WINE QUALITY DETECTION
Machine learning and pattern recognition

MARCO SAPIO - 291691
s291691@studenti.polito.it
January 31, 2023

Contents

1. Introduction

The datasets are related to red and white variants of the Portuguese "Vinho Verde" wine. Due to privacy and logistic issues, only physicochemical (inputs) and sensory (the output) variables are available (e.g. there is no data about grape types, wine brand, wine selling price, etc.).

These datasets can be viewed as classification or regression tasks. The classes are ordered and not balanced (e.g. there are many more normal wines than excellent or poor ones). Outlier detection algorithms could be used to detect the few excellent or poor wines. Also, we are not sure if all input variables are relevant. So it could be interesting to test feature selection methods.

NOTE: For the project the classes have been binarized - Feature 12 is simply 0 (low quality, ≤ 5) or 1 (high quality, ≥ 7). To simplify the problem, wines with quality 6 have been removed. Red and white wines have been merged into a single dataset

2. Feature Analysis

Each candidate is described by 11 continuous variables. The features are summarised below:

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

Output variable (based on sensory data):

12 quality (score between 0 and 10)

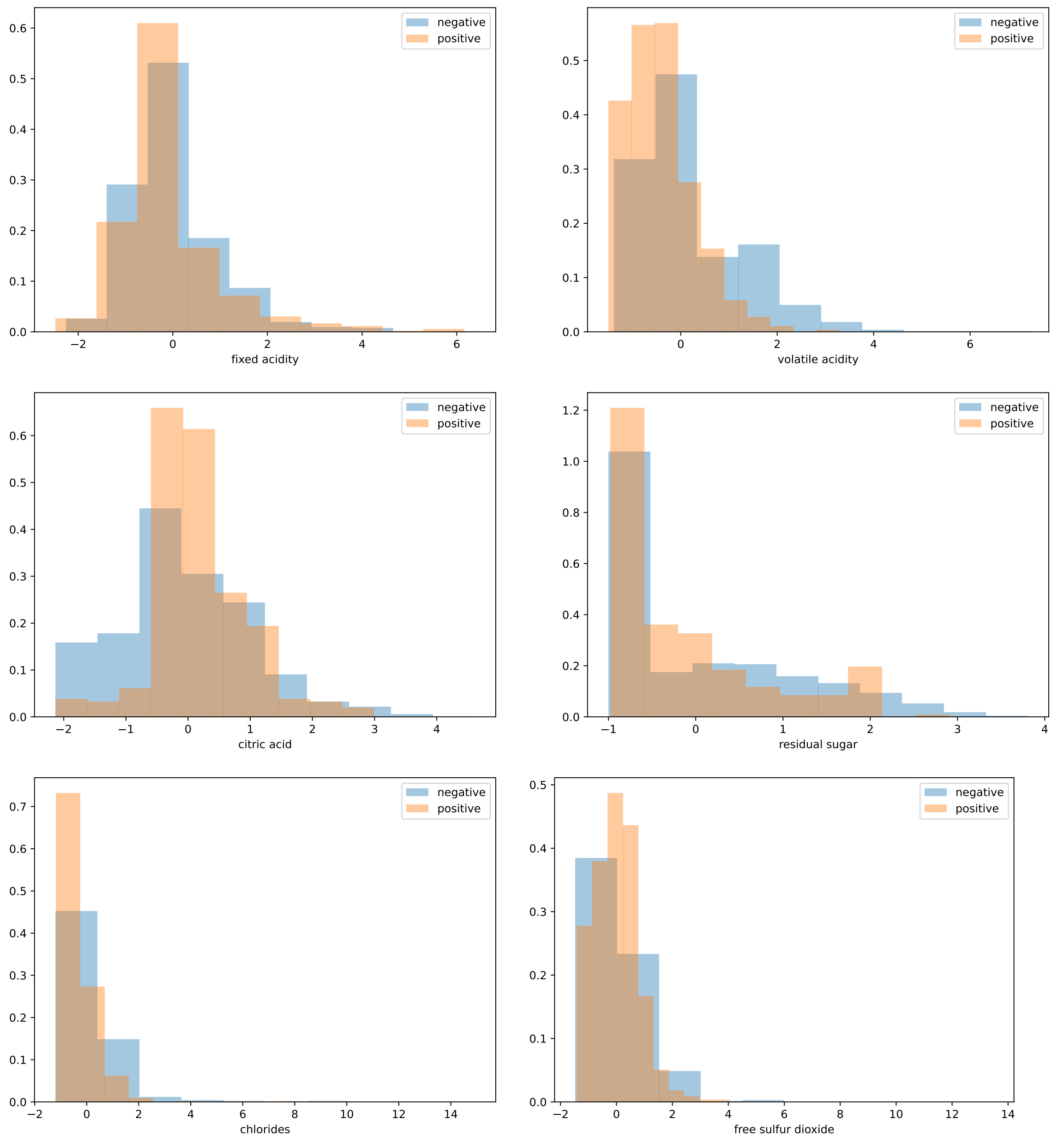
A preliminary analysis of the training data shows that, in many cases, the raw features have irregular distributions, characterized by a presence of significantly large outliers.

Due to the presence of outliers, we expect that classification approaches may produce sub-optimal results (especially Gaussianization methods).

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 processing consists in mapping the features to a uniform distribution and then transforming the mapped features through the inverse of Gaussian cumulative distribution function
We therefore further pre-process data by "Gaussianizing" the features

Let x be the feature we want to transform We first compute its rank over the training dataset:

$$r(x) = \left(\sum_{k=1}^{N} [x_i < x] + 1\right) \tag{1}$$



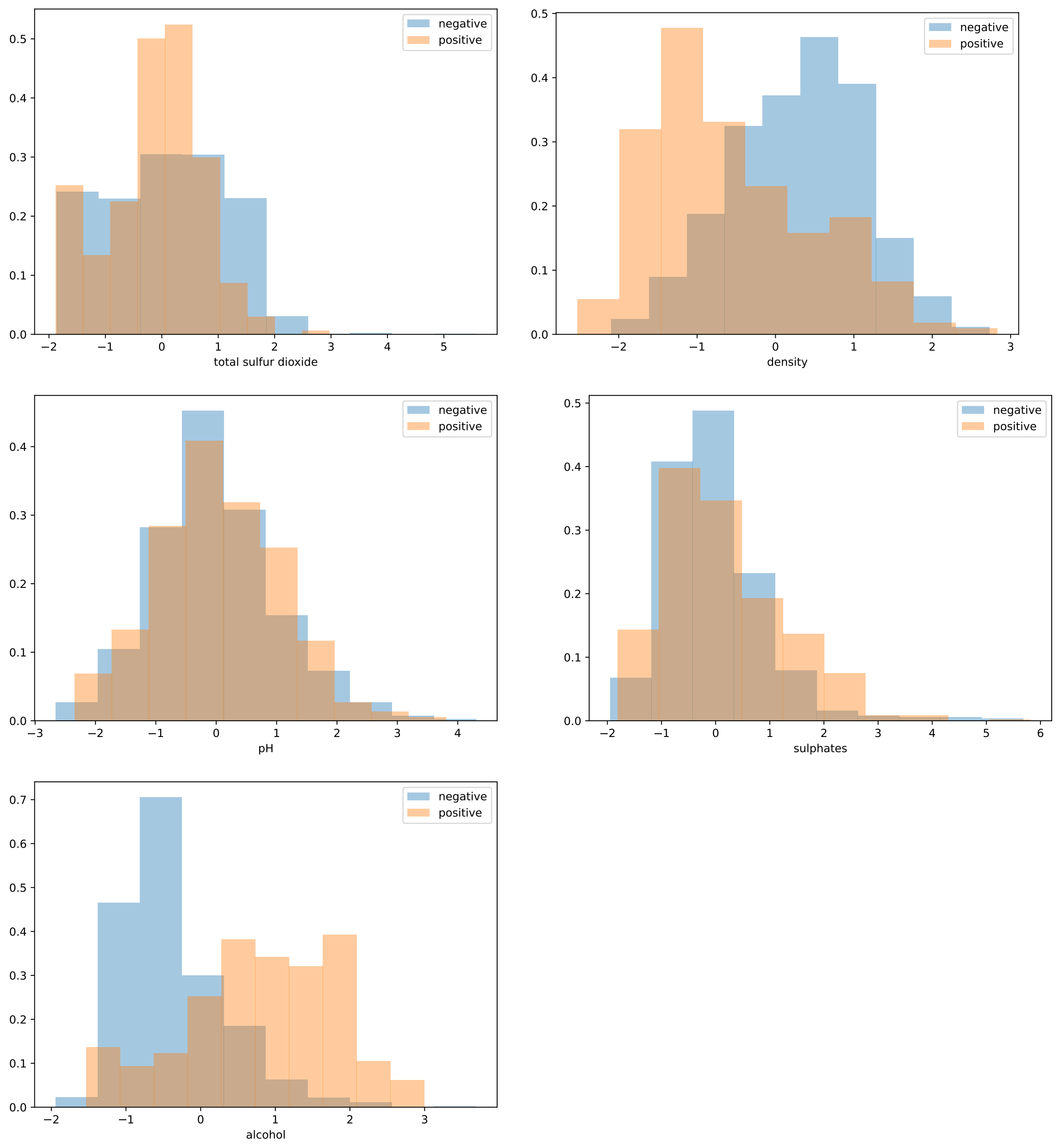
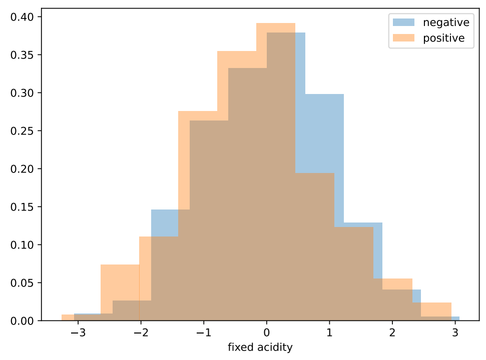
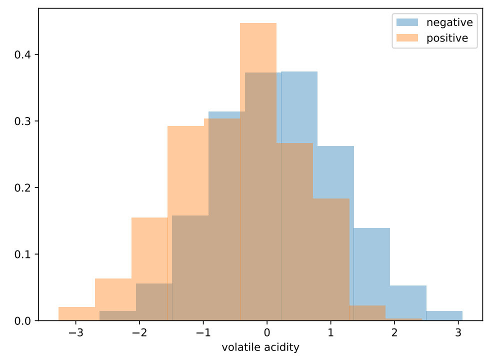
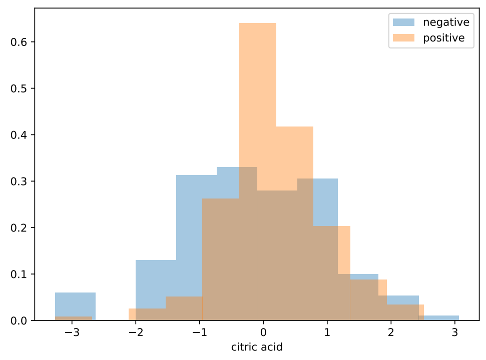
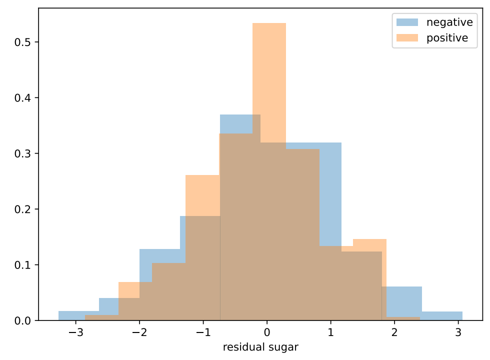


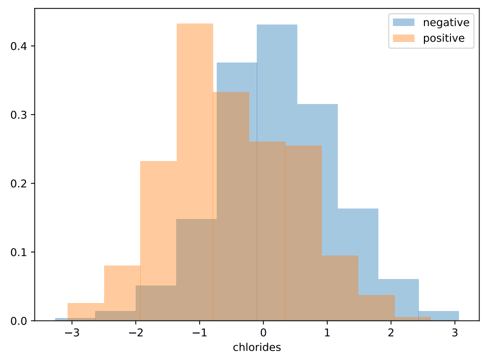
Figure 1: Histogram of the (modified) Wine dataset features (training set).

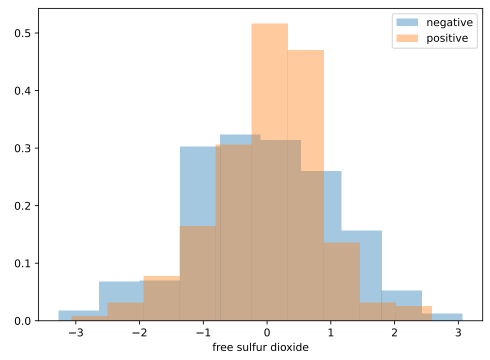

negative
positive
fixed acidity

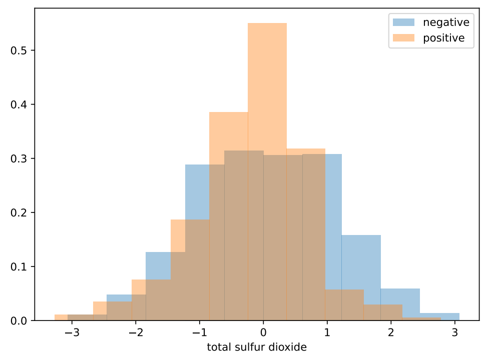

negative
positive
volatile acidity

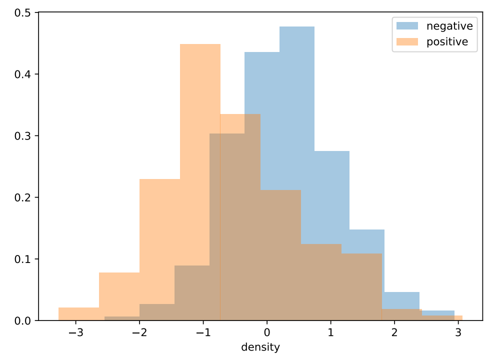

negative
positive
citric acid


negative
positive
residual sugar


negative
positive
chlorides


negative
positive
free sulfur dioxide


negative
positive
total sulfur dioxide


negative
positive
density

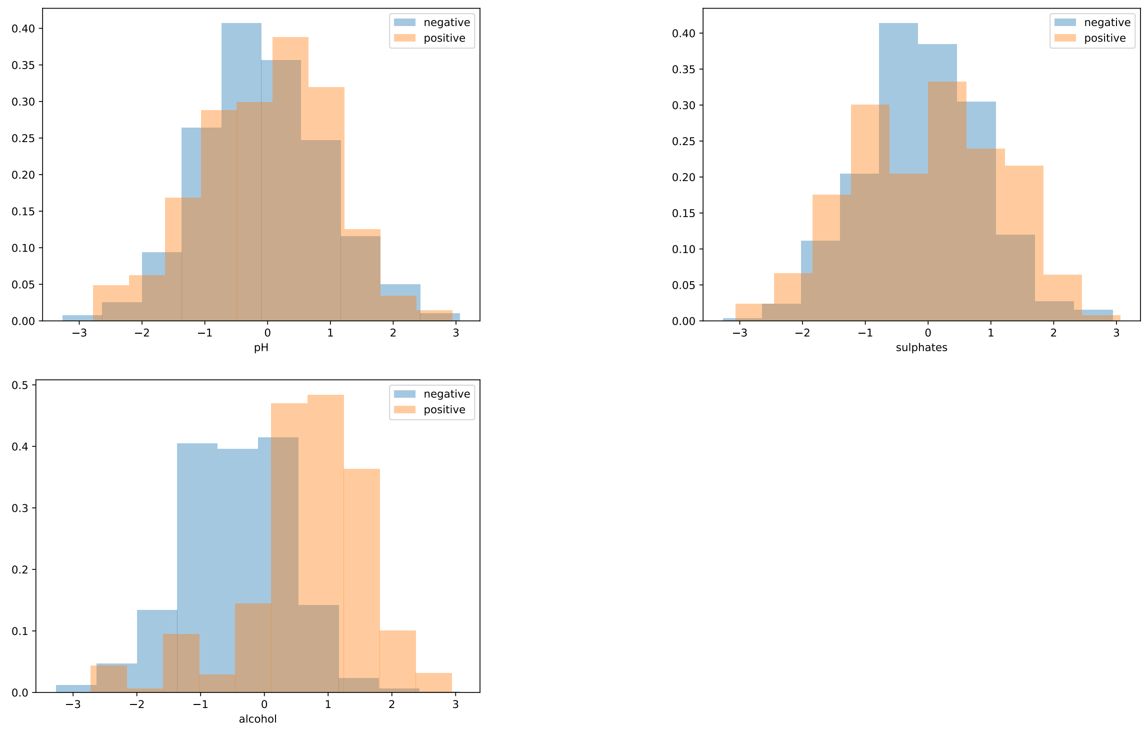


Figure 2: Histogram of the (modified) Wine dataset features (training set) after Gaussianization.

Gaussianzed features shown seem well distributed, with no evident outliers. For this reason, no further pre-processing has been considered. Furthermore, heatmaps of the Gaussianized training set features have been plotted to analyse features correlation. The heatmaps show the Pearson correlation coefficient:

$$\frac{Cov(X,Y)}{\sqrt{Var(X)}\sqrt{Var(Y)}} \tag{2}$$

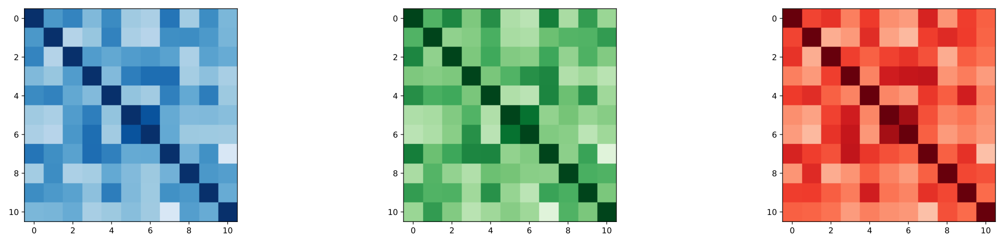


Figure 3: Heat maps: Blue: whole dataset. Green: low quality. Red: high quality

The heat map that displays the Pearson Correlation Coefficient shows us that some characteristics are related to each other (e.g., 5-6). While this is true, the correlation is not strong enough to use a feature size reduction technique such as PCA in our models.

3. Classification

In order to perform an analysis and understand which model is most promising we will use the k-fold cross validation.

Cross-validation is a statistical method used to estimate the skill of machine learning models.
It is commonly used in applied machine learning to compare and select a model for a given predictive modeling problem because it is easy to understand, easy to implement, and results in skill estimates that generally have a lower bias than other methods.

Advantages are:

- More data available for training and validation
- The final classifier will be obtained by re-training over the the whole training set, so it will leverage additional data
- Decisions are made over the validation set for the models trained using folds. They may not be optimal for the model learned from all training data

The K-Fold is implemented with K = 5.
We will consider 3 application: one balanced and the other two unbalanced.

$$(\tilde{pi}; C_{fp}; C_{fn}) = (0.5; 1; 1)$$
$$(\tilde{pi}; C_{fp}; C_{fn}) = (0.9; 1; 1)$$
$$(\tilde{pi}; C_{fp}; C_{fn}) = (0.1; 1; 1)$$

Our main application is the balanced one.
In order to find the most promising approach we measure performances with minimun detection function that represents the cost we would pay if we made optimal decisions for the test set using the recognizer scores.

3.1 Gaussian Classifier

In this first part we consider Gaussian Classifiers assuming samples are independent distributed according to $X_t, C_t \sim X, C$.
The purpose is to assign the class with highest posterior probability

$$c_t^* = argmax_c P(C_t = c | X_t = x_t)$$

The classifiers used are four:

- MVG classifier: We have one mean and one covariance matrix per class.

- MVG Naive Bayes classifier: the Naive Bayes version of the MVG is simply a Gaussian classifier where the covariance matrices are diagonal.
- MVG classifier with tied covariances: In this case, the class covariance matrices
- MVG classifier with Tied Diagonal covariances. are tied, with $\Sigma_c = \Sigma$.

The following table shows the results in terms of minimum DCF of the studied Gaussian classifiers

	$\tilde{\pi} = 0.5$	$\tilde{\pi} = 0.1$	$\tilde{\pi} = 0.9$
Gaussianized Features - No PCA			
Full Cov	0.310	0.793	0.789
Diag Cov	0.448	0.851	0.898
Tied Cov	0.355	0.876	0.944
Tied Diag-Cov	0.451	0.876	0.944
Gaussianized Features - PCA(m = 10)			
Full Cov	0.303	0.793	0.714
Diag Cov	0.402	0.815	0.851
Tied Full Cov	0.355	0.819	0.866
Tied Diag-Cov	0.355	0.811	0.886
Gaussianized Features - PCA(m = 9)			
Full Cov	0.311	0.825	0.727
Diag Cov	0.404	0.856	0.827
Tied Full Cov	0.357	0.815	0.845
Tied Diag-Cov	0.359	0.818	0.866
Gaussianized Features - PCA(m = 8)			
Full Cov	0.310	0.844	0.745
Diag Cov	0.412	0.898	0.833
Tied Full Cov	0.382	0.854	0.890
Tied Diag-Cov	0.379	0.867	0.917
Gaussianized Features - PCA(m = 7)			
Full Cov	0.322	0.832	0.757
Diag Cov	0.422	0.899	0.853
Tied Full Cov	0.381	0.876	0.894
Tied Diag-Cov	0.389	0.881	0.907

	$\tilde{\pi} = 0.5$	$\tilde{\pi} = 0.1$	$\tilde{\pi} = 0.9$
Raw Features - NO PCA			
Full Cov	0.314	0.779	0.844
Diag Cov	0.421	0.846	0.922
Tied Full	0.334	0.811	0.748
Tied Diag-Cov	0.403	0.866	0.932
Raw Features - PCA = 10			
Full Cov	0.323	0.798	0.855
Diag Cov	0.390	0.815	0.904
Tied Full	0.333	0.826	0.758
Tied Diag-Cov	0.341	0.823	0.775
Raw Features - PCA = 9			
Full Cov	0.328	0.812	0.822
Diag Cov	0.390	0.805	0.844
Tied Full	0.333	0.821	0.760
Tied Diag-Cov	0.340	0.823	0.777
Raw Features - PCA = 8			
Full Cov	0.350	0.824	0.853
Diag Cov	0.396	0.837	0.905
Tied Full	0.380	0.854	0.860
Tied Diag-Cov	0.377	0.835	0.861
Raw Features - PCA = 7			
Full Cov	0.359	0.815	0.874
Diag Cov	0.393	0.832	0.896
Tied Full	0.375	0.831	0.862
Tied Diag-Cov	0.375	0.832	0.866

As we see, the diagonal models (both full diagonal and tied diagonal) produce worse results if compared to the other models with or without Gaussianization and PCA. These models work under the naive-bayes-assumption, according to which the different components for each class are uncorrelated So, this assumption does not produce accurate results. Applying PCA slightly improves the performance, probably because the within-class correlation decreases by removing the low variances directions.

The tied full covariance model performs pretty good on the validation data, confirming the similarity between classes shown by the correlation analysis. It performs better than the diagonal models accounting the within-class correlations. In this case, PCA does not help producing better results. The best-performing model is the full covariance model, which is able to account for correlations. What is more, having enough data compared to the dimensionality of the samples, we succeed in obtaining robust results.

Gaussianization performs slightly better if compared to raw features. Even if it does not help to increase much the performances, PCA can still be used to reduce the number of parameters and therefore to reduce the complexity of the model.

To sum up, the best candidate is currently the MVG model with full covariance matrices. The chosen one is the one with Gaussianized features, since it shows slightly better results than the one with raw features.

3.2 Logistic Regression

In this section, We start considering regularized Logistic Regression.
Here, rather than modeling the distribution of observed samples $X|C$, we directly model the class posterior distribution $C|X$.

Since classes are not balanced, we re-balance the costs of the different classes, minimizing

$$J(w, b) = \frac{\lambda}{2} \left\|w^2\right\| + \frac{\pi_T}{n_T} \sum_{i==1|c_t==i}^{N} log(1 + e^{(-z_i(w^T x_i + b))}) + \frac{1-\pi}{n_F} \sum_{i==1|c=1}^{N} log(1 + e^{(-z_i(w^T x_i + b))})$$
(3)

We can start comparing the models using different values of λ for the main application.

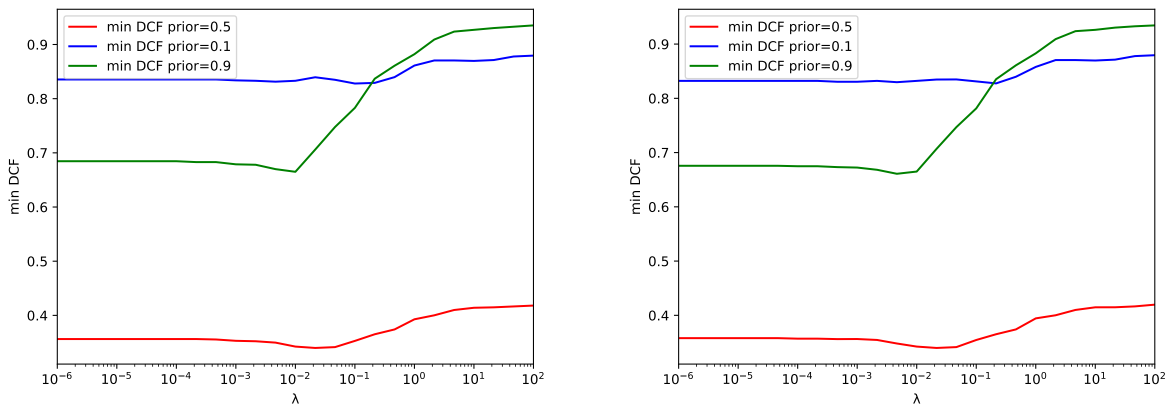


Figure 4: Linear Logistic Regression on Z-Normalized features: min DCF for different values of λ; left:No PCA and right: PCA (m = 10)

It is evident that for Raw and Gaussianized Data the main application give better results, followed by the one with prior = 0.9 and the worse one with prior = 0.1.

Of course, if λ is too large, we will obtain a solution that has small norm, but is not able to well separate the classes.

On the other hand, if λ is too small, we will get a solution that has good separation on the training set, but may have poor classification accuracy for unseen data (i.e. poor generalization).

For those reasons a good value of λ could be 10^{-3} Also in this case results do not differ much between No PCA and PCA m = 10. Below the table containing the results for differently balanced models with $\lambda = 10^{-2}$ and 0 .
We can also consider training using a different prior π_T to see model behaviours.

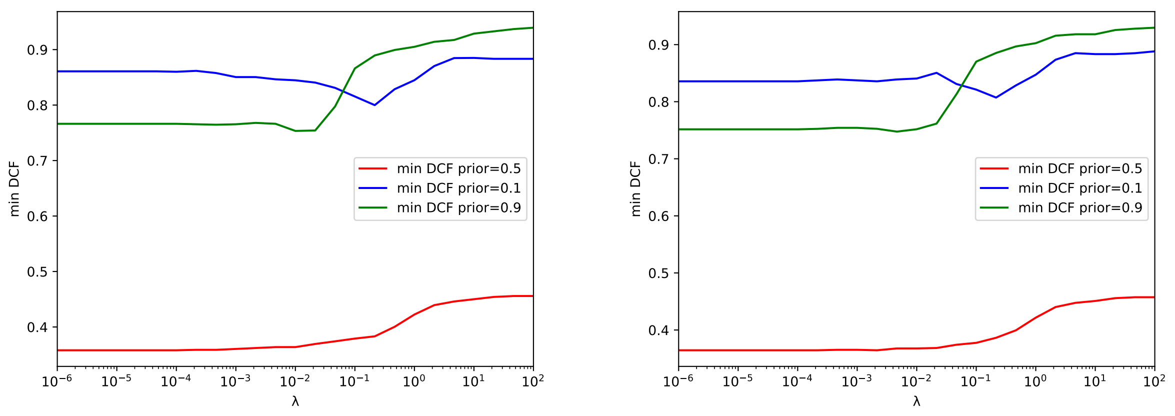


Figure 5: Linear Logistic Regression on Gaussianized features: min DCF for different values of λ; left:No PCA and right: PCA (m = 10)

	$\tilde{\pi} = 0.5$	$\tilde{\pi} = 0.1$	$\tilde{\pi} = 0.9$
Raw Data			
LR (λ =10^{-2}, $\pi_T = 0.5$)	0.342	0.833	0.665
LR (λ =10^{-2}, $\pi_T = 0.1$)	0.339	0.818	0.776
LR (λ =10^{-2}, $\pi_T = 0.9$)	0.354	0.845	0.691
Raw Data - PCA = 10			
LR (λ =10^{-2}, $\pi_T = 0.5$)	0.342	0.832	0.665
LR (λ =10^{-2}, $\pi_T = 0.1$)	0.341	0.818	0.774
LR (λ =10^{-2}, $\pi_T = 0.9$)	0.354	0.844	0.685
Gauss Data - NO PCA			
LR ($\lambda = 0, \pi_T = 0.5$)	0.364	0.845	0.753
LR ($\lambda = 0, \pi_T = 0.1$)	0.347	0.793	0.935
LR ($\lambda = 0, \pi_T = 0.9$)	0.374	0.897	0.699
Gauss Data - PCA = 10			
LR ($\lambda = 0, \pi_T = 0.5$)	0.368	0.841	0.752
LR ($\lambda = 0, \pi_T = 0.1$)	0.344	0.802	0.923
LR ($\lambda = 0, \pi_T = 0.9$)	0.375	0.892	0.690

Comparing these results with the best linear MVG classifier (Tied covariance model), the logistic regression performed slightly worse. Since the full covariances MVG corresponds to quadratic separation rules, while logistic regression corresponds to a linear one we can repeat the analysis quadratic logistic regression model and we can expect better results.

In order to confirm our assumption, we will focus on Quadratic Logistic Regression:

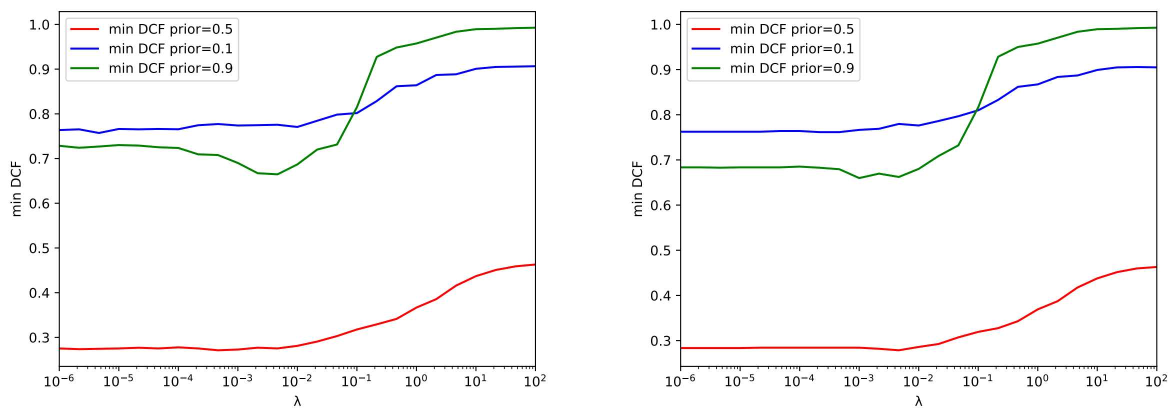


Figure 6: Quadratic Linear Logistic Regression on Z-Normalized features: min DCF for different values of λ; left:No PCA and right: PCA (m = 10)

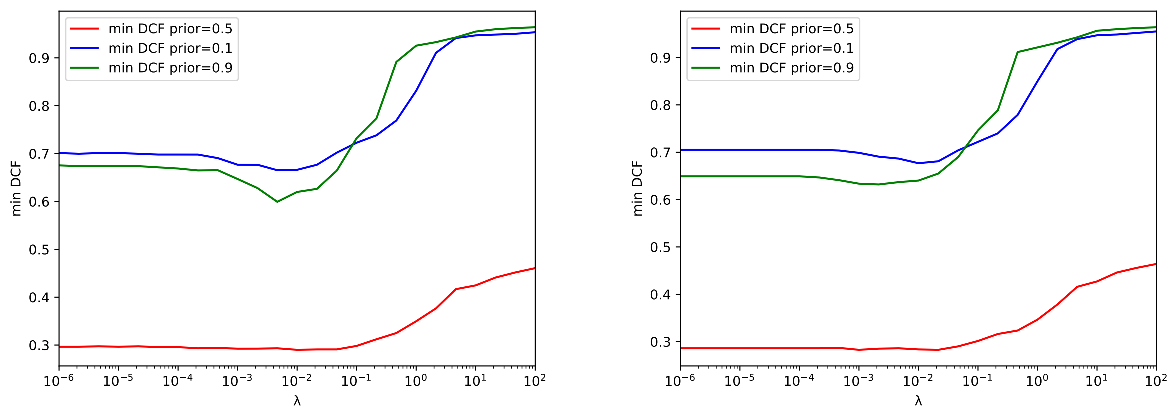


Figure 7: Quadratic Linear Logistic Regression on Gaussianized features: min DCF for different values of λ; left:No PCA and right: PCA (m = 10)

	$\tilde{\pi} = 0.5$	$\tilde{\pi} = 0.1$	$\tilde{\pi} = 0.9$
Gauss features			
QuadLR (λ =10^{-2}, $\pi_T = 0.5$)	0.290	0.666	0.620
QuadLR (λ =10^{-2}, $\pi_T = 0.1$)	0.296	0.673	0.673
QuadLR (λ =10^{-2}, $\pi_T = 0.9$)	0.306	0.732	0.579
Gauss features - PCA = 10			
QuadLR (λ =10^{-2}, $\pi_T = 0.5$)	0.284	0.677	0.640
QuadLR (λ =10^{-2}, $\pi_T = 0.1$)	0.296	0.668	0.712
QuadLR (λ =10^{-2}, $\pi_T = 0.9$)	0.295	0.737	0.616
Raw Data - NO PCA			
QuadLR (λ =10^{-2}, $\pi_T = 0.5$)	0.277	0.765	0.729
QuadLR (λ =10^{-2}, $\pi_T = 0.1$)	0.276	0.721	0.736
QuadLR (λ =10^{-2}, $\pi_T = 0.9$)	0.296	0.811	0.678
Raw Data - PCA = 10			
QuadLR (λ =10^{-2}, $\pi_T = 0.5$)	0.284	0.726	0.684
QuadLR (λ =10^{-2}, $\pi_T = 0.1$)	0.287	0.738	0.665
QuadLR (λ =10^{-2}, $\pi_T = 0.9$)	0.296	0.806	0.645

Since the MVG Full-Cov and Quad LR, which are quadratic models, perform better than others, which are linear models, we may start to assume that quadratic models perform better than linear ones on this data set.

Also in this case Gaussianization does not improve the results with respect to the raw features results. Once again PCA brings no considerable benefit, so we can stop using it and will not be considered for the features analysis.

	$\tilde{\pi} = 0.5$	$\tilde{\pi} = 0.1$	$\tilde{\pi} = 0.9$
Raw Features - No PCA			
Quadratic LR (λ =10^{-2}, $\pi_T = 0.5$)	0.277	0.765	0.729
Gauss Features - No PCA			
MVG Full Cov	0.303	0.793	0.714

3.3 Support vector machine

Now we will focus on SVMs.

We will consider linear SVM, the polynomial quadratic kernel and the Radial Basis function kernel formulations and given the previous results we expect polynomial and RBF SVMs performing better.

For linear SVM, we need to tune the hyper-parameter C. Again, we use K-fold cross validation. We start with models that does not balance the two classes and then others balanced. We consider different value of C where $C_i = C_T$ for samples of the true class and $C_i = C_F$ for samples of false class and select

$$C_T = C\frac{\pi}{\pi_T^{emp}}; C_F = C\frac{\pi}{\pi_F^{emp}}$$

where π_T^{emp} and π_F^{emp} are the empiricals priors. (4)

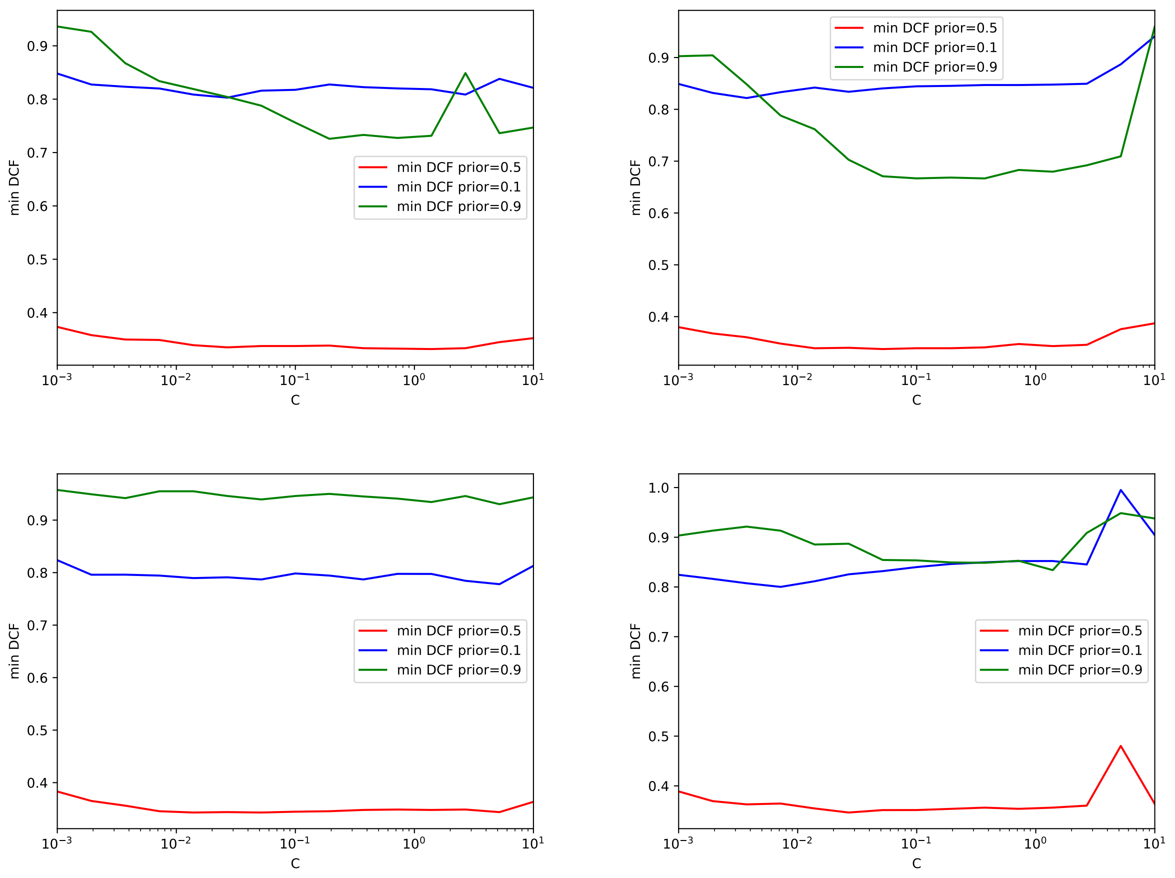


Figure 8: Linear SVM: left: Unbalanced; right: balanced; Top: Raw features; Bottom: Gaussianized features

For the hyper-parameter C we can take 10^{-1} since it is the biggest value that assures good performances. The graphs already show that re-balancing the model and Gaussianized pre-processing does not improve performances for the target application of this analysis but we will considered them the same for the following analysis just for a further check.

	$\tilde{\pi} = 0.5$	$\tilde{\pi} = 0.1$	$\tilde{\pi} = 0.9$
Raw Data			
Linear SVM (C = 10^{-1})	0.337	0.818	0.756
Linear SVM (C = $10^{-1}, \pi_T = 0.5$)	0.339	0.844	0.667
Linear SVM (C = $10^{-1}, \pi_T = 0.1$)	0.814	1.000	0.990
Linear SVM (C = $10^{-1}, \pi_T = 0.9$)	0.385	0.875	0.695
Gauss Data - NO PCA			
Linear SVM (C = 10^{-1})	0.345	0.798	0.946
Linear SVM (C = $10^{-1}, \pi_T = 0.5$)	0.351	0.840	0.853
Linear SVM (C = $10^{-1}, \pi_T = 0.1$)	0.617	0.964	0.996
Linear SVM (C = $10^{-1}, \pi_T = 0.9$)	0.395	0.947	0.684

Analyzing the linear SVM we confirm again the fact that linear models work worse than quadratic ones on this dataset.
Therefore, the analysis proceeds with the quadratic version of the SVM.

3.4 Quadratic Polynomial Kernel

The first non-linear SVM model used is the polynomial quadratic kernel, that is similar to the quadratic Logistic Regression model and we therefore expect similar results.

$$k(x1, x2) = (x_1^T x_2 + c)^d \tag{5}$$

The chose model has degree d = 2 (quadratic), whereas c is a hyper-parameter selected through cross-validation.

To add a regularized bias to the non- linear SVM version, a constant value K has been added to the kernel function.

$$k(x1, x2) = (x_1^T x_2 + c)^d + K^2 \tag{6}$$

In order to estimate the values of c and K that achieve the best results, different models have been trained by varying those parameters. In the following graphs are shown the minDCF obtained by the best models found through this analysis, through the variation of the value of the hyper-parameter C showing that we can select k = 1 and c = 1 and C=0.1
.

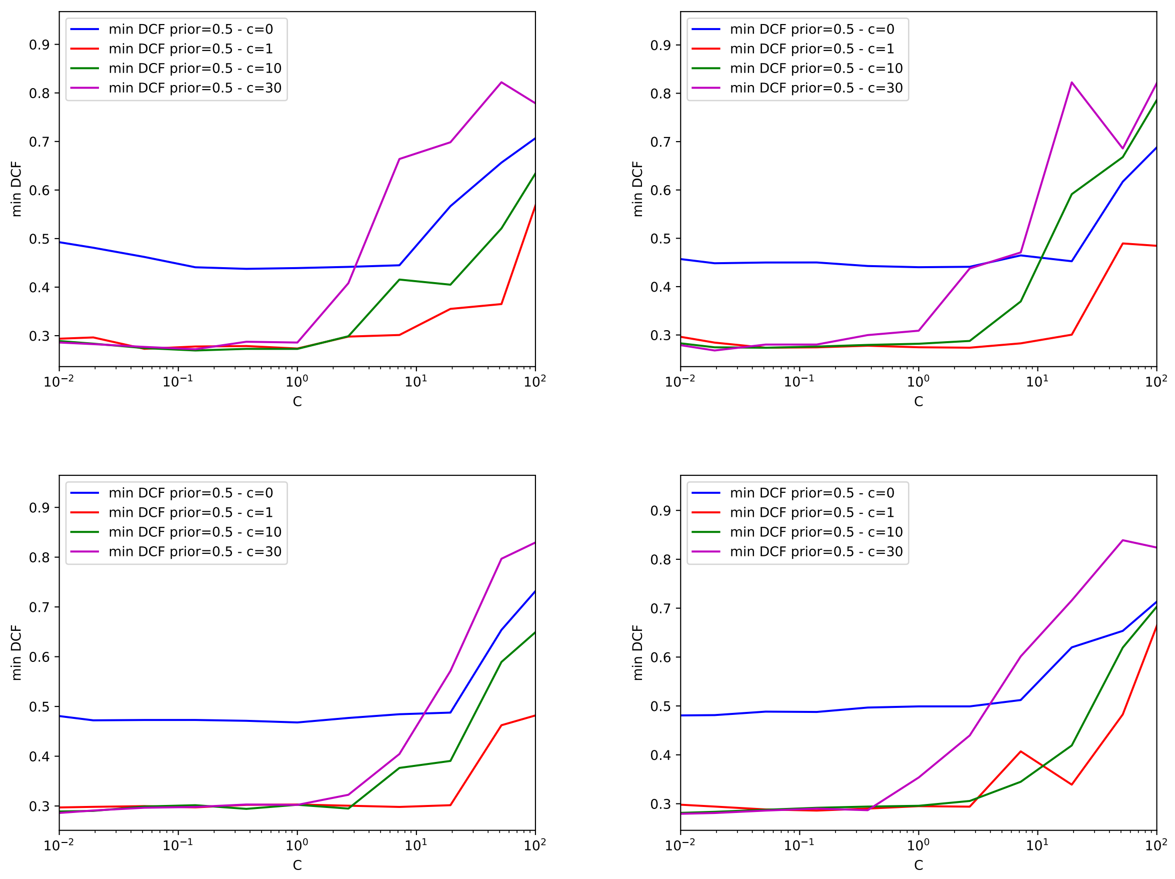


Figure 9: Quadratic SVM: left: Unbalanced; right: balanced; Top: Raw features; Bottom: Gaussianized features

	$\tilde{\pi} = 0.5$	$\tilde{\pi} = 0.1$	$\tilde{\pi} = 0.9$
Raw Data			
Quadratic SVM (C=0.1, c=1)	0.280	0.768	0.729
Quadratic SVM (C=0.1, c=1 $\pi_T = 0.5$)	0.273	0.805	0.680
Quadratic SVM (C=0.1, c=1 $\pi_T = 0.1$)	0.309	0.778	0.862
Quadratic SVM (C=0.1, c=1 $\pi_T = 0.9$)	0.325	0.949	0.657
Gauss Data			
Quadratic SVM (C=0.1, c=1)	0.296	0.669	0.702
Quadratic SVM (C=0.1, c=1 $\pi_T = 0.5$)	0.288	0.726	0.671
Quadratic SVM (C=0.1, c=1 $\pi_T = 0.1$)	0.321	0.695	0.896
Quadratic SVM (C=0.1, c=1 $\pi_T = 0.9$)	0.313	0.869	0.615

The results seem to be slightly better for the unbalanced version respect to the balanced one with $pi_T = 0.5$. Gaussianizing the features produce a slight deterioration.

3.5 RBF Kernel

Another kernel function that can be used for non- linear SVM models is the Radial Basis Function.

$$k(x1, x2) = e^{-\gamma \|x_1 - x_2\|^2} \tag{7}$$

γ is the width of the kernel: small γ correspond to a wide kernel, while large γ to a narrow kernel.
Using k-fold we estimate the optimal values of γ and C.

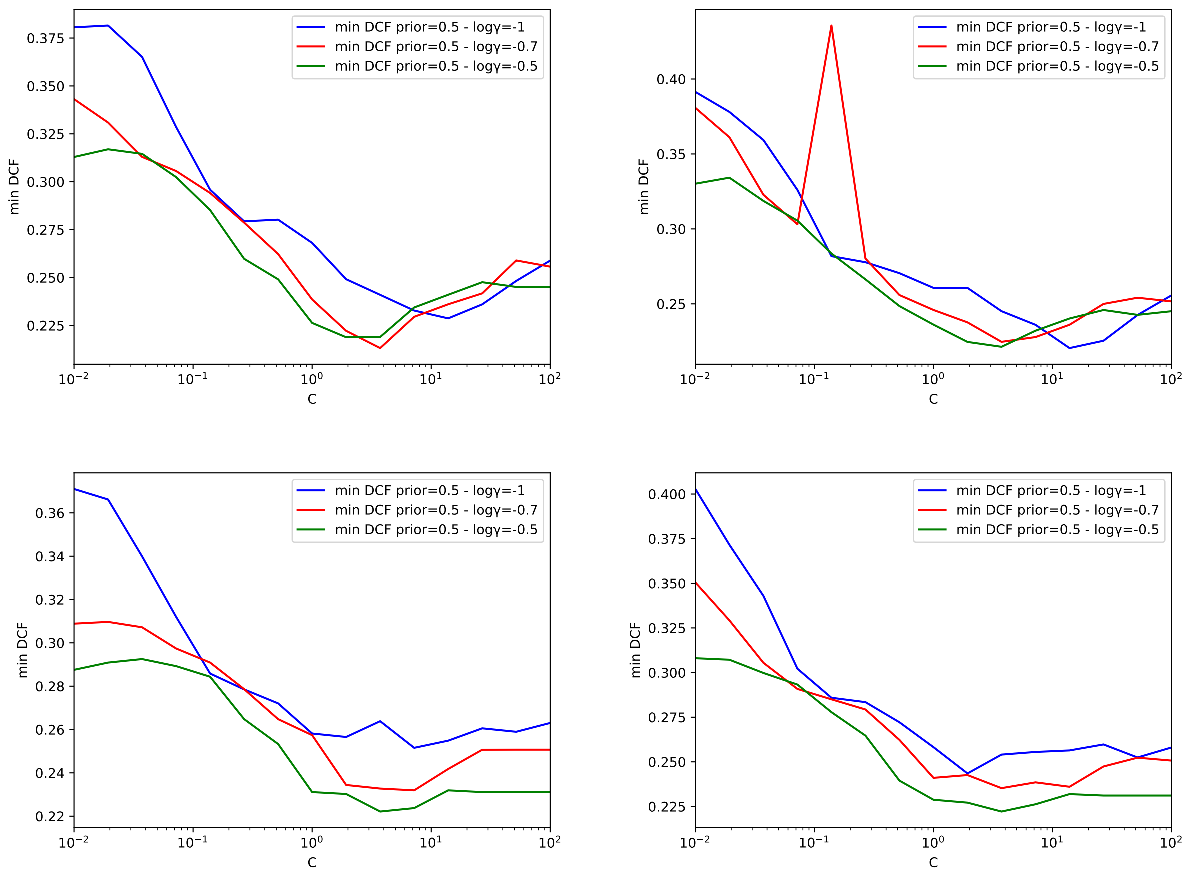


Figure 10: RBF SVM: left: Unbalanced; right: balanced; Top: Raw features; Bottom: Gaussianized features

Choosing $\gamma = 10^{-0.5}$ and $C = 10^{0.1}$ we can train the model with and without re-balancing.

	$\tilde{\pi} = 0.5$	$\tilde{\pi} = 0.1$	$\tilde{\pi} = 0.9$
Raw Data			
RBF SVM (C=$10^{0.1}$, $\gamma = 10^{-0.5}$)	0.221	0.593	0.774
RBF SVM (C=$10^{0.1}$, $\gamma = 10^{-0.5}$ $\pi_T = 0.5$)	0.235	0.585	0.611
RBF SVM (C=$10^{0.1}$, $\gamma = 10^{-0.5}$ $\pi_T = 0.1$)	0.241	0.615	0.927
RBF SVM (C=$10^{0.1}$, $\gamma = 10^{-0.5}$ $\pi_T = 0.9$)	0.254	0.767	0.567
Gauss Data			
RBF SVM (C=$10^{0.1}$, $\gamma = 10^{-0.5}$)	0.231	0.549	0.650
RBF SVM (C=$10^{0.1}$, $\gamma = 10^{-0.5}$ $\pi_T = 0.5$)	0.232	0.528	0.600
RBF SVM (C=$10^{0.1}$, $\gamma = 10^{-0.5}$ $\pi_T = 0.1$)	0.253	0.594	0.809
RBF SVM (C=$10^{0.1}$, $\gamma = 10^{-0.5}$ $\pi_T = 0.9$)	0.237	0.680	0.613

RBF is the best model up to now and obtain best results without re-balancing. Gaussianization seem to produce slightly better results.

	$\tilde{\pi} = 0.5$	$\tilde{\pi} = 0.1$	$\tilde{\pi} = 0.9$
Raw Features - No PCA			
Quadratic LR ($\lambda = 0, \pi_T = 0.5$)	0.277	0.765	0.729
Quadratic SVM (C=1, c=1)	0.274	0.777	0.709
RBF SVM (C=$10^{0.1}$, $\gamma = 10^{-0.5}$)	0.221	0.593	0.774
Gauss Features - No PCA			
MVG Full Cov	0.303	0.793	0.714

3.6 GAUSSIAN MIXTURE MODELS

The last model we analys is the Gaussian Mixture Model.
GMMs can approximate generic distributions, so we expect to obtain better results than with the Gaussian model.
We will consider the full covariance model, the diagonal model and the one with covariance tying.
For the mode with tied covariance, tying takes place at class level, so different covariance matrices.

Again we will use the 5-folds approach to choose the components number and compare different models.

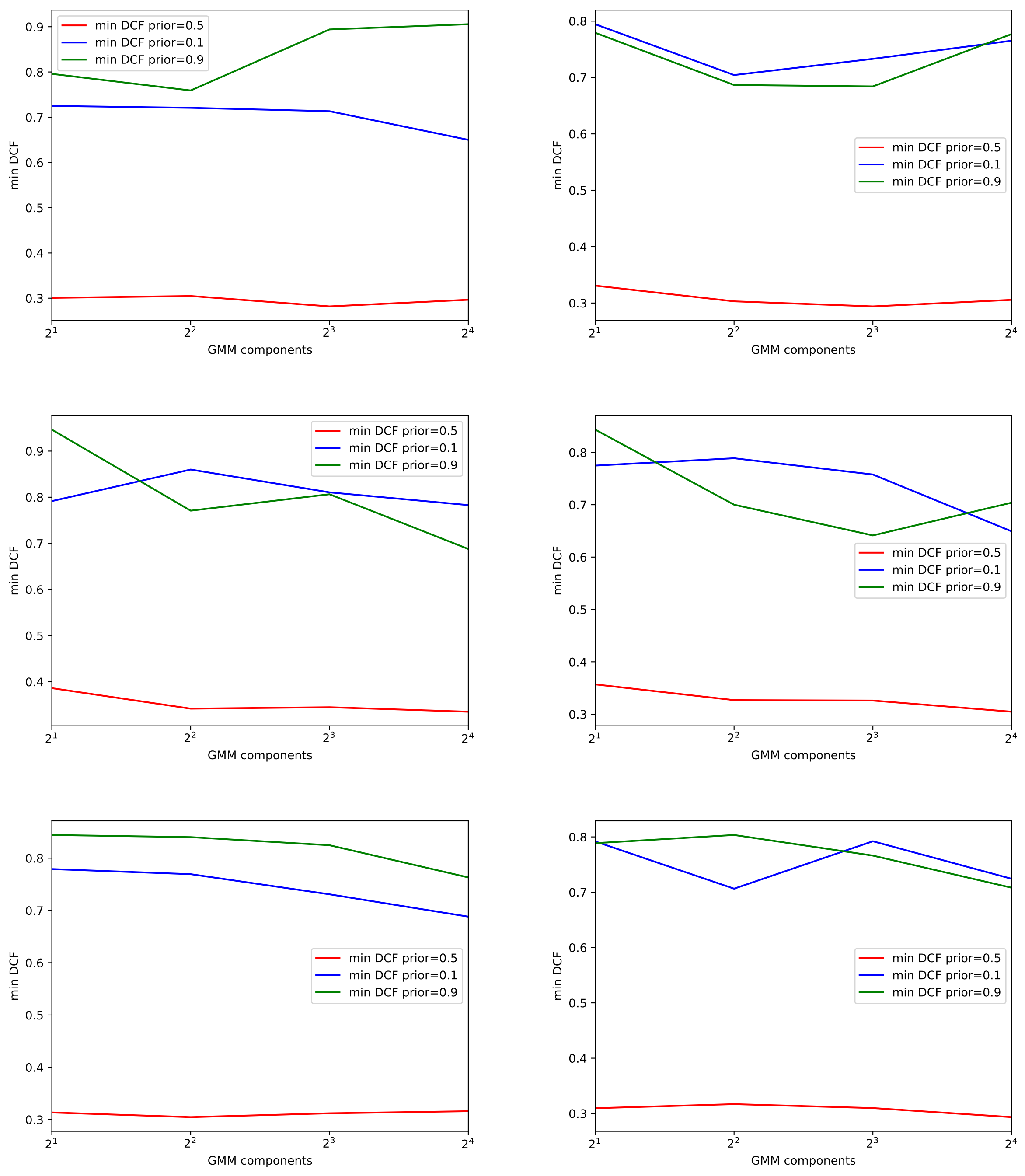


Figure 11: GMM SVM: left: Raw features; right: Gaussianized features; Top: GMM FUll-Cov; Middle: GMM Diag Cov; Bottom: GMM Tied-Cov

The models trained with Gaussianized and Raw features are pretty similar.

	$\tilde{\pi} = 0.5$	$\tilde{\pi} = 0.1$	$\tilde{\pi} = 0.9$
Raw Data			
GMM FUll Cov (n = 2)	0.301	0.725	0.796
GMM FUll Cov (n = 4)	0.305	0.721	0.759
GMM FUll Cov (n = 8)	0.282	0.713	0.894
GMM FUll Cov (n = 16)	0.297	0.650	0.905
GMM FUll Cov (n = 32)	0.286	0.669	0.905
GMM Diag Cov (n=2)	0.386	0.792	0.946
GMM Diag Cov (n=4)	0.342	0.856	0.771
GMM Diag Cov (n=8)	0.345	0.811	0.806
GMM Diag Cov (n=16)	0.335	0.705	0.688
GMM Diag Cov (n=32)	0.311	0.509	0.732
GMM Tied (n = 2)	0.314	0.779	0.844
GMM Tied (n = 4)	0.305	0.769	0.840
GMM Tied (n = 8)	0.312	0.731	0.827
GMM Tied (n = 16)	0.316	0.688	0.763
GMM Tied (n = 32)	0.297	0.745	0.826
Gauss Data			
GMM FUll Cov (n = 2)	0.331	0.794	0.779
GMM FUll Cov (n = 4)	0.303	0.704	0.687
GMM FUll Cov (n = 8)	0.294	0.733	0.684
GMM FUll Cov (n = 16)	0.305	0.765	0.777
GMM FUll Cov (n =)	0.343	0.739	0.753
GMM Diag Cov (n=2)	0.357	0.774	0.843
GMM Diag Cov (n=4)	0.327	0.789	0.700
GMM Diag Cov (n=8)	0.326	0.758	0.641
GMM Diag Cov (n=16)	0.305	0.649	0.704
GMM Diag Cov (n=32)	0.299	0.652	0.748
GMM Tied (n = 2)	0.310	0.771	0.789
GMM Tied (n = 4)	0.317	0.706	0.803
GMM Tied (n = 8)	0.310	0.792	0.766
GMM Tied (n = 16)	0.293	0.724	0.708
GMM Tied (n = 32)	0.291	0.674	0.704

The results are consistent with the ones of MVG models. The best model is Full Covariance with 8 components trained with raw features. Furthermore, in this model the Gaussianization shows slightly worse performances. This can be explained since Gaussianization reduces the dynamic range of samples far from the data mean and thus reduces the separability of some clusters.

4. Score Calibration

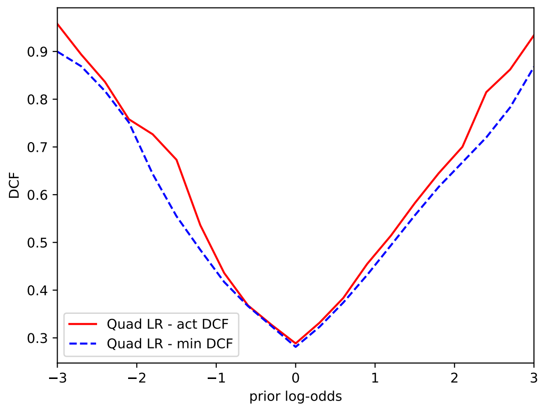
Up to now we have only considered min DCF, misuring the cost we would pay if we made optimal decisions for the validation set using the scores of the rocognizer.
However, the cost we actually pay depends on the goodness of the threshold we use to

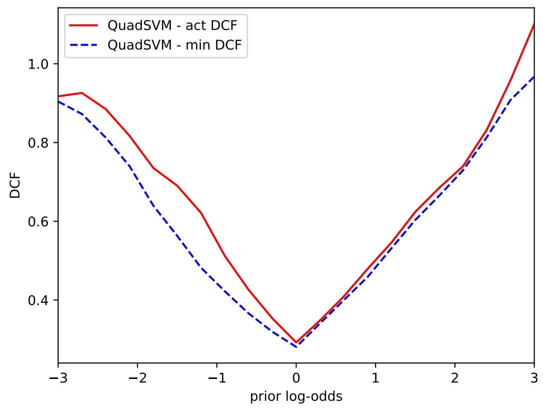
perform class assignment.
We therefore turn our attention to actual DCFs.
If scores are well calibrated, the optimal threshold that optimizes the Bayes risk is the theoretical threshold $t = -log\frac{\tilde{\pi}}{1-\tilde{\pi}}$.

So, at this point we evaluate the actual DCF, only on the best models, to measure how good the models would behave if we were using the theoretical threshold for each application, which means we are assuming that scores are already well-calibrated:

	$\tilde{\pi} = 0.5$	$\tilde{\pi} = 0.1$	$\tilde{\pi} = 0.9$
	min DCF - act DCF	min DCF - act DCF	min DCF - act DCF
Raw Data			
Quadratic LR ($\lambda = 0, \pi_T = 0.5$)	0.281 - 0.288	0.771 - 0.785	0.687 - 0.736
Quadratic SVM (C=1, c=1)	0.281 - 0.292	0.778 - 0.845	0.753 - 0.761
GMM FUll Cov (n = 8)	0.282 - 0.290	0.713 - 0.865	0.894 - 1.029
RBF SVM (C=$10^{0.1}$,$\gamma = 10^{-0.5}$)	0.222 - 0.259	0.593 - 0.991	0.774 - 0.995

We can observe that all the models except RBF are almost calibrated for the main application and there is a loss that grows for the unbalanced applications. This analysis can also be verified through Bayes error plots, which show the DCFs for different applications.





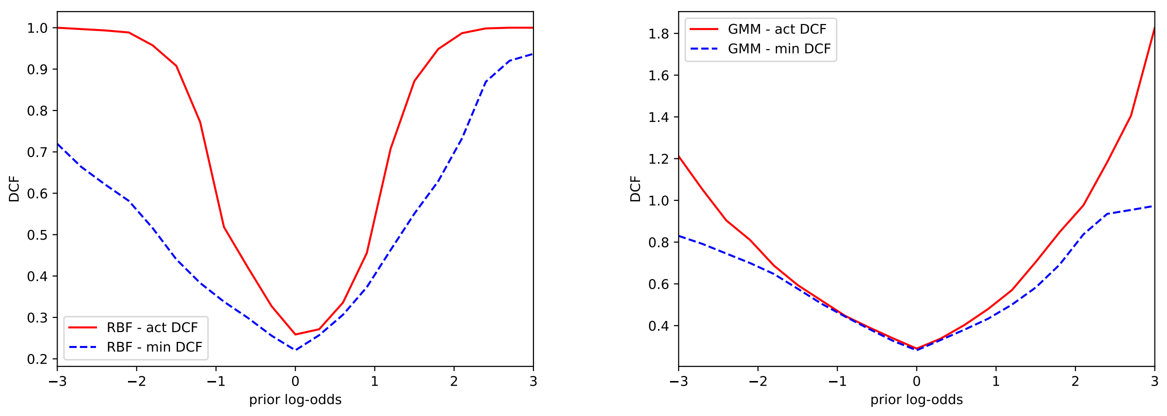


Figure 12: Bayes error plots

However, we can try employing score calibration on all the models. A general approach consists in computing a transformation function f that maps the scores s of each classifier to well-calibrated scores $s_{cal} = f(s)$
We assume that function f is linear in s:

$$f(s) = \alpha s + \beta$$

Since $f(s)$ should output well-calibrated scores, it can be interpreted as the log-likelihood ratio for the two class hypotheses:

$$f(s) = log \frac{f_{S|C}(s|H_T)}{f_{S|C}(s|H_F)} = \alpha s + \beta$$

so the class posterior probability for a prior $\tilde{\pi}$ can be written as

$$f(s) = log \frac{f_{S|C}(s|H_T)}{f_{S|C}(s|H_F)} = \alpha s + \beta$$

We can interpret the scores s as features so that this expression will be similar to the log posterior ratio of the Logistic Regression model. If we rewrite:

$$\beta^{'} = \beta + log \frac{\tilde{\pi}}{1 - \tilde{\pi}}$$

then we have exactly the same model and we can use the prior-weighted Logistic Regression model that we already discussed to learn the model parameters α, $\beta^{'}$ over To recover the calibrated scores f(s) we need to compute:

$$f(s) = \alpha s + \beta - log \frac{\tilde{\pi}}{1 - \tilde{\pi}}$$

We are specifying a certain prior $\tilde{\pi}$ (0.5 in our case), so we are effectively optimizing the calibration for this specific application, but this approach will provide good calibration for different applications anyway. This time we start from Bayes error plots so that we can try different values for the hyper-parameter and see how they affect score calibration.

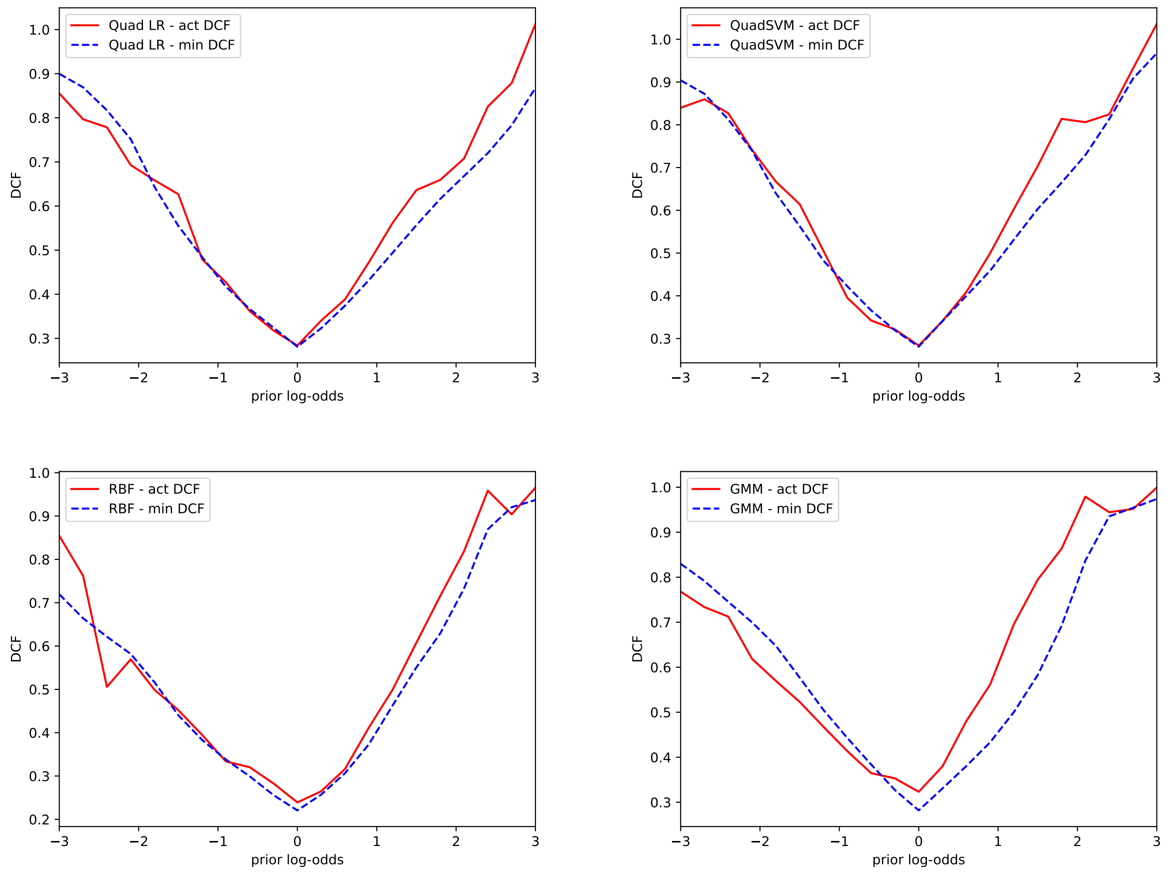


Figure 13: Calibrated scores

5. Experimental results

In this section, we analyse the different models performances on the test set in terms of min DCFs. We can expect the new minDCF measures to be slightly different than the previous ones, since the distribution of the data is not exactly the same in the training and test set. If the distribution is similar enough and our assumptions were correct, though, the difference should be very small. The results are collected in the following table.

	$\tilde{\pi} = 0.5$	$\tilde{\pi} = 0.1$	$\tilde{\pi} = 0.9$
Raw Features			
MVG Full	0.337	0.656	0.700
MVG Diag	0.367	0.731	0.864
MVG Tied	0.315	0.681	0.0.705
MVG Tied-Diag	0.369	0.738	0.930
LR (λ =10^{-2}, $\pi_T = 0.5$)	0.331	0.683	0.654
Quadratic LR ($\lambda = 0, \pi_T = 0.5$)	0.257	0.656	0.574
Quadratic SVM (C=1, c=1)	0.278	0.617	0.608
RBF SVM (C=$10^{0.1}$,$\gamma = 10^{-0.5}$ pi$_T = 0.5$)	0.251	0.696	0.681
GMM FUll Cov (n = 8)	0.306	0.785	0.851
Gauss Features			
MVG Full	0.328	0.683	0.795
MVG Diag	0.384	0.756	0.931
MVG Tied	0.325	0.694	0.809
MVG Tied-Diag	0.387	0.823	0.927
LR (λ =10^{-2}, $\pi_T = 0.5$)	0.337	0.694	0.710
Quadratic LR ($\lambda = 0, \pi_T = 0.5$)	0.284	0.668	0.543
Quadratic SVM (C=1, c=1)	0.288	0.654	0.598
RBF SVM (C=$10^{0.1}$, $\gamma = 10^{-0.5}$)	0.269	0.696	0.681
GMM FUll Cov (n = 8)	0.333	0.722	0.740

The results are consistent with the ones obtained in the previous analysis and the best models for our target application remain RBF SVM, followed by Quadratic Logistic Regression and Quadratic SVM.

After that we can also see the effects of hyper-parameters on performance on the test set and make a comparison with those estimated during the evaluation phase.

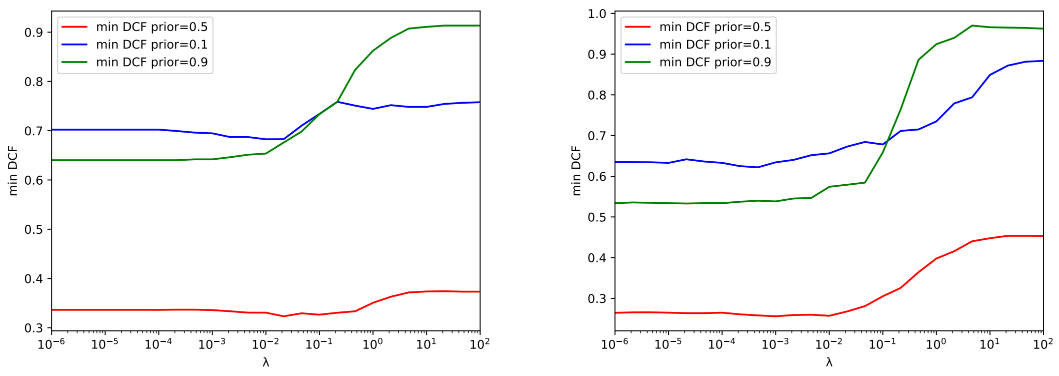
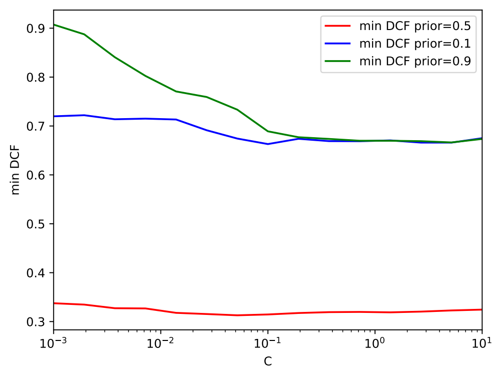


Figure 14: min DCF for different values of λ on the evaluation set. Left: Linear LR; right: Quadratic LR



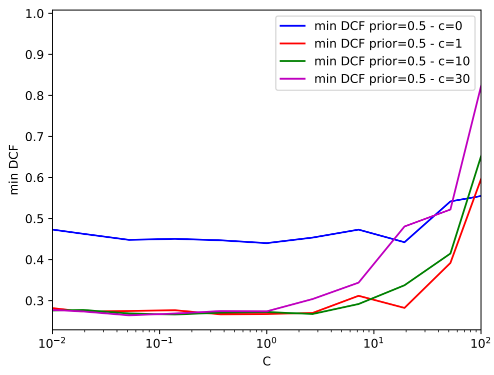
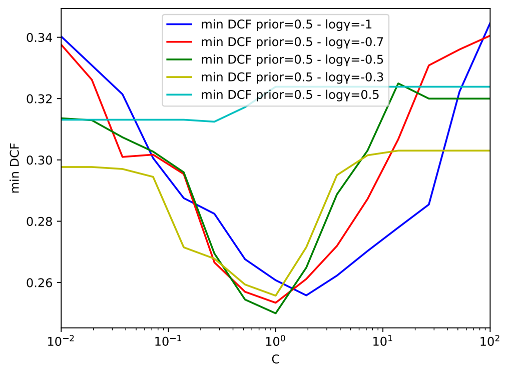


Figure 15: min DCFs on the evaluation set. Left: Linear SVM; right: Quadratic SVM



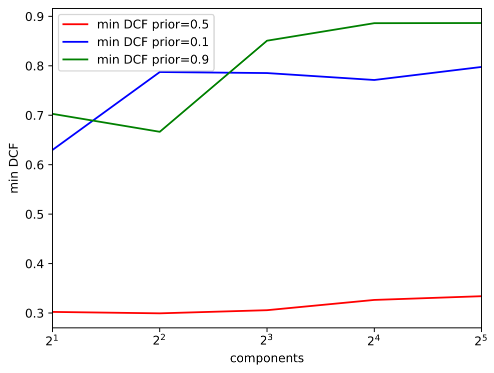


Figure 16: min DCFs on the evaluation set. Left: RBF SVM; right: GMM FUll Cov

Take a look of all the graphs and compare it with the ones computed during the validation phase we can notice the evaluation results are in line with expectations and for all the models the curves for the validation and evaluation set have the same trend, showing that our choices was indeed effective. In conclusion We can compare our classifiers through a ROC plot. The best classifiers are the ones with the highest AUC (Area Under Curve); in our case the curve is pretty much the same for all the classifiers. Moreover, Their curves show a high slope in the left part which means the models correctly classify high quality wines while keeping the low quality ones to a minimum.

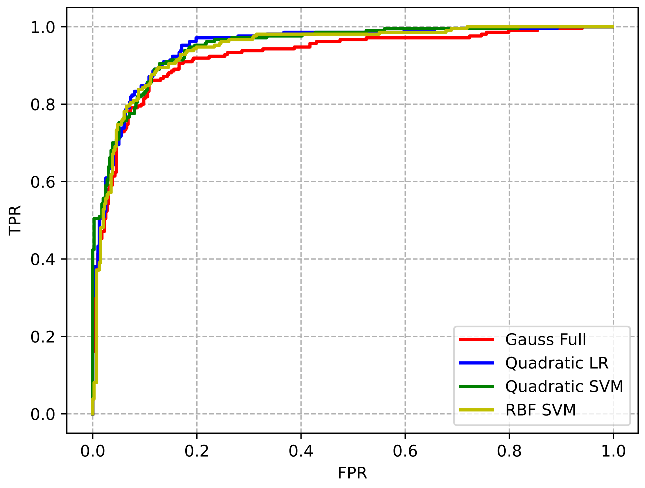


Figure 17: ROC plot for models comparison

6. Conclusion

To conclude we can say that the dataset works best on quadratic algorithms with respect to linear models. We are able to achieve a DCF of ≈ 0.2 for the target application $\pi = 0.5$, altough also for the other two considered applications with $\pi = 0.1$ and $\pi = 0.9$ results are acceptable, with DCFs of ≈ 0.7 and ≈ 0.8, respectively. So, the choices made on our training set proved to be effective also for the evaluation set.