Machine Learning and Pattern Recognition
Fingerprint Spoofing

Federica Amato s310275
Marco Colangelo s309798

1 Introduction

1.1 Abstract

This report aims to examine a dataset that contains various low-level images of real (defined as Authentic) and fake (defined as Spoofed) fingerprints using different Machine Learning models. First, we will explore the structure of the dataset and try to understand how it is distributed. Then, we will evaluate various classifiers and try to find the best system that can accurately classify our samples with the lowest cost.

1.2 First considerations about the problem

The dataset is composed of samples that represent fingerprint images through low-dimensional representations called embeddings. Each fingerprint is represented by a 10-dimensional vector of continuous real numbers, which is obtained by mapping the images to a lower-dimensional space. Real fingerprints are labeled as 1, while spoofed fingerprints are labeled as 0. The individual components of the embeddings do not have any physical interpretation. Spoofed fingerprint samples can belong to one of six different sub-classes, each representing a different spoofing technique, but the specific technique used is not provided. The target application assumes that the prior probabilities for the two classes are equal, but this does not hold for the misclassification costs. The training set contains 2325 samples and the test set contains 7704 samples, with the fake fingerprint class being significantly overrepresented.

1.3 Features analysis

Here is a series of plot about the several features distributions:

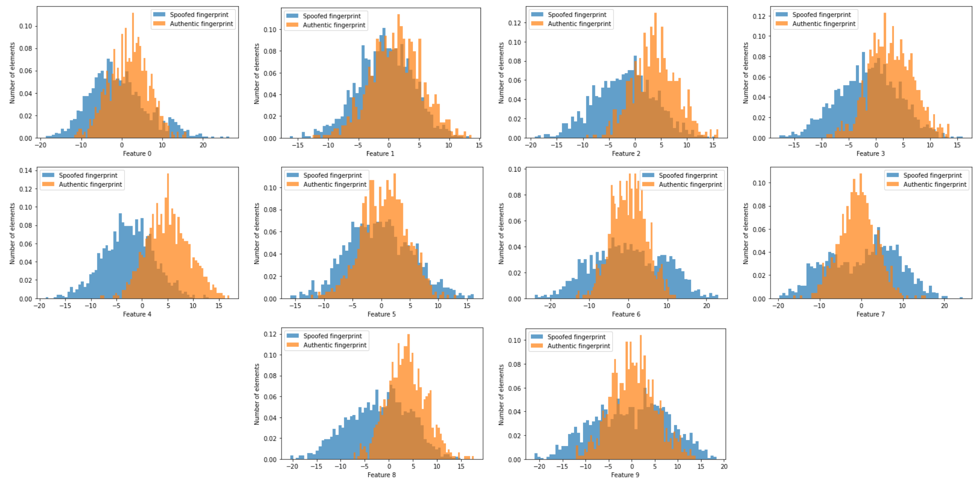


Figure 1: Histograms of the features

You can see how some of these features can easily be described with a Gaussian distribution. Feature 1 (starting to count from 0) is the most evident case. In any case, it is evident that the shape of the distribution to describe the features of the Authentic class is much more similar to a Gaussian form while this is not the case for Spoofed class plots. This can be partly justified by the fact that the samples defined as Spoofed are actually composed of samples obtained with different spoofing techniques. Ergo, the class in question is constructed by putting together elements of different sub-classes. Feature 4 looks like the one that most easily divides the elements of the two classes. Here are some of the most representative cross-feature plots. It's evident how the Authentic class is well described by a Gaussian form and how the position of the Spoofed samples is organized over several clusters (that correspond to several sub-classes):

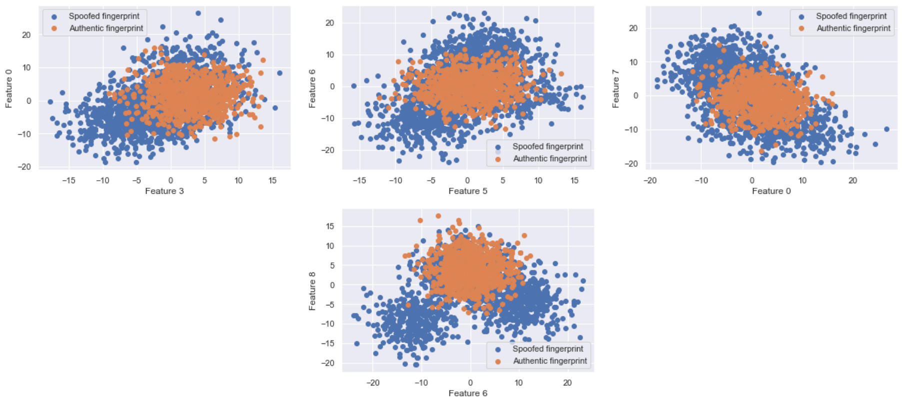


Figure 2: Cross features plot

A transformation that we can use is the Linear Discriminant Analysis (LDA), this preprocessing step allows us to understand if the features are linearly discriminable, so this means that if a linear and/or Gaussian model may perform better than no-gaussian and/or no-linear model. LDA founds C-1 directions over to plot our features where C is the number of classes (in our case we have only 1 dimension because we trait a binary classification problem). Looking at the graph we see how it is possible to use linear separation methods for the two classes but the separation would not be without errors.

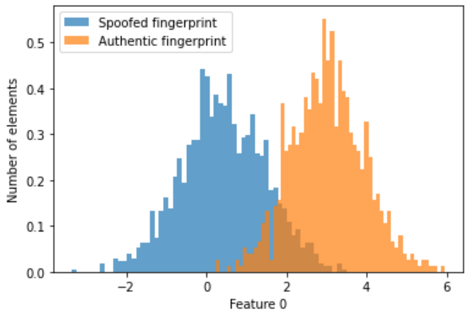


Figure 3: LDA application with m = 1

We focus now on the correlation between the features using Pearson correlations plots.

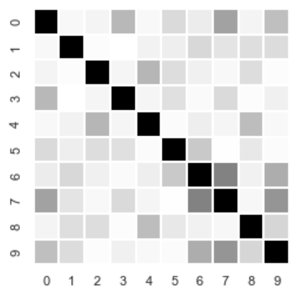


Figure 4: Whole dataset

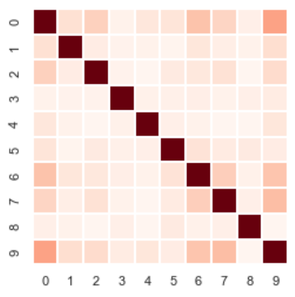


Figure 5: Authentic class

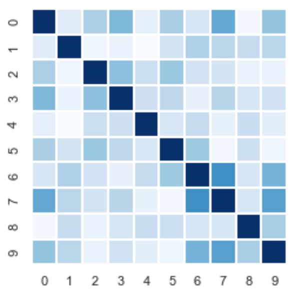


Figure 6: Spoofed class

Looking at the heatmaps, it can be deduced that almost all features are poorly correlated, especially if we look at the Authentic class plot. However, in the plot bound to the whole dataset and in the plot for the Spoofed class we can see some strictly correlated features. This may be reconducted to the several clusters of the Spoofed class. For example, features 6,7 and 9 seem to be quite correlated. this suggests that we may have benefit if we map data from 10-dimensional to 7-dimensional or 6-dimensional space in order to reduce the number of parameters to estimate for a model.

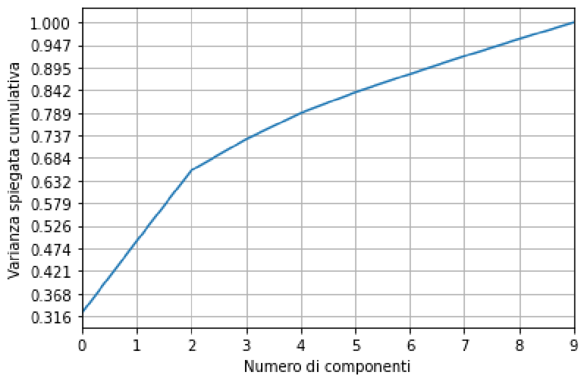


Figure 7: Cumulative variance over PCA dimensions

From the graph we see how with pca 6 we store up to 89.5% of the information, so we will try our models applying PCA to reduce up to 6 dimensions

2 Classification and Validation

2.1 Introduction

This report is based on the results we obtained from the following list of classification models:

- Generative models - Gaussian classifier:
  - Multivariate Gaussian classifier (MVG)
  - MVG + Diagonal Covariance
  - MVG + Tied Covariance
  - MVG + Diagonal and Tied Covariance
- Logistic Regression:
  - Prior weighted Logistic Regression
  - Quadratic Logistic Regression
- SVM - Support Vector Machine:

  - Linear SVM
  - SVM with Polynomial kernel function with degree=2
  - SVM with RBF kernel function

- GMM - Gaussian Mixture Model:
  - Gaussian Mixture Model
  - GMM + Diagonal Covariance
  - GMM + Tied Covariance
  - GMM + Diagonal and Tied Covariance

For the validation phase, some explanations are needed:

- To determine the most promising model and evaluate the impact of using PCA, we used K-Fold cross validation. All of the results presented here were obtained using K-Fold Validation with K = 5.
- We reported in this phase minDCF. The actDCF are considered in a second phase
- Our models are trained with the effective prior $\pi_T = \frac{1}{11}$ as a project choice. In fact: $(\pi, Cfn, Cfp) = (0.5, 1, 10) = (\frac{1}{11}, 1, 1)$

2.2 Multivariate Gaussian Classifier

We are now ready to begin our examination of machine learning models by looking at Gaussian classifiers such as the full covariance (MVG), the diagonal covariance (Naive Bayes, which assumes that assumes that covariance matrices Σ are diagonal matrices), as well as their tied assumption, which means that each class has its own mean μ_c but the covariance matrix Σ is computed over the whole dataset.
Gaussian Classifiers works under the assumption that data follows a gaussian distribution:

$$X|C = c \sim N(\mu_c, \Sigma_c)$$

2.2.1 Expectation

Given that our histograms exhibit a similar Gaussian distribution, we anticipate that the MVG will perform well on our data. Furthermore, based on our previous discussion of Pearson correlation, Naive Bayes should not perform poorly, however we expect it to perform slightly worse than the MVG case since from the Pearson correlation we notice a correlation between features 6, 7 and 9 (counting from 0). Our histograms also reveal that data for each class is spread differently, resulting in distinct covariance matrices. As a result, we believe that MVG + Tied Covariance may not perform as well as the others.

2.2.2 Results

	MVG	MVG + Diag	MVG + Tied	MVG + Diag + Tied
minDCF (no PCA)	**0.331**	0.472	0.486	0.551
minDCF (PCA 6)	**0.336**	0.360	0.483	0.549
minDCF (PCA 7)	**0.341**	0.361	0.484	0.541
minDCF (PCA 8)	**0.333**	0.360	0.485	0.544
minDCF (PCA 9)	**0.330**	0.369	0.492	0.543

Table 1: Gaussian Classifiers (minDCF).

As anticipated, the MVG model yielded the best results, indicating that quadratic models perform better on our dataset, the Naive model performs slightly worse, while for the MVG + Diag + Tied case we have a worsening of performance. Looking at the results obtained, we are able to see that PCA helps us greatly improve the performance of the model. However, not having substantial differences between the various minDCFs, we opted to verify with other models which could be the candidate PCA for our analysis.

2.3 Logistic Regression

We shift now our attention to Logistic Regression models. We tried the balanced version of logistic regression and, as evidence that the classes are unbalanced (as reported in the project trace), we noticed that the model does not perform well. Having said that, since classes are unbalanced, we employ a prior-weighted regularized version of the objective function:

$$J(\omega, b) = \frac{\lambda}{2}||\omega||^2 + \frac{\pi_t}{n_T}\sum_{i=1|c_i=1}^{n_T} \log(1 + e^{-z_i s_i}) + \frac{(1-\pi_T)}{n_F}\sum_{i=1|c_i=0}^{n} \log(1 + e^{-z_i s_i})$$

with $s_i = (\omega^T x_i + b)$, where (ω, b) are the model parameters, $\frac{\lambda}{2}||\omega||^2$ a regularization term that helps obtaining a ω with lower norm (reducing the risk of over-fitting the training data) and λ, which is an hyperparameter called regularization coefficient:

- $\lambda >> 0$: poor separation of classes and a small $||\omega||$.
- $\lambda \approx 0$: good separation of classes but poor generalization on unseen data.

Let's proceed with the choice of λ by plotting minDCF graphs for a range of values.

2.3.1 Expectations

Given that the best results for Gaussian classifiers were achieved using the quadratic MVG model, we can anticipate that quadratic Logistic Regression will outperform its linear counterpart, as our data appears to be better separated by quadratic rules.

2.3.2 Results

We started our analysis by plotting minDCF graphs for a range of values of λ ,using both Z-Norm and no Z-Norm versions. As we said before, we tried different PCA and even different value of π_T to see how results change.

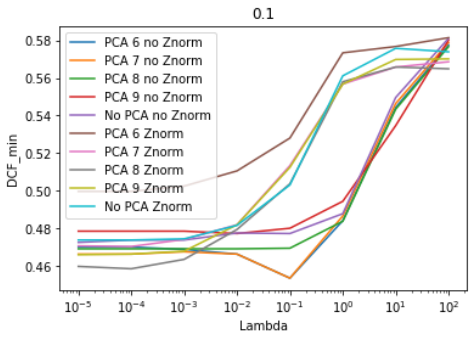


Figure 8:
Logistic Regression with $\pi_T = 0.1$

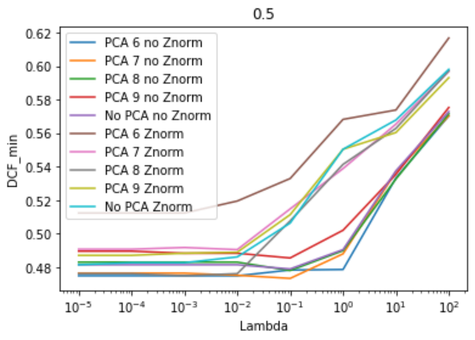


Figure 9:
Logistic Regression with $\pi_T = 0.5$

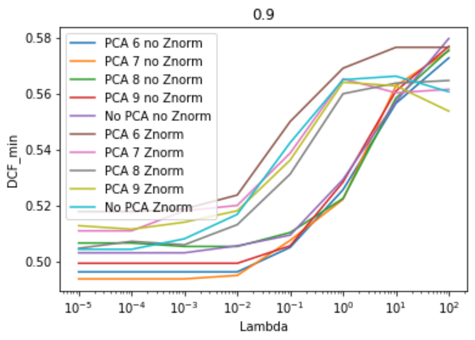


Figure 10:
Logistic Regression with $\pi_T = 0.9$

As we see from these plots, the minimum is reached by setting λ=0.1, so we will consider it as the best value for hyperparameter λ. We can notice that PCA with 6 components helped us reduce the minDCF value, while Z-normalization did not improve the performance. In fact, even when we combined PCA and Z-normalization, we got worse results than with PCA alone.

Below,we reported the values obtained.

λ	Z-norm	no Z-norm
10^{-5}	**0.474**	**0.472**
10^{-4}	0.474	0.474
10^{-3}	0.474	0.474
10^{-2}	0.481	0.477
10^{-1}	0.503	0.477
1	0.561	0.487
10^{1}	0.576	0.550
10^{2}	0.574	0.581

Table 2: Logistic Regression (minDCF) no PCA $\pi_T = 0.1$.

λ	Z-norm	no Z-norm
10^{-5}	**0.500**	0.470
10^{-4}	**0.500**	0.470
10^{-3}	0.502	0.469
10^{-2}	0.510	**0.466**
10^{-1}	0.528	0.453
1	0.573	0.484
10^{1}	0.577	0.544
10^{2}	0.581	0.577

Table 3: Logistic Regression (minDCF) PCA 6 $\pi_T = 0.1$.

We can notice that for values of λ too high the model doesn't perform correctly, infact we know that a high value of λ causes the model to over-generalized and become too simple, leading to subpar classification performance. To strike a balance between overfitting and underfitting, a suitable value of λ could be chosen and can be in the order of 10^{-2}

Finally, we can observe that classifier with $\pi_T = 0.1$ performs better than the others with $\pi_T = 0.5$ and $\pi_T = 0.9$, this is due to the imbalance towards samples of class 0 of the fingerprint dataset. As proof, we present here the table for PCA 6, that we think is our best candidate, for $\pi_T = 0.5$ and $\pi_T = 0.9$

λ	Z-norm	no Z-norm
10^{-5}	**0.512**	**0.475**
10^{-4}	**0.512**	**0.475**
10^{-3}	**0.512**	**0.475**
10^{-2}	0.520	**0.475**
10^{-1}	0.533	0.478
1	0.568	0.479
10^{1}	0.574	0.533
10^{2}	0.617	0.570

Table 4: Logistic Regression (minDCF) PCA 6 $\pi_T = 0.5$.

λ	Z-norm	no Z-norm
10^{-5}	**0.500**	0.470
10^{-4}	**0.500**	0.470
10^{-3}	0.502	0.469
10^{-2}	0.510	**0.466**
10^{-1}	0.528	0.453
1	0.573	0.484
10^{1}	0.577	0.544
10^{2}	0.581	0.577

Table 5: Logistic Regression (minDCF) PCA 6 $\pi_T = 0.9$.

2.3.3 Quadratic Logistic Regression

Quadratic logistic regression is a type of regression model that allows us to estimate the probabilities of belonging to a binary class as a function of continuous or categorical explanatory variables, including quadratic terms of continuous variables. This allows to capture any non-linear effects of explanatory variables on the response variable. Another aspect to consider in quadratic logistic regression is the transformation of explanatory variables. We can use the

$$\phi(x) = \begin{pmatrix} \mathrm{vec}(xx^T) \\ x \end{pmatrix} \tag{1}$$

function to create an expanded feature space that includes the outer product of the x vector with itself and the original x vector. In this way, we can train the logistic regression model using the $\phi(x)$ feature vectors instead of x. This allows us to calculate linear separation rules for $\phi(x)$, which corresponds to estimating quadratic separation surfaces in the original space.
Until now the best results were obtained with MVG (which is a quadratic model),so we expect that quadratic LR perform better than linear version, because our data seems to be better separated by quadratic rules.

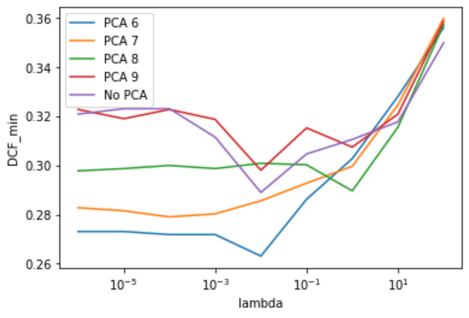


Figure 11: $\pi_T = 0.1$

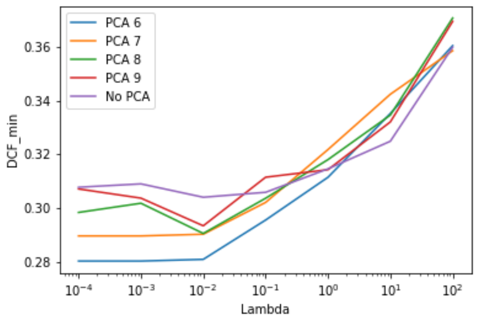


Figure 12: $\pi_T = 0.33$

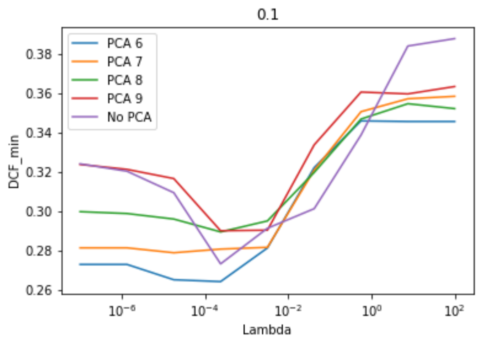


Figure 13: $\pi_T = 0.1$ + Z-Norm

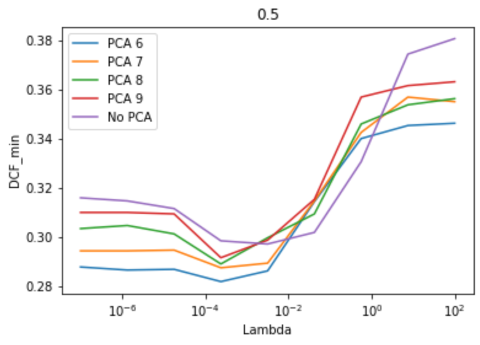


Figure 14: $\pi_T = 0.33$ + Z-Norm

Due to the lack of performance improvement with π_T=0.5 and π_T=0.9 seen with the linear version, we tried now with π_T=0.33 because similar to the ratio of the number of Authentic samples over the Spoofed ones, but also in this case we did not notice any improvement compared with π_T=0.1. We report just the results for π_T=0.1 and π_T=0.33. Z-Norm data are not put in the tables because we considered them as not relevant in this case. We report just one comparison for π_T=0.1 between the PCA=6 configuration and PCA=6+Z-Norm configuration because of them good (and similar) performances. We report the most relevant configurations below:

	π_T=0.1	π_T=0.33
$\lambda = 10^{-5}$	0.273	**0.280**
$\lambda = 10^{-4}$	0.272	**0.280**
$\lambda = 10^{-3}$	0.272	**0.280**
$\lambda = 10^{-2}$	**0.263**	0.281
$\lambda = 10^{-1}$	0.286	0.295
$\lambda = 1$	0.303	0.311
$\lambda = 10^{-1}$	0.328	0.335
$\lambda = 10^{-2}$	0.356	0.360

Table 6: PCA = 6

	π_T=0.1	π_T=0.33
$\lambda = 10^{-5}$	0.323	0.308
$\lambda = 10^{-4}$	0.323	0.309
$\lambda = 10^{-3}$	0.311	**0.305**
$\lambda = 10^{-2}$	**0.289**	0.306
$\lambda = 10^{-1}$	0.305	0.314
$\lambda = 1$	0.310	0.325
$\lambda = 10^{-1}$	0.318	0.325
$\lambda = 10^{-2}$	0.350	0.360

Table 7: PCA = None

	No Z-Norm	Z-Norm
$\lambda = 10^{-5}$	0.273	0.265
$\lambda = 10^{-4}$	0.272	**0.264**
$\lambda = 10^{-3}$	0.272	0.281
$\lambda = 10^{-2}$	**0.263**	0.322
$\lambda = 10^{-1}$	0.286	0.346
$\lambda = 1$	0.303	0.346
$\lambda = 10^{-1}$	0.328	0.346
$\lambda = 10^{-2}$	0.356	0.346

Table 8: PCA=6 with π_T=0.1 - Comparison between Z-Norm and No Z-Norm

Applying PCA with m=6 and without Z-Norm is still the best solution. The results are slightly different between Raw features and Z-Scored because we have done a feature expansion operation that computes the dot product inside another embedding space, so the model is sensitive to the transformation of data. We obtain the best model with a value of λ chosen 10^2 and no Z-Norm applied.
We can conclude that classes are better separated with quadratic decision rules. Now we'll test the remaining models.

2.4 Support Vector Machine

In this section, we will focus on Support Vector Machine (SVM) model. The problem of minimizing risk in Logistic Regression can be extended to a more general problem that allows for the separation of samples up to a certain margin. This approach is known as Support Vector Machine. To solve the SVM problem, we can use the dual formulation, which is easier to optimize because its complexity depends only on the number of samples. This also allows us to compute non-linear hyperplanes without having to explicitly expand the features:

$$J_D(\alpha) = -\tfrac{1}{2}\alpha^T H\alpha + \alpha^T 1$$

$$\text{with } 0 \leq \alpha_i \leq C, \forall i \in \{1, ..., n\} \text{ and } \textstyle\sum_{i=1}^{n} \alpha_i z_i = 0$$

Even in this case, such as in Logistic Regression, we can consider a balanced version of the SVM where balancing is done by considering different values of C for each class in the box constraint of the dual formulation:

$$\text{where } C_i = \begin{cases} \frac{C}{\pi_T}\frac{\pi^T_{emp}}{\pi_T} & \text{if } i \in \text{Class1} \\ \frac{C}{\pi_F}\frac{\pi^F_{emp}}{\pi_F} & \text{if } i \in \text{Class0} \end{cases}$$

The implemented SVM models are:

- Linear SVM:
  - That we can obtain by solving the primal problem, expressed as the minimization of:

$$J(\hat{w}) = \tfrac{1}{2}\|\hat{w}\|^2 + C\textstyle\sum_{i=1}^{n} \max(0, 1 - z_i(\hat{w}^T\hat{x}_i))$$

where $\hat{x}_i = \begin{bmatrix} x_i \\ K \end{bmatrix}$ and $\hat{w} = \begin{bmatrix} w \\ b \end{bmatrix}$

- Quadratic SVM
  - The dual SVM formulation depends on the samples through dot products:
    $$H_{ij} = z_i z_j x_i^T x_j$$
    This property allows us to calculate scores without having to explicitly expand the features. All we need is the ability to compute dot products between training and test samples. If we have a function that can efficiently calculate dot products in the expanded space, then we can use a kernel function, k, for both training and scoring.
    $$k(x_i, x_j) = \Phi(x_i)^T \Phi(x_j)$$
    This enables us to find a linear separation surface in the expanded space, which corresponds to a non-linear separating surface in the original feature space. There are two types of kernel functions that we can use for this purpose:
    * **RBF-Radial Basis Kernel Function:** $k(x_1, x_2) = e^{-\gamma \|x_1 - x_2\|^2}$
    * **Polynomial:** $k(x_1, x_2) = (x_i^T x_j + c)^d$

2.5 Expectations

As mentioned before, we expect that quadratic separation surfaces will perform better than linear ones.

2.6 Results:

2.6.1 Linear SVM

Since we do not expect the linear SVM to give any benefits, we simply tested it with various C and only K = 1. Below are graphs showing that the minDCF does not improve on previous quadratic models.

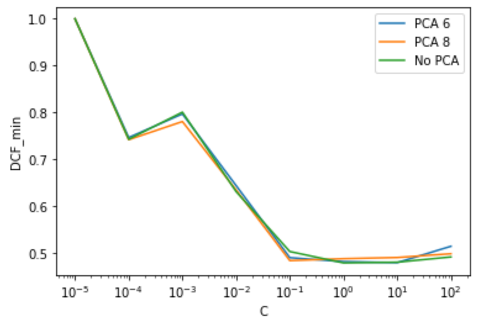


Figure 15:
Comparison between PCAs with a linear SVM

C	PCA 6	PCA 8	No PCA
10^{-5}	1	1	1
10^{-4}	0.746	0.742	0.743
10^{-3}	0.797	0.780	0.801
10^{-2}	0.644	0.634	0.631
10^{-1}	0.490	**0.483**	0.503
1	0.481	0.487	**0.478**
10	**0.479**	0.490	0.480
10^{2}	0.514	0.498	0.491

Table 9:
DCFmin for Linear SVM

As shown in the table, we have the best value for the no PCA case, with $C = 1$, but we considered this case to be of little relevance, as demonstrated by the small difference between the DCFmin of PCA 6 and no PCA, which is why we continued to test for PCA 6 in order to have a more accurate analysis

We also tried to see if Z-norm (applied before PCA) could improve the performances but we saw that it doesn't.

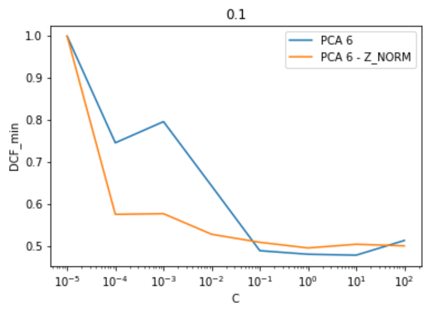


Figure 16:
Comparison between PCA 6 with Z-norm
and no Z-norm

2.6.2 Quadratic SVM - Polynomial kernel

Even in this case, we have a hyper-parameter to tune "C" and we will consider a range of values in a logarithmic fashion. For the polynomial kernel case we tried to test degree (d) 2, with different values of c.

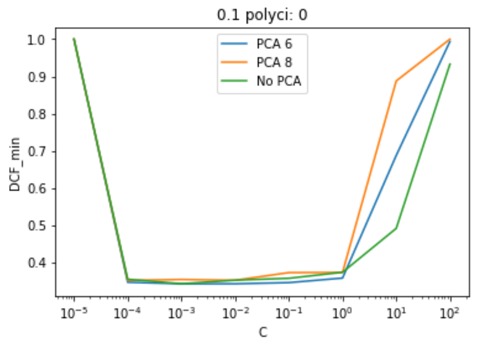


Figure 17:
Polynomial kernel of degree=2 for $K = 1$ and $c = 0$

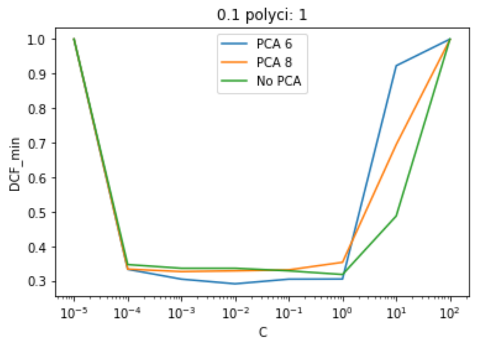


Figure 18:
Polynomial kernel of degree=2 for $K = 1$ and $c = 1$

We can say that models without PCA and with PCA 6 perform better, especially those using a value of $10^{-3} \leq C \leq 1$. Having said that, we choose to try polynomial kernel with $K = 0$ only using PCA 6 and without PCA, to check the performance.

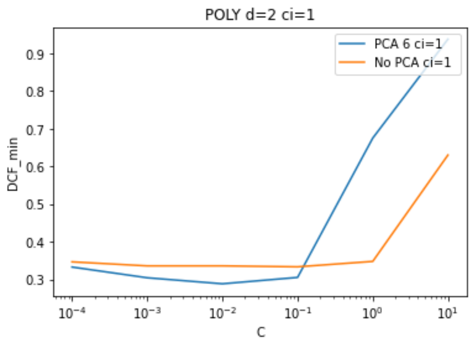


Figure 19:
Polynomial kernel of degree=2 for $K = 0$ and $c = 1$

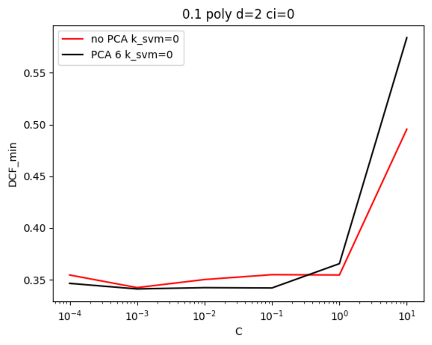


Figure 20:
Polynomial kernel of degree=2 for $K = 0$ and $c = 0$

At this point, we saw that for $K = 0$ and $c = 1$ polynomial kernel have a minimum in $C = 10^{-2}$, as the $K = 1$ and $c = 1$ case. We compared the two cases, ultimately choosing the first one after seeing that it was better.

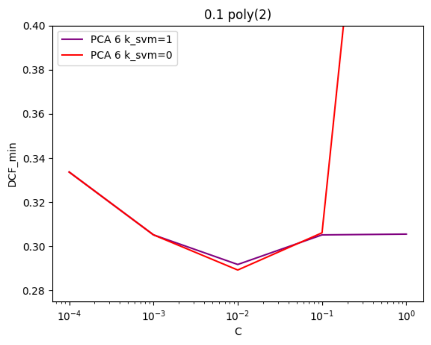


Figure 21:
Comparison between $K = 0$ and $K = 1$ case

For this kernel mode, we also try to check if with Z-norm and degree $d = 3$ we could have any improvement, but we saw that these perform worse, which is why we no longer included them in our analysis of kernel SVM.

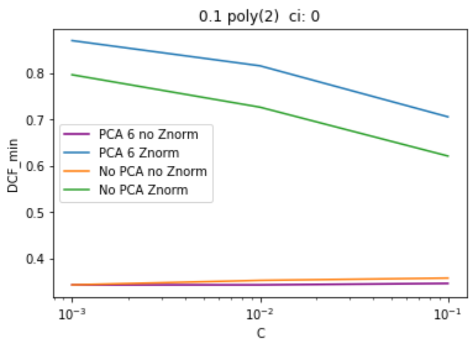


Figure 22:
Polynomial $d = 2$ and $c = 0$

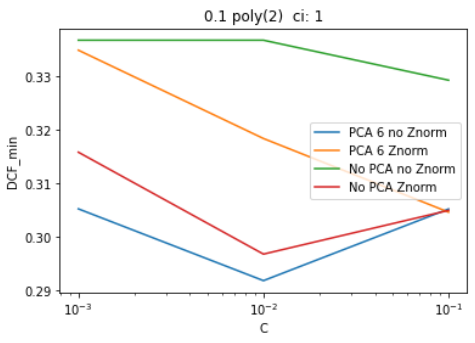


Figure 23:
Polynomial $d = 2$ and $c = 1$

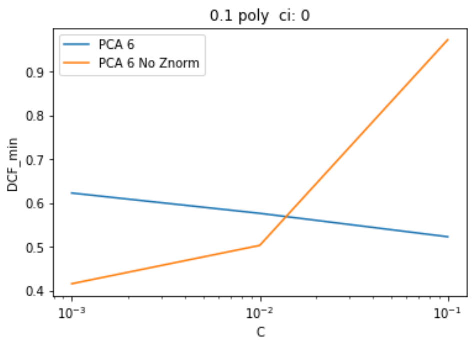


Figure 24:
Polynomial $d = 3$ and $c = 0$

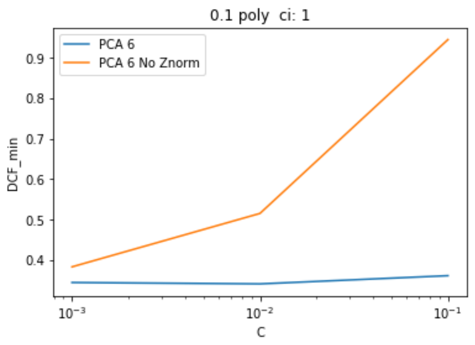


Figure 25:
Polynomial $d = 3$ and $c = 1$

Below, the results we obtained with the best case:

K	C	no Z-norm
0	10^{-4}	0.334
0	10^{-3}	0.305
0	10^{-2}	**0.289**
0	10^{-1}	0.306
0	1	0.675
0	10	0.938

Table 10: SVM-Polynomial kernel with $d = 2$, $K = 0$ and $c = 1$ for PCA 6

K	C	no Z-norm
0	10^{-4}	0.347
0	10^{-3}	0.337
0	10^{-2}	0.337
0	10^{-1}	**0.334**
0	1	0.349
0	10	0.630

Table 11: SVM-Polynomial kernel with $d = 2$, $K = 0$ and $c = 1$ for No PCA

K	C	no Z-norm
1	10^{-4}	0.334
1	10^{-3}	0.305
1	10^{-2}	**0.292**
1	10^{-1}	0.305
1	1	0.305
1	10	0.923

Table 12: SVM-Polynomial kernel with $d = 2$, $K = 1$ and $c = 1$ for PCA 6

K	C	no Z-norm
1	10^{-4}	0.347
1	10^{-3}	0.337
1	10^{-2}	0.337
1	10^{-1}	0.329
1	1	**0.319**
1	10	0.488

Table 13: SVM-Polynomial kernel with $d = 2$, $K = 1$ and $c = 1$ for no PCA

2.6.3 Quadratic SVM - RBF kernel

Since K=0 performed better than K=1, we use this configuration for the analysis of RBF kernel. In order to find a first possible configuration for the hyperparameters, i.e. C, K, and γ, we initially try for various values of γ and C (using $K = 0$).

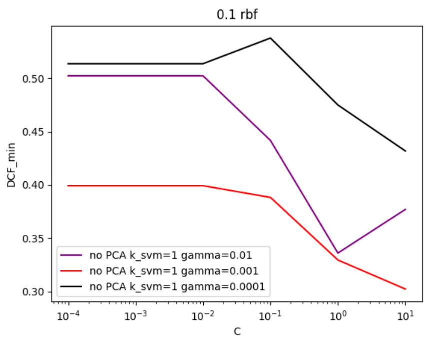


Figure 26:
RBF kernel with $K = 0$ for no PCA

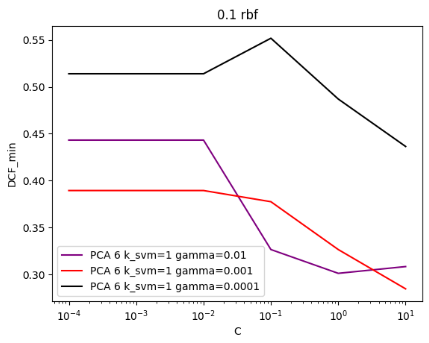


Figure 27:
RBF kernel with $K = 0$ for PCA 6

γ	$C = 10^{-4}$	$C = 10^{-3}$	10^{-2}	10^{-1}	1	10
10^{-4}	0.514	0.514	0.514	0.552	0.487	0.436
10^{-3}	**0.390**	**0.390**	**0.390**	0.378	0.326	**0.285**
10^{-2}	0.443	0.443	0.443	**0.327**	**0.301**	0.309

Table 14: SVM-Polynomial kernel with $d = 2$, $K = 0$ and $c = 1$ for PCA 6

γ	$C = 10^{-4}$	$C = 10^{-3}$	10^{-2}	10^{-1}	1	10
10^{-4}	0.514	0.514	0.514	0.538	0.475	0.432
10^{-3}	**0.399**	**0.399**	**0.399**	**0.388**	**0.330**	**0.302**
10^{-2}	0.502	0.502	0.502	0.442	0.336	0.377

Table 15: SVM-Polynomial kernel with $d = 2$, $K = 0$ and $c = 1$ for no PCA

From these results, we can say that PCA does not bring substantial improvements, but it allows us to work with fewer features.

To confirm our previous hypotheses, we also check the performance of PCA 8 and if $K_{svm} = 1$ was actually worse performing. The results we have obtained are consistent with what has been previously stated, so we can consider as our best choice PCA 6 with $\gamma = 0.001$, and C=10.

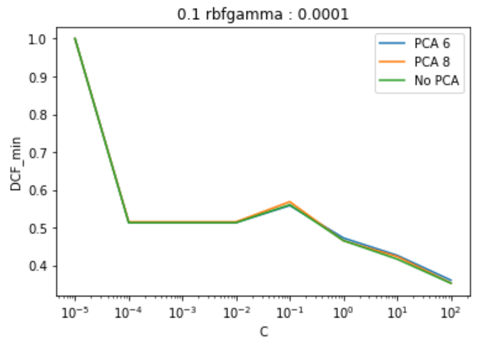


Figure 28:
RBF kernel with $\gamma = 0.0001$ and $K = 1$

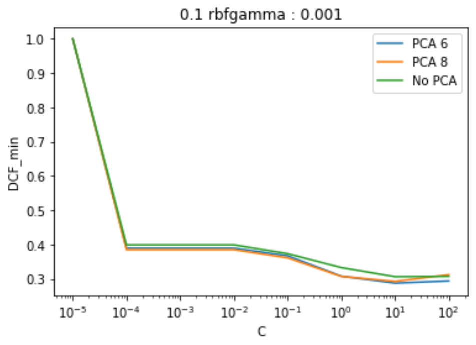


Figure 29:
RBF kernel with $\gamma = 0.001$ and $K = 1$

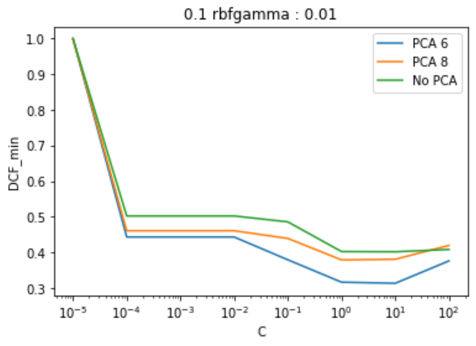


Figure 30:
RBF kernel with $\gamma = 0.01$ and $K = 1$

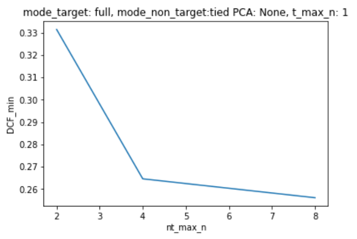
2.7 Gaussian Mixture Model - GMM

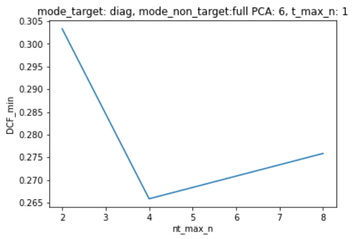
The last type of classifier we test is the Gaussian mixture model. This assumes that it is more convenient to represent data with Gaussian distributions composed of multiple components (or clusters). Results on the first Gaussian models suggested discrete performance for the MVG and Naive Bayes models. In particular, the latter performed slightly worse due to the slight correlation between some classes. However, since this was not so prevalent, the performance between MVG and NB was almost comparable. So we expect to get similar results if not better than those just described for Gaussian models.
We approached the problem by trying different combinations of models and number components for Non-Target class (Spoofed) and Target Class (Authentic). In particular, we used:

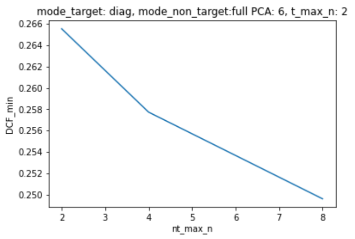
- (1,2) components for class 'Authentic'
- (2,4,8) components for class 'Spoofed'

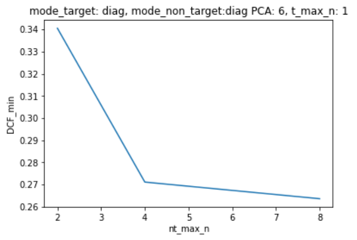
We know that Spoofed samples are distributed into 6 different sub-classes, so we expected more optimistic results when representing Spoofed fingerprints with a higher number of components. We then tried to configure our model trying with all the combinations of the Full-covariance model, Diagonal-covariance model and Tied Full-covariance model for both Target-class and Non-Target class. Because not so useful during the previous analysis, we did not apply the Z-norm for the GMM models.

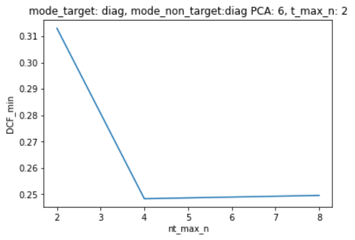
We report just some of the most significant plots:

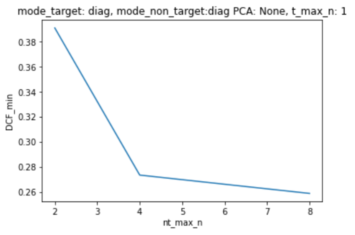


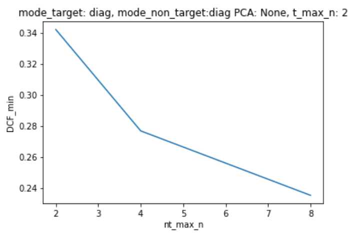


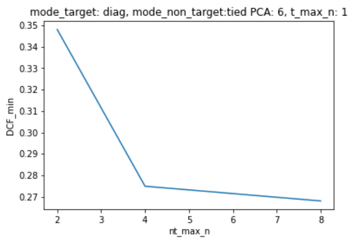


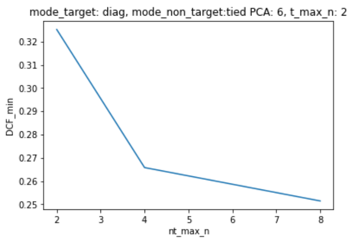












So we reported some of the most iconic plots. Among some of them, you can find some of the lowest DCF values. It is interesting to note that the best results are obtained by applying 8 components to the non-target class, which confirms what was previously assumed about the Spoofed Fingerprint distribution over several sub-classes. In addition, performance is better for the number of components equal to 2 for the target class. A value however low enough to confirm the analysis at the beginning of the report about the Gaussian distribution of Authentic samples. The same can be said when you notice that the best performance is obtained for the configuration with Diag Cov for both classes. Only the absence of PCA for the best configuration deviates from what has been stated so far.
We decided to report just the results for the best configurations, so the cases with Target class samples represented by a Diagonal Covariance Gaussian

Components/(TargetMode-NonTargetMode)	Diag-Full	Diag-Diag	Diag-TiedFull
(2,1)	**0.303**	0.340	0.348
(4,1)	**0.266**	0.271	0.275
(8,1)	0.276	**0.264**	0.268
(2,2)	**0.265**	0.313	0.325
(4,2)	0.258	**0.248**	0.266
(8,2)	0.250	**0.249**	0.251

Table 16: minDCF for Target mode = DiagCov and PCA6

Components/(TargetMode-NonTargetMode)	Diag-Full	Diag-Diag	Diag-TiedFull
(1,2)	**0.303**	0.391	0.353
(1,4)	**0.284**	0.273	0.290
(1,8)	0.296	**0.259**	0.278
(2,2)	**0.270**	0.342	0.321
(2,4)	**0.248**	0.276	0.268
(2,8)	0.252	**0.235**	0.253

Table 17: minDCF for Target mode = DiagCov and PCANone

We find the best configuration with Diagonal Covariance applied to both Target class and Non-Target class and 2 components for the Target class and 8 components to represent the Non-Target class, without PCA. This is an absolutely consistent result with what we have seen so far, considering the level of correlation between the features, the Gaussian distribution of the Target class and the number of sub-classes (remembering it is 6, using 8 components for Non-Target class seems quite reasonable) in which the samples of the non-target class are divided. We note that with this latest model, we were able to find the minimum DCF among all models.

3 Final Considerations about Validation

We resume the best three models' information with the following table:

Models	minDCF
GMM Target Mode=DiagCov e Non-Target Mode=DiagCov (2,8), PCA=None	0.235
Quadratic Logistic Rregression lambda = 10^{-2}, PCA = 6, piT = 0.1	0.263
SVM with kernel RBF, KSVM = 0, gamma = 10^{-3}, C = 10, PCA = 6, piT = 0.1	0.285

Table 18: Best Models

In conclusion, we consider the previously chosen GMM model as the best solution for our problem.

4 Calibration

To evaluate the different models, so far we have used only the minimum cost of detection (minDCF), which however depends on a threshold. To understand if the threshold is the theoretical one, we will use a metric called actual DCF (actDCF). The method that we will adopt is based on Logistic Regression, which works like a relation of verisimilitude to posterior, so we can obtain the calibrated score by simply subtracting the theoretical threshold. To estimate the parameters of the calibration function, we will use a K-Fold approach, since the number of samples we have is limited. We take just the best 3 models to calibrate. Consider "DCF" string in the legend as a reference to "actDCF".

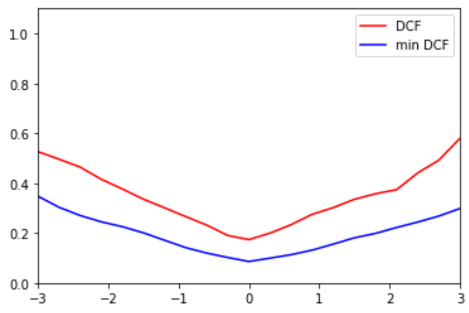


Figure 31: Quad LR model not calibrated

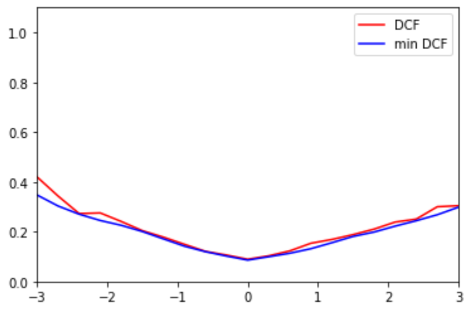


Figure 32: Quad LR model calibrated

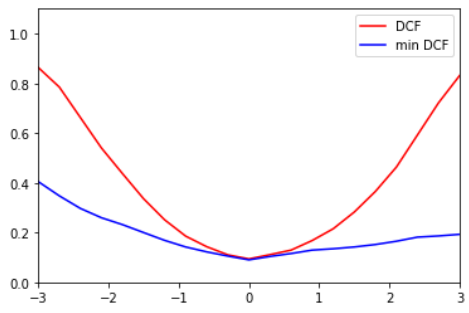


Figure 33: SVM RBF model not calibrated

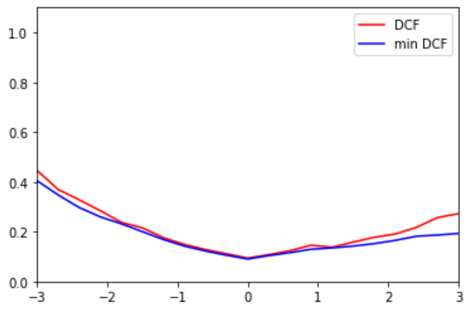


Figure 34: SVM RBF model calibrated

Although the GMM model did not need calibration, we tried to apply it anyway. In fact, the results remained almost unchanged.

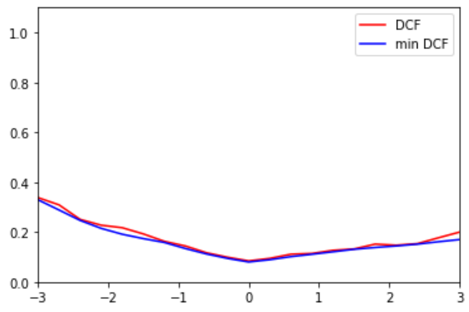


Figure 35: GMM model not calibrated

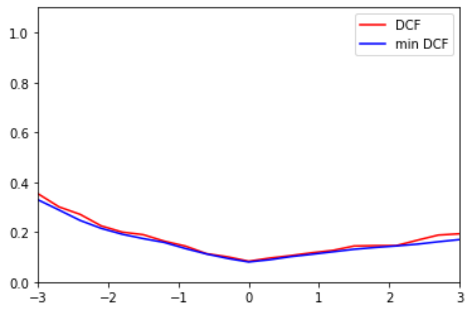


Figure 36: GMM model calibrated

As we can see LR and RBF SVM have the score not calibrated and if we make calibration, we can obtain some improvements. Looking at these Bayes Error plots we can also have an idea about how our best models perform in other working points.

5 Evaluation

Now we move on to the testing phase and check if the choices made previously during the Valuation phase were optimal or not. We also make a comparison by testing models by applying hyperparameters of values discarded previously. We will carry out the tests only for the 3 best models: we will not analyze the linear models of Logistic Regression and SVM nor will we analyze the Gaussian models (as already integrated in GMM)
The training and the test will be carried out not through K-Fold but directly using the Test set and the whole Training set.

5.1 Logistic Regression

We first consider the logistic regression models along λ for the same interval that we use during the training. During the Validation phase our best model was LR trained with piT = 0.1 so we will consider the plot for this value of piT:

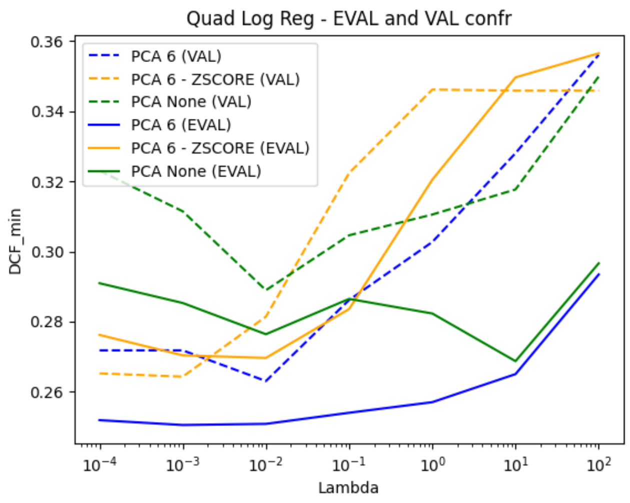


Figure 37: Comparison for Quadratic LR models

From the plot, it is possible to notice as, although the values of DCF are slightly different, the trends of the models are practically the same if we make a comparison with results obtained during Evaluation and those during Validation. We also applied Z-Norm to the model trained on PCA6 which continues to perform worse than the version without Z-Norm. The value $\lambda = 10^{-2}$ still remains the best one to use.

λ	PCA=6	PCA=None	PCA=6 + Z-Norm
10^{-4}	0.252	0.291	0.276
10^{-3}	0.251	0.285	0.270
10^{-2}	**0.250**	0.276	0.270
10^{-1}	0.254	0.286	0.283
1	0.257	0.282	0.320
10^{1}	0.265	0.269	0.350
10^{2}	0.293	0.297	0.356

Table 19: Quadratic LR models (EVAL)

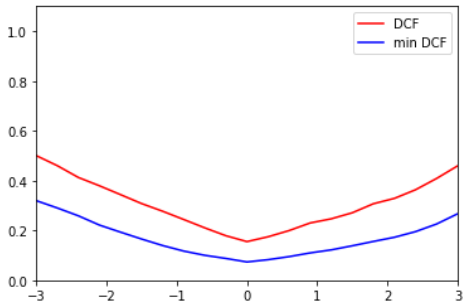


Figure 38: Quadratic LR not calibrated

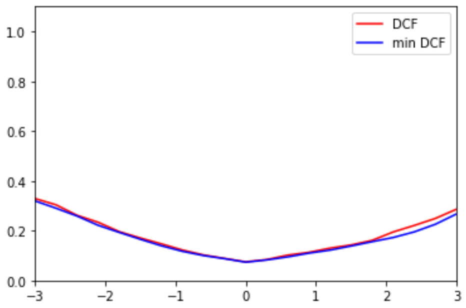


Figure 39: Quadratic LR calibrated

5.2 Support Vectore Machine

In the valuation phase, we have seen that Linear SVM does not give good results, so we choose to analyze only quadratic SVM, using both Polynomial and RBF kernel.

5.2.1 Polynomial kernel - SVM

In order to verify the behavior of our chosen model, we compared the evaluation's results with validation's one. As said before, we use different hyperparameters, in this case we use $c \in [0, 1]$ for the polynomial's kernel setting $K_{svm} = 0$.

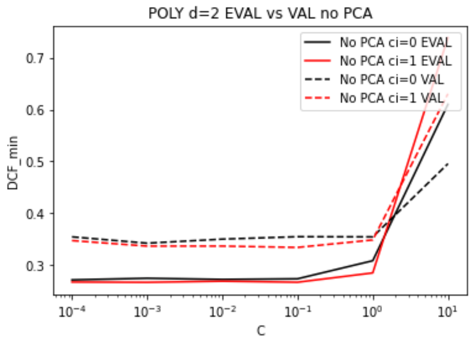


Figure 40:
Comparison for no PCA between validation and evaluation

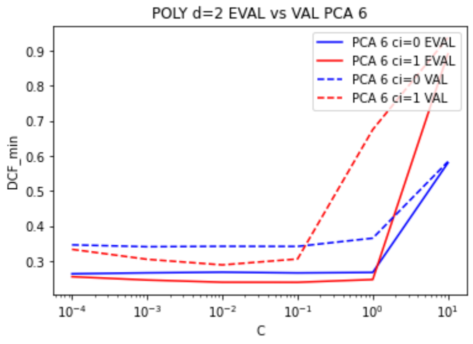


Figure 41:
Comparison for PCA 6 between validation and evaluation

c	C	no Z-norm	Z-norm
0	10^{-4}	**0.264**	0.718
0	10^{-3}	0.267	**0.854**
0	10^{-2}	0.268	0.866
0	10^{-1}	0.266	0.858
0	1	0.267	0.857
0	10	0.580	0.857

Table 20: SVM-Polynomial kernel with $d = 2$, $K = 0$ and $c = 0$ for PCA 6

c	C	no Z-norm	Z-norm
1	10^{-4}	0.256	0.325
1	10^{-3}	0.246	0.305
1	10^{-2}	**0.240**	0.286
1	10^{-1}	**0.240**	0.280
1	1	0.247	**0.274**
1	10	0.891	0.277

Table 21: SVM-Polynomial kernel with $d = 2$, $K = 0$ and $c = 1$ for PCA 6

c	C	no Z-norm	Z-norm
0	10^{-4}	**0.272**	**0.666**
0	10^{-3}	0.275	0.795
0	10^{-2}	**0.272**	0.755
0	10^{-1}	0.274	0.752
0	1	0.309	0.768
0	10	0.610	0.769

Table 22: SVM-Polynomial kernel with $d = 2$, $K = 0$ and $c = 0$ for no PCA

c	C	no Z-norm	Z-norm
1	10^{-4}	0.268	0.304
1	10^{-3}	**0.267**	0.273
1	10^{-2}	0.269	0.264
1	10^{-1}	**0.267**	**0.251**
1	1	0.285	0.266
1	10	0.739	0.269

Table 23: SVM-Polynomial kernel with $d = 2$, $K = 0$ and $c = 1$ for no PCA

Analyzing this values, the results seems to be consistent with the ones obtained during the previous analysis over the training set,with $d = 2$ and $c = 1$ being the best hyperparameters and PCA 6 being our best choice. For better accuracy, we tried to compute even the z-normalization for this part, demonstrating that Z-norm does not improve performance even in the evaluation's phase.

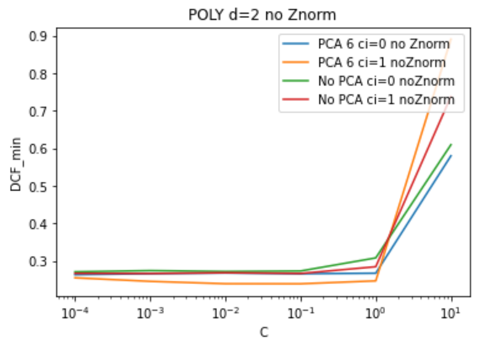


Figure 42:
Comparison between PCA 6 and no PCA (no Z-norm)

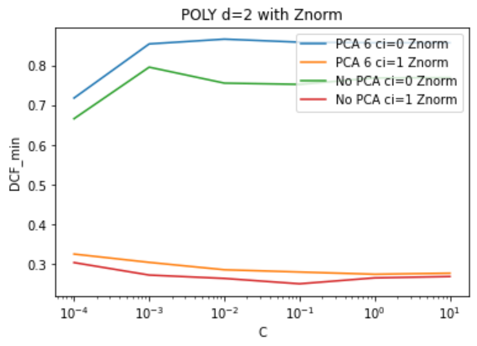


Figure 43:
Comparison between PCA 6 and no PCA (Z-norm)

5.2.2 RBF kernel - SVM

In this phase, we check that with RBF kernel we have the same results that in the validation's phase. Even in this step we fixed the K value to 0, changing γ values in order to choose the best hyperparameter.

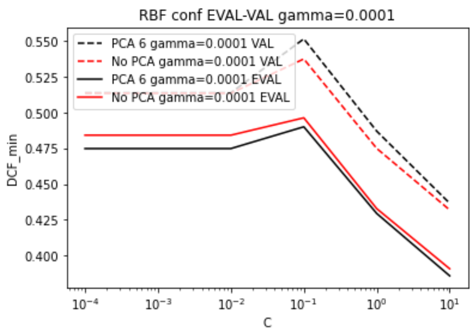


Figure 44:
Comparison between evaluation and validation phase ($\gamma = 0.0001$)

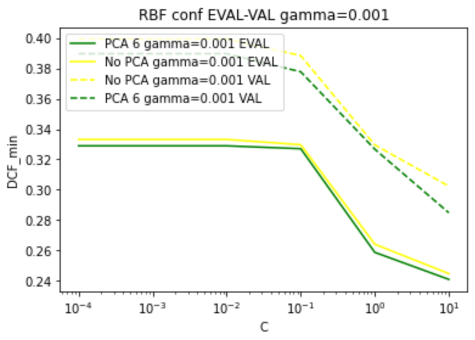


Figure 45:
Comparison between evaluation and validation phase ($\gamma = 0.001$)

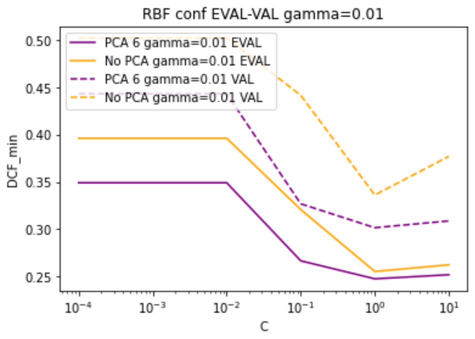


Figure 46:
Comparison between evaluation and validation phase ($\gamma = 0.01$)

γ	$C = 10^{-4}$	$C = 10^{-3}$	$C = 10^{-2}$	$C = 10^{-1}$	$C = 1$	$C = 10$
0.0001	0.475	0.475	0.475	0.490	0.429	0.386
0.001	0.329	0.329	0.329	0.327	0.259	**0.241**
0.01	0.349	0.349	0.349	0.266	0.247	0.252

Table 24: SVM-RBF kernel with $K = 0$ for PCA 6 (no Z-norm)

γ	$C = 10^{-4}$	$C = 10^{-3}$	$C = 10^{-2}$	$C = 10^{-1}$	$C = 1$	$C = 10$
0.0001	0.484	0.484	0.484	0.496	0.433	0.391
0.001	0.333	0.333	0.333	0.330	0.264	**0.245**
0.01	0.396	0.396	0.396	0.321	0.255	0.262

Table 25: SVM-RBF kernel with $K = 0$ for no PCA (no Zn-norm)

γ	$C = 10^{-4}$	$C = 10^{-3}$	$C = 10^{-2}$	$C = 10^{-1}$	$C = 1$	$C = 10$
0.0001	0.509	0.509	0.509	0.509	0.509	0.493
0.001	0.502	0.502	0.502	0.502	0.489	0.475
0.01	0.444	0.444	0.444	0.447	0.400	**0.337**

Table 26: SVM-RBF kernel with $K = 0$ for PCA 6 (Z-norm)

γ	$C = 10^{-4}$	$C = 10^{-3}$	$C = 10^{-2}$	$C = 10^{-1}$	$C = 1$	$C = 10$
0.0001	0.499	0.499	0.499	0.499	0.499	0.478
0.001	0.491	0.491	0.491	0.491	0.472	0.450
0.01	0.430	0.430	0.430	0.428	0.381	**0.312**

Table 27: SVM-RBF kernel with $K = 0$ for no PCA (Z-norm)

Even in this case, the results seems to be consistent with the ones obtained during the previous analysis over the training set,with $\gamma = 10^{-3}$ and $C = 10$ being the best hyperparameters and PCA 6 without Z-normalization being our best choice.

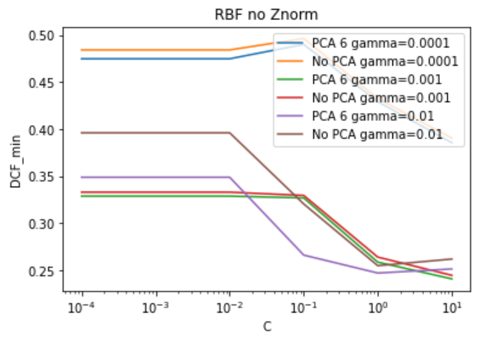


Figure 47:
Comparison between evaluation and validation phase for PCA 6 and no PCA (without Z-Norm)

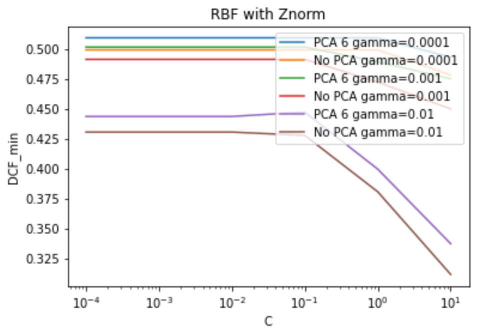


Figure 48:
Comparison between evaluation and validation phase for PCA 6 and no PCA (with Z-Norm)

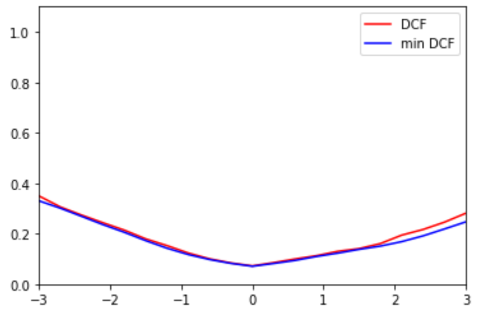


Figure 49:
SVM - Calibrated

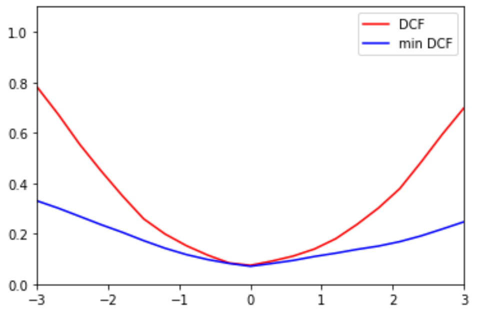


Figure 50:
SVM - Not calibrated

5.3 Gaussian Mixture Model

Finally, we consider the Gaussian Mixture Models. We only focus on the best model Target Mode=DiagCov (2,8) with PCA=None data, showing, in addition, a plot that denotes how the GMM FullCov-FullCov (so applying Full Covariance for both of the classes)works compared with the respective Validation model.

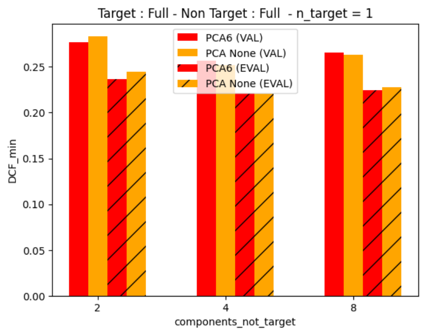


Figure 51: FullCov-FullCov with 1 Target component

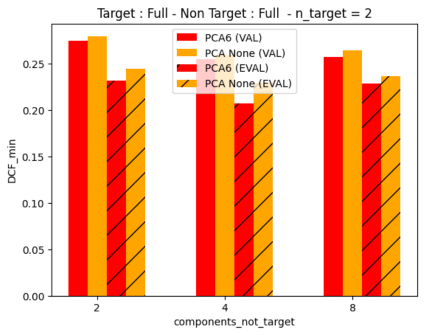


Figure 52: FullCov-FullCov with 2 Target components

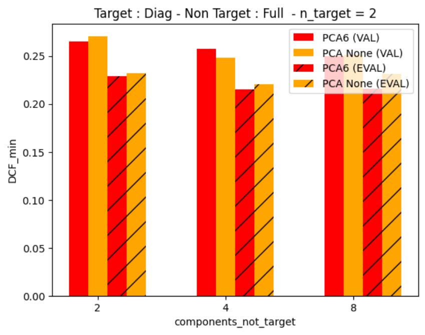


Figure 53: DiagCov-FullCov with 2 Target components

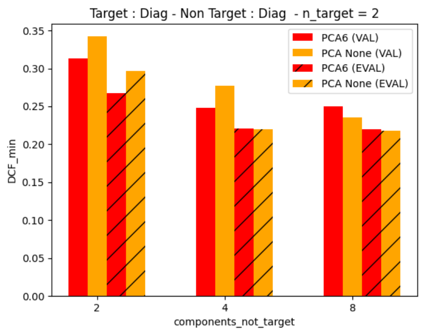


Figure 54: DiagCov-DiagCov with 2 Target components

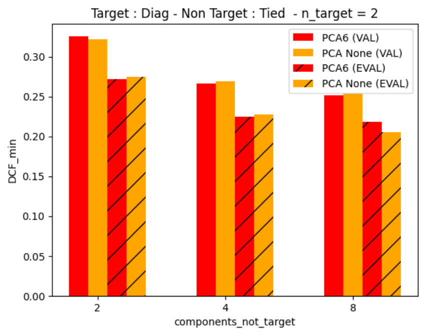


Figure 55: DiagCov-TiedCov with 2 Target components

We can notice some small differences compared to what we saw during the validation phase. In general, however, the trends of the models are very close to what was previously seen and predicted. The solution chosen, however, is a sub-optimal solution because in this case, the performance of the model that describes the Target class with a Diagonal Covariance matrix and the non Target class with a Tied Covariance matrix performs slightly better.

Models/NumComponents	2,2	2,4	2,8
GMM TMode=DiagCov e NTMode=FullCov	0.232	0.220	0.215
GMM TMode=DiagCov e NTMode=DiagCov	0.296	0.219	0.231
GMM TMode=DiagCov e NTMode=TiedCov	0.274	0.227	**0.205**

Table 28: GMM TMode=DiagCov and PCA=None and 2 Non-Target Components

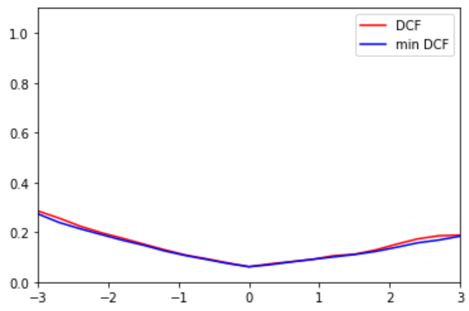


Figure 56: GMM not calibrated

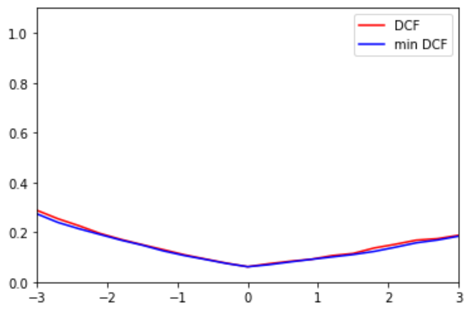


Figure 57: GMM calibrated

As we can see the chosen model is a sub-optimal solution (the global-optimal one is obtained with model Diag-Tied with DCF = 0.205), but the performances are less or more the same.

6 Conclusion

The results obtained during the Evaluation phase are similar to those obtained during the Validation. This suggests that the evaluation population has a distribution similar to that of validation. We went through several configurations, convinced that the GMM model with DiagCov for the Target Class and TiedCov for the non-target class and (2.8) components was the best configuration. We made a very small mistake when we discovered that a globally excellent solution is obtained if you use a TiedCov for the Non-Target class. In fact, with a DCFmin=0.231, with the optimal solution it drops to DCFmin=0.205