Norm)**	0.314	0.808	0.612
Fusion	0.285	0.639	0.579

Figure 22 - minDCF values calculated on the evaluation dataset after training with 4/5 of the train data and performing the fusion on the remaining 1/5.

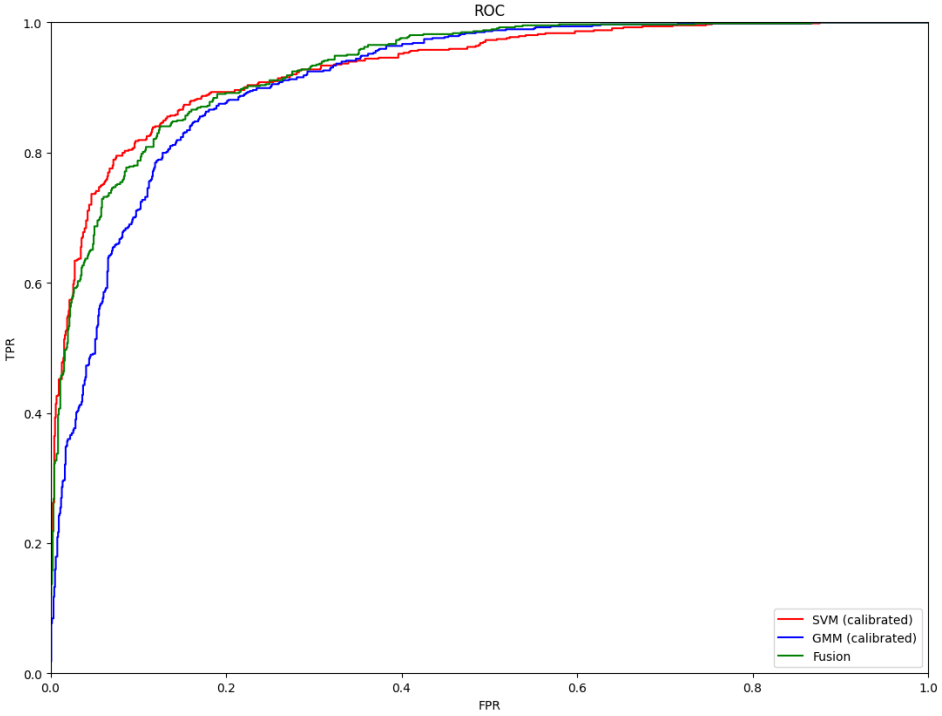


Figure 23 - ROC curve comparing the best estimated models and its fusion.

Again, results are quite consistent with our expectations, we see a small difference in the ROC plot as we increase the TPR showing that GMM outperforms SVM in those cases. Summing up, the fused model is considerably better when we want high rates of true positives, which can be a quite interesting feature as the task we are trying to cope with could reflect this scenario in real use cases. We notice again a performance drop when we pass from validation minDCF to evaluation minDCF, consistently with our previous analyses.

Scores Calibration

As we did in the last step of our optimization and model selection procedure, we are now interested in assessing if recalibrating the scores on a portion of the training dataset is working good for our evaluation dataset. As we have seen, we can either pick the threshold which gives the minimum DCF or re-calibrate the scores with a logistic regression model. As we have analyzed the second approach, we repeat our analysis on the evaluation set. For our analyses, we train our models on 4/5 of the train data, then we use the remaining 1/5 of the data to train the score recalibration model and lastly, we generate our scores over the evaluation dataset with the uncalibrated models, and we proceed by using the recalibration model to recalibrate them. As we need to embed a target application effective prior in our recalibration model to recover log-likelihood ratios, we embed only the main target application prior $\tilde{\pi}$=0.5.

We first show how uncalibrated and calibrated models are performing on the evaluation set.

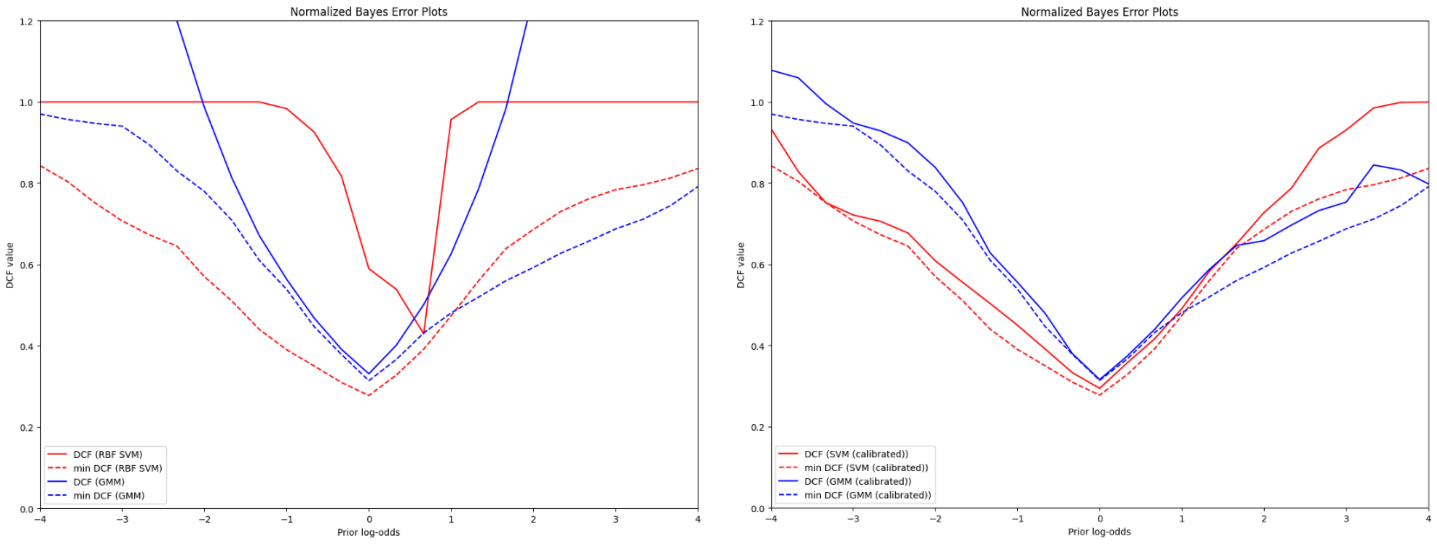


Figure 24 - Normalized Bayes Error plots on the evaluation dataset. Left: uncalibrated candidate models. Right: calibrated candidate models. ($\tilde{\pi}$=0.5)

Focusing on the calibration reasoning only, again the achieved results are very similar, with an effective recalibration for the two candidate models, working better for a wide range of applications.

As our fusion model should produce already well-calibrated scores, we assess it with another plot.

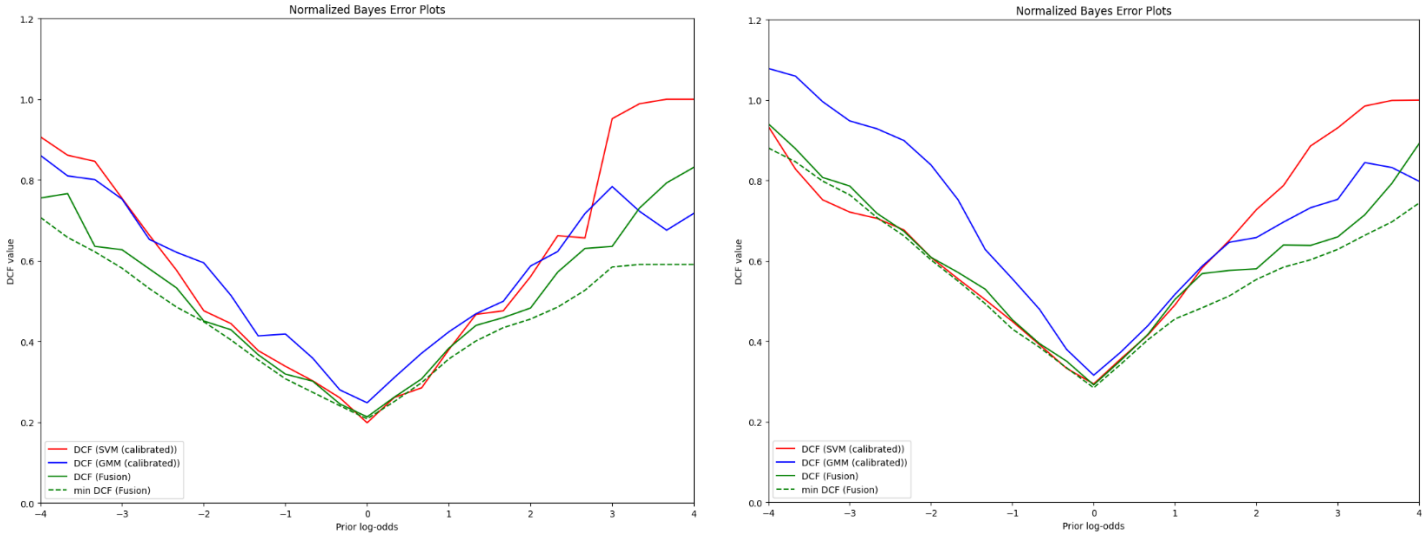


Figure 25 - Bayes Error Plots of the calibrated models and the model fusion. Left: validation set. Right: evaluation set. ($\tilde{\pi}$=0.5)

As the plots show, results are again very similar. However, we notice that for the evaluation set, the calibrated SVM outperforms the fusion for very negative values of the prior log odds.

Model	$\tilde{\pi} = 0.5$		$\tilde{\pi} = 0.1$		$\tilde{\pi} = 0.9$	
	minDCF	*actDCF*	*minDCF*	*actDCF*	*minDCF*	*actDCF*
RBF SVM (calibrated)	**0.278**	0.294	**0.615**	**0.650**	0.713	0.768
GMM (calibrated)	0.314	0.316	0.808	0.879	0.612	0.699
Fusion	0.285	**0.292**	0.639	**0.650**	**0.579**	**0.592**

Table 17 - minDCF and actual DCF of the calibrated models and the fused one. The values are calculated on the evaluation dataset.

As the table shows, the results are very consistent with our validation scores, indeed, the results on the evaluation dataset are in proportion worse than the ones obtained from the validation set; however, this only means that our validation set has provided an over-optimistic estimate of the metric values that we would have obtained if we had

tested our models on the evaluation set. In fact, these results show and confirm our decision we have made during the model selection procedure. The fusion still obtains more well-calibrated scores over a wide range of target applications.

Conclusion

To wrap up, our proposed fusion model is effective for the proposed task, achieving a normalized DCF cost of ≈ 0.29 for our main target application with $\tilde{\pi}=0.5$. The model is working fairly good also on the unbalanced considered target applications, achieving a normalized DCF cost of ≈ 0.6. To give a practical sense to the numbers, a normalized DCF cost of ≈ 0.29 on a target application with $\tilde{\pi}=0.5$ corresponds to an error rate of ≈ 14.5 %. From the analysis we have seen that we generally obtain over-optimistic results on the validation set, but fortunately the results on the evaluation dataset often lead to estimate similar model types and parameters. Moreover, we have seen that diagonal and tied variants of generative models perform in proportion slightly well on the evaluation dataset, which could mean that the evaluation distribution has more similar and diagonal within-class covariance matrices. From the RBF SVM analysis has emerged that the model does not improve as we increase the number of training samples, demonstrating the possible presence of disruptive data in the training dataset. The hypothesis is reinforced by the GMM analysis, where training and testing on the training data, in contrast with testing on the evaluation data, provide very different results as we increase the number of components. In fact, as we try to better fit our training data distribution, the results on the evaluation data drop, meaning that the two distributions differ from each other.