FINGERPRINT SPOOFING DETECTION
COURSE TITLE: MACHINE LEARNING AND PATTERN RECOGNITION

Submitted by

Milena Yahya s306978

Submitted to

Prof. Sandro Cumani

Department of Control and Computer Engineering DAUIN

POLITECNINO DI TORINO

Turin, Italy

September 2024

Contents

Abstract

The goal of this project is to detect manipulated fingerpints. In the context of machine learning, this reduces to a binary classification task, where the goal is to classify fingerprints into two classes: genuine (authentic), and fake (spoofed). To this aim, we employ various machine learning algorithms, and we explore multiple hyper-parameter combinations and configurations. We compare the obtained using different performance metrics, in search for the model that is able to differentiate the two classes in the most effective way. Note that we denote the genuine class with label 1, and the fake class with label 0.

Workflow

In this project, we analyze a labeled dataset consisting of 6000 samples, each represented by 6 features. The dataset is divided into two classes: 2990 samples labeled as *genuine* and 3010 as *fake*. We begin by examining the characteristics of each feature individually and in combination, to assess the separability of the data.

Next, we explore data preprocessing techniques such as Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA) to enhance the dataset's suitability for classification tasks. Additionally, we fit univariate Gaussian models to the individual features, identifying those that can be well-approximated by Gaussian distributions.

For generative modeling, we split the dataset into training and evaluation sets. We then train models such as Multivariate Gaussian (MVG), Tied-Covariance MVG, and Naive Bayes, and evaluate their performance using Optimal Bayes Decisions. We compute metrics such as the Detection Cost Function (DCF) and its minimum value (minDCF), while also experimenting with PCA and varying effective priors.

Then, we implement discriminative classification models, including Logistic Regression, Support Vector Machines (SVM), and Gaussian Mixture Models (GMM). These models are evaluated to identify the best-performing approach for the given data.

In the last stage, we apply score calibration techniques to adjust the model outputs for better decision-making. We also perform score fusion, combining the outputs of multiple models to leverage complementary strengths and improve overall performance. Finally, the model delivering the lowest *minDCF* and hlowest *actualDCF* is selected as the best-performing approach.

1. Feature Analysis

In this section we aim to analyze our labeled dataset consisting of 6000 samples, with 2990 labeled as *genuine* and 3010 labeled as *fake*. With this split, it is fair to say that the dataset is quite balanced and we have an even distribution. Before applying any pre-processing techniques, we visualize the distribution of the features using histograms.

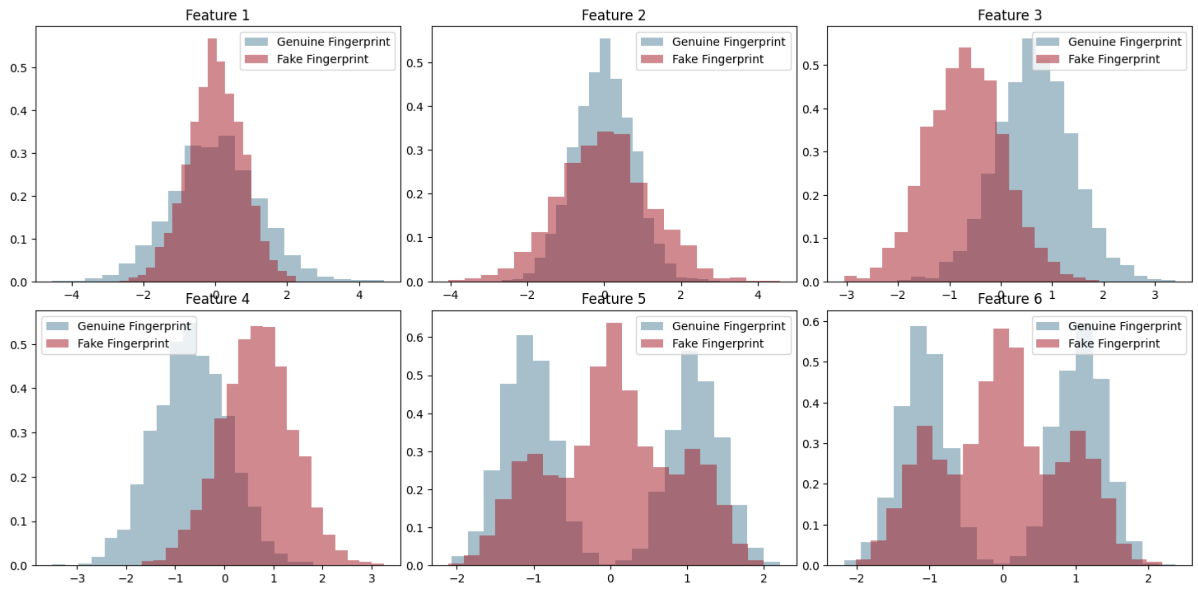


Figure 1.1: The distribution of the raw features of the fingerprints dataset

We observe that a Gaussian distribution effectively models only features 1 to 4 of the dataset, whereas Gaussian densities may not adequately represent the last two features of the dataset.

- For the first two features, the classes have a large overlap. This means it is difficult to differentiate the two classes using these two features, they do not provide class separability. As evident in the histograms, the two classes have similar mean but different variance when considering the first two features, and there is one single peak. This one peak implies there is a single mode, so the features can be represented by a Gaussian distribution.
- The third and fourth features provide the most efficient class separation among the six features, as there is the least overlap in the histograms. These features present a different mean for the two classes, but similar variance, and there is also only one peak. This one peak implies there is a single mode, so the features can be represented by a Gaussian distribution.

- As for the last two features, there is two modes for for class Genuine, and three modes for class Fake, meaning that a class cannot be represented using one single Gaussian distribution based on the last two features. We also observe significant overlap of the two classes, implying we cannot rely on the last two features to separate the classes. We can see three clusters for each class

In short, features 4 and 5 could be said to be the most discriminatory features for this dataset. Next, to enhance our understanding of the dataset, we will create scatter plots of feature pairs along with histograms for a more comprehensive visualization.

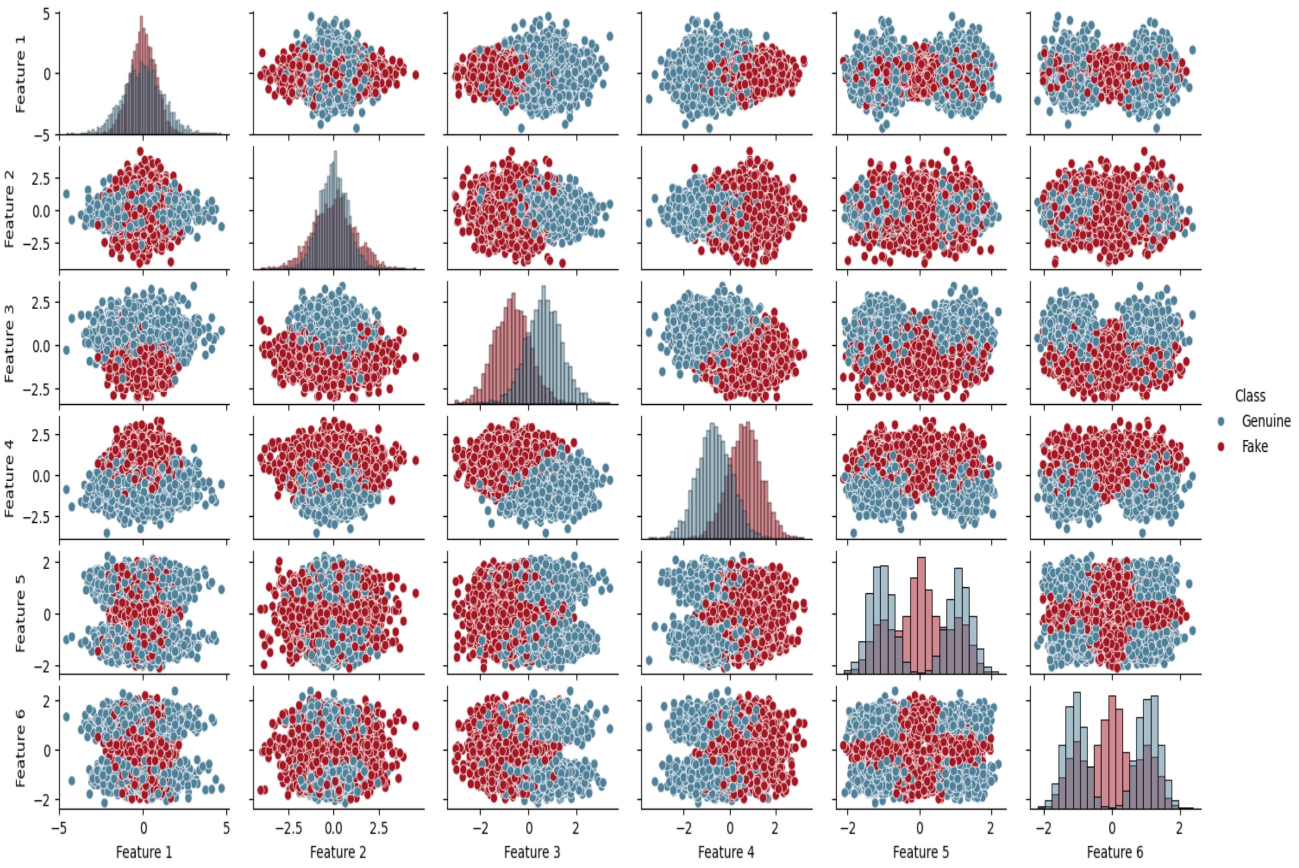


Figure 1.2: Pair-wise scatter plots of the different features

The observed 2D plots confirm that the two classes can be modeled by a Gaussian distribution using the first four features, as there is one cluster per class, while there are multiple clusters for each class when considering features 5 and 6. Further, there is no feature that effectively separates the two classes, i.e, there is no feature for which the two classes do not overlap at all.

1.1 PCA

Next, we proceed with data pre-processing to gain deeper insights from our features. We employ an unsupervised approach called Principal Component Analysis (PCA). PCA helps us identify the directions of maximum variability within the data and projects the features onto these directions. Below, we visualize the features projected onto the PCA directions:

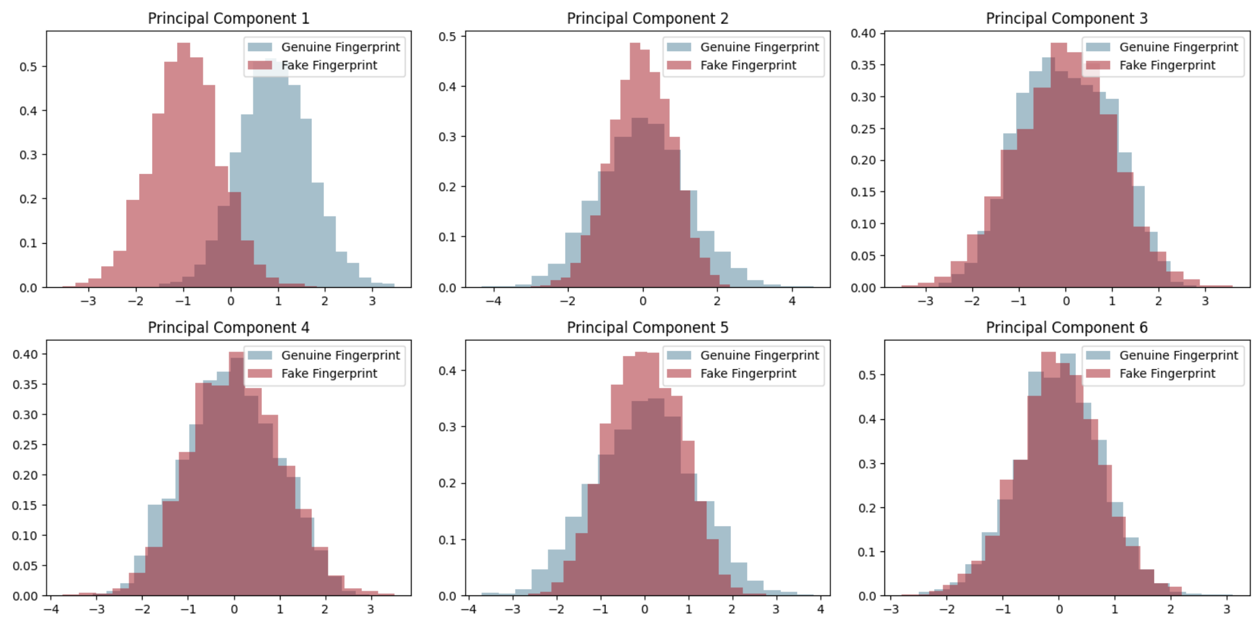


Figure 1.3: The features projected on the 6 PCA directions

As anticipated, the first principal component, which corresponds to the eigenvector with the largest eigenvalue, offers the greatest class separability. This means that genuine and fake fingerprints are most distinguishable when the data is projected onto this direction. However, some overlap between the classes still remains. We further plot the PCA-projected features in 2D-scatter plots:

The plots confirm that the first two principal component directions capture the most variance of the two classes. The two classes form two clusters, but some overlap between the classes remains, indicating that the first two principal components alone may not be sufficient for perfect classification. We conclude that for our dataset, we cannot reply solely on the features to distinguish between the two classes.

1.2 LDA

While PCA finds the direction with maximum variability, Linear Discriminant Analysis (LDA) finds the direction that provides the maximum separability between the classes. The LDA direction has a large separation between the classes and small spread inside each class. We apply 1-dimensional LDA on our features, since we have only two classes. The results are shown below:

LDA does indeed separate the two classes much more efficiently than any of the raw features, but some overlap still exists, and thus LDA projection is also not sufficient for complete class separation.

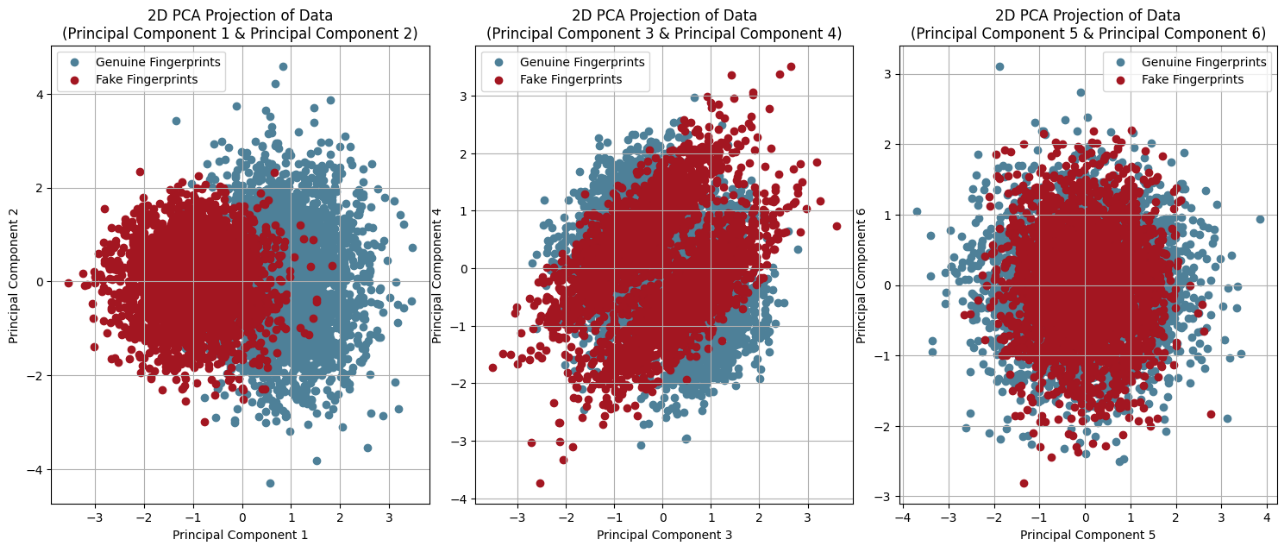


Figure 1.4: Scatter plots of PCA-projected features

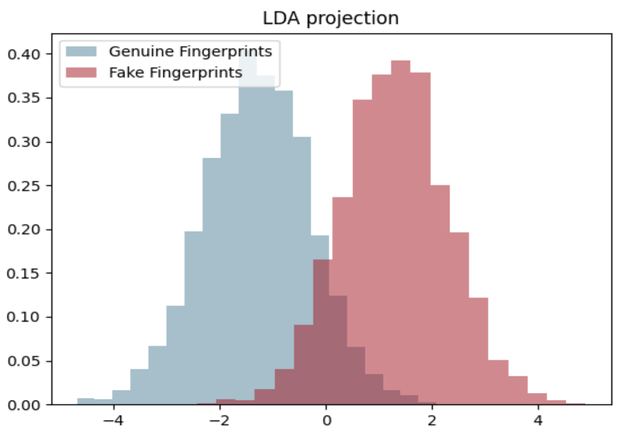


Figure 1.5: Features projected on the LDA direction

1.3 LDA as a Classifier

For this part, we split the dataset into a training set and a validation set. We train LDA using only training data, and then we project the validation data on the computed LDA direction. Below we visualize the training and validation data with LDA:

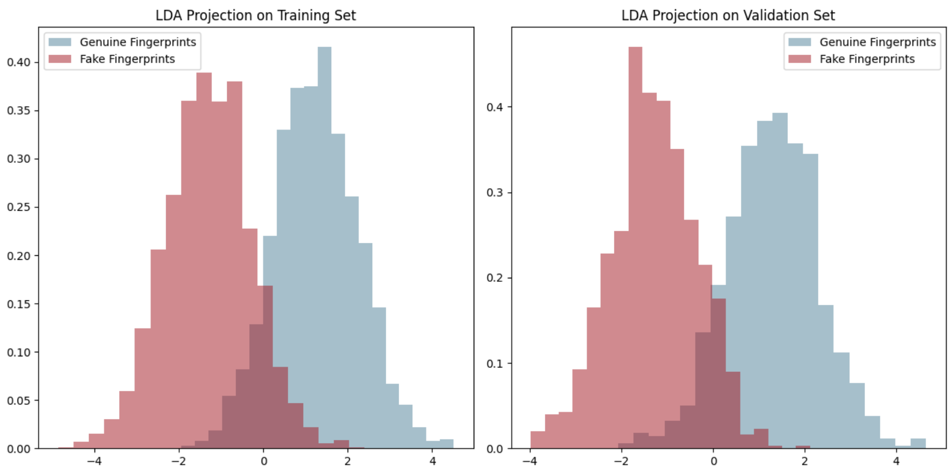


Figure 1.6: LDA projection

We use a threshold to classify the data of the validation set. We employ the mean of the projected class means, computed on the model training data as threshold, and assign data greater than the threshold to class genuine. Then, as a performance measure, for the moment we simply consider the ratio of correctly classified samples over the total number of samples. For our dataset, we calculate the threshold to be **-0.01853437**. We obtain the following result:

Threshold	Error	Accuracy
base threshold	0.093	0.907
threshold/5	0.0925	0.9075
threshold/10	0.093	0.907
threshold/100	0.093	0.907
threshold*5	0.092	0.908
threshold*100	0.372	0.628
threshold*500	0.496	0.504

Table 1.1: Thresholds, Errors, and Accuracies

It seems that increasing the threshold makes the model perform worse, while decreasing it does not affect the error significantly.

We now investigate whether applying PCA as a pre-processing technique enhances the performance. We train the PCA using the training subset of data only, then we project the validation data on the PCA direction. We continue as in the previous section to employ LDA for classification. We summarize the obtained results in the following table, while experimenting with the number pf PCA dimensions m.

m	1	2	3	4	5	6
correct labels	1813	1821	1815	1815	1814	1814
accuracy	0.9065	0.9105	0.9075	0.9075	0.907	0.907
error	0.0935	0.0895	0.0925	0.0925	0.093	0.093

Table 1.2: Performance of the LDA classifier as function of PCA dimensions m

We observe that although applying PCA does not imporve the results dramatically, it does perform better than LDA without PCA. Moreover, for a PCA dimension of $m = 2$, we obtain the best performance so far, with an error of **0.0895** as opposed to **0.093** without PCA. We conclude that PCA can be beneficial for our task when combined with the LDA classifier.

1.4 Gaussian Density Estimation

To analyze further the distribution of the features, we fit uni-variate Gaussian models to each feature of the dataset for each class. We compute the mean and variance of the Gaussian distribution using the Maximum Likelihood approach. Below we visualize the features with histograms, and we fit them with a Gaussian density:

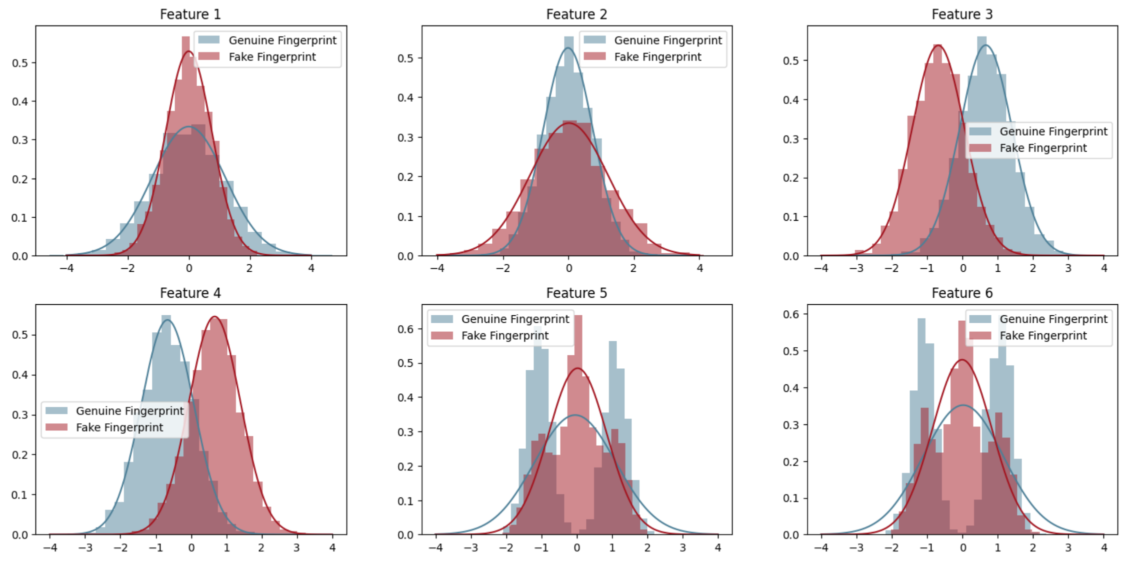


Figure 1.7: Fitting uni-variate Gaussian densities to the features

The figures show that the Gaussian model represents well features 1,2,3, and 4, but it is significantly less accurate for features 5 and 6, since they have more than one peak, and thus cannot be modeled by one single Gaussian density.

2. Generative Models for Classification

2.1 Gaussian Classifiers

In this section, we implement and compare three Gaussian models:

- MVG (Full Covariance): This classifier presumes that each class can be estimated by a Gaussian distribution with a full covariance matrix, capturing the relationships between all feature pairs.
- Naive Bayes (Diagonal Covariance): This method assumes that features are independent of one another, resulting in covariance matrices that are diagonal.
- Tied MVG (Unique Covariance): Each class has its unique mean, but they all share a common covariance matrix, which is calculated over the entire dataset.

We also explore pre-processing the data using PCA before applying these classifiers, experimenting with different PCA dimensions m. For a comprehensive comparison, we include the LDA model from the previous section alongside these classifiers. (see Table 2.1)

In comparing the Tied Covariance (TC) model with the Full Covariance (MVG) and Linear Discriminant Analysis (LDA) models, the MVG model consistently outperforms both TC and LDA across all PCA dimensions, demonstrating the highest accuracy and lowest error rates. The Naive Bayes (NB) Gaussian model performs better than the TC model, especially without PCA and in most PCA dimensions. Interestingly, LDA and TC models show identical performance. This can be explained by the fact that the LDA subspace contains all the information used by the tied model **Overall, the superior performance of the MVG model implies that the features in the data are likely not independent, and the classes have distinct covariance structures that MVG can capture most effectively**. The Naive Bayes model's relatively strong performance also indicates that while some independence assumptions hold, the more detailed covariance structure in MVG provides a better fit for the data.

The fact that NB outperforms TC indicates that assuming feature independence (diagonal covariance) is more effective than using a shared covariance matrix (tied covariance) in this context, while the similar performance of LDA and TC suggests that the assumption of shared covariance in TC and the linearity in LDA do not provide distinct benefits over each other.

Classifier	PCA	Error	Accuracy
MVG	**-**	**0.0700**	**0.9300**
	1	0.0925	0.9075
	2	0.0880	0.9120
	3	0.0880	0.9120
	4	0.0805	0.9195
	5	0.0710	0.9290
	6	0.0700	0.9300
TC	-	0.0930	0.9070
	1	0.0935	0.9065
	2	0.0925	0.9075
	3	0.0925	0.9075
	4	0.0925	0.9075
	5	0.0930	0.9070
	6	0.0930	0.9070
NB	-	0.0720	0.9280
	1	0.0925	0.9075
	2	0.0885	0.9115
	3	0.0900	0.9100
	4	0.0885	0.9115
	5	0.0875	0.9125
	6	0.0890	0.9110
LDA	-	0.0930	0.9070
	1	0.0935	0.9065
	2	0.0895	0.9105
	3	0.0925	0.9075
	4	0.0925	0.9075
	5	0.0930	0.9070
	6	0.0930	0.9070

Table 2.1: Classifier performance with different PCA dimensions

2.1.1 Feature covariances and correlations

To verify our analysis above, we calculate the covariance and correlation matrices of each class:

Covariance Matrix of True Class

$$\begin{bmatrix} 1.448 & -0.0147 & 0.0056 & 0.0157 & 0.0195 & -0.0002 \\ -0.0147 & 0.553 & -0.0112 & -0.0091 & -0.0147 & 0.0163 \\ 0.0056 & -0.0112 & 0.557 & 0.0276 & -0.0038 & -0.0146 \\ 0.0157 & -0.0091 & 0.0276 & 0.570 & -0.0117 & 0.0350 \\ 0.0195 & -0.0147 & -0.0038 & -0.0117 & 1.342 & 0.0169 \\ -0.0002 & 0.0163 & -0.0146 & 0.0350 & 0.0169 & 1.304 \end{bmatrix}$$

Covariance Matrix of False Class

$$\begin{bmatrix} 0.601 & 0.0001 & 0.0191 & 0.0193 & 0.0128 & -0.0135 \\ 0.0001 & 1.447 & -0.0161 & -0.0159 & -0.0265 & 0.0229 \\ 0.0191 & -0.0161 & 0.565 & -0.0018 & -0.0069 & 0.0169 \\ 0.0193 & -0.0159 & -0.0018 & 0.542 & 0.0053 & 0.0136 \\ 0.0128 & -0.0265 & -0.0069 & 0.0053 & 0.696 & 0.0158 \\ -0.0135 & 0.0229 & 0.0169 & 0.0136 & 0.0158 & 0.687 \end{bmatrix}$$

Correlation Matrix of True Class

$$\begin{bmatrix} 1.000 & -0.0164 & 0.0062 & 0.0173 & 0.0140 & -0.0001 \\ -0.0164 & 1.000 & -0.0202 & -0.0161 & -0.0170 & 0.0192 \\ 0.0062 & -0.0202 & 1.000 & 0.0489 & -0.0044 & -0.0171 \\ 0.0173 & -0.0161 & 0.0489 & 1.000 & -0.0134 & 0.0406 \\ 0.0140 & -0.0170 & -0.0044 & -0.0134 & 1.000 & 0.0128 \\ -0.0001 & 0.0192 & -0.0171 & 0.0406 & 0.0128 & 1.000 \end{bmatrix}$$

Correlation Matrix of False Class

$$\begin{bmatrix} 1.000 & 0.0001 & 0.0327 & 0.0337 & 0.0198 & -0.0210 \\ 0.0001 & 1.000 & -0.0178 & -0.0179 & -0.0264 & 0.0230 \\ 0.0327 & -0.0178 & 1.000 & -0.0033 & -0.0110 & 0.0271 \\ 0.0337 & -0.0179 & -0.0033 & 1.000 & 0.0086 & 0.0223 \\ 0.0198 & -0.0264 & -0.0110 & 0.0086 & 1.000 & 0.0229 \\ -0.0210 & 0.0230 & 0.0271 & 0.0223 & 0.0229 & 1.000 \end{bmatrix}$$

By inspecting the covariance matrices, we find that the covariances are relatively small compared to the variances, for both classes. As for the correlation matrices, we find that both classes exhibit weak correlations between features, with most correlation coefficients close to zero.

Since the Naive Bayes model assumes that features are independent, i.e., their covariance is zero, this assumption is fairly reasonable given the weak correlations observed. Moreover, the relatively good performance of the Naive Bayes model compared to the Tied Covariance model suggests that the independence assumption does not significantly degrade performance in this context.

2.1.2 Evaluation of Gaussian Assumptions

For all of the above Gaussian models, a significant assumption is made that the features can be jointly modeled by Gaussian distributions. However, our previous analysis has demonstrated that features 5 and 6 do not conform to Gaussian distributions, as they exhibit multiple peaks. This indicates that the

Gaussian assumption is inaccurate for these features. To understand if the inaccuracy of the assumption affects the model performance, we repeat the classification considering the first 4 features only. We report the results below:

Model	Features	Error	Accuracy
MVG	All 6	0.070	0.930
	4 (removed last 2)	0.080	0.9205
TC	All 6	0.093	0.907
	4 (removed last 2)	0.095	0.905
NB	All 6	0.072	0.928
	4 (removed last 2)	0.077	0.9235

Table 2.2: Comparison of model performance with all features vs. removing last 2 features

We can see that the performance of the TC model remains almost invariable, while that of MVG and NB slightly decrease when removing the last two features. This suggests that despite the inaccuracy in modeling features 5 and 6 with a Gaussian distribution, the Gaussian models were still able to extract some useful information from these features. Removing them resulted in a minor drop in performance. We conclude that the last two features, while not perfectly Gaussian, still contributed valuable insight beneficial for classification.

2.1.3 Feature Impact on Classification Performance

Knowing that features 1 and 2 have means that are similar, but variances that are not, whereas for features 3 and 4 the two classes mainly differ for the feature mean, but show similar variance, we conduct the following experiment:

Features	Model	Error	Accuracy
All features	MVG	0.070	0.930
	TC	0.093	0.907
1 and 2	MVG	0.365	0.635
	TC	0.4945	0.4945
3 and 4	MVG	0.0945	0.9055
	TC	0.0939	0.906

Table 2.3: Comparison of model performance grouped by features

We expect the MVG model to perform better when the features have different variances across classes (like features 1 and 2), as it allows for each feature to have its own covariance matrix for each class, capturing these differences. Conversely, when features have similar variances across classes (like features 3 and 4), the performance gap between MVG and TC may narrow, and TC should outperform MVG. The TC model, which assumes a shared covariance matrix for all classes, can perform adequately in such cases.

In our case, the MVG model demonstrates superior performance over the TC model for features 1-2, with an error less by 13%. This is an expected result and can be attributed to MVG's ability to model different covariance matrices for each class, allowing it to better capture the nuances in data distributions even when the variances are similar. For features 3-4, both models have bery similar performance, but the TC does slighlty outperform the MVG model, with a tiny increase in the accuracy.

However, we note that the error values for the classification using only features1 and 2 are extremely high, and this is due to the nature of the dataset. We have seen in an earlier analysis that features 1 and 2 have a big overlap in their histograms and do not offer class separability.

To conclude, the best generative model we have obtained so far is **MVG, no PCA pre-processing**.

3. Bayes Decisions

In this section, we consider the following applications:

- $(0.5, 1.0, 1.0)$, i.e., uniform prior and costs.
- $(0.9, 1.0, 1.0)$, i.e., the prior probability of a genuine sample is higher (in our application, most users are legit).
- $(0.1, 1.0, 1.0)$, i.e., the prior probability of a fake sample is higher (in our application, most users are impostors).
- $(0.5, 1.0, 9.0)$, i.e., the prior is uniform (same probability of a legit and fake sample), but the cost of accepting a fake image is larger (granting access to an impostor has a higher cost than labeling as impostor a legit user - we aim for strong security). This application can be represented in terms of effective prior as: $(0.22222, 1, 1)$.
- $(0.5, 9.0, 1.0)$, i.e., the prior is uniform (same probability of a legit and fake sample), but the cost of rejecting a legit image is larger (granting access to an impostor has a lower cost than labeling a legit user as impostor - we aim for ease of use for legit users).This application can be represented in terms of effective prior as: $(2, 1, 1)$.

By representing the last two applications in terms of effective prior, we notice that when we aim for strong security, we predict less users as legit, and so the prior decreases. In contrast, when we aim for ease of use for legit users, we predict users as legit more often, and the prior increases. In the following we focus on the applications $(0.1, 1, 1)$, $(0.5, 1, 1)$, and $(0.9, 1, 1)$.

We now perform our classification task using Bayes decisions, meaning we leverages prior knowledge, such as the application prior, and incorporate the costs associated with different types of errors (false positives and false negatives) to make informed and probabilistically sound predictions.

To assess the performance of our models, we employ the Detection Cost Function (DCF). It combines false positive and false negative rates, weighted by their respective costs and class priors, to provide a practical evaluation of the classifier's effectiveness.

3.1 Comparing model performance

To further analyze the goodness of a model, we compare its DCF with its $minDCF$, which represents the lowest possible DCF achievable by optimally adjusting the decision threshold. It serves as a benchmark for the classifier's best performance. The results are reported below.

$\tilde{\pi}$	**Model**	**PCA**	**DCF**	**minDCF**
0.1	MVG	-	0.3051	0.2629
		1	0.3970	0.3686
		2	0.3879	0.3526
		3	0.3879	0.3563
		4	0.3529	0.3012
		5	0.3041	0.2738
		6	0.3041	0.2738
	TC	-	0.4061	0.3628
		1	0.4021	0.3686
		2	0.3960	0.3630
		3	0.4082	0.3681
		4	0.4031	0.3610
		5	0.4051	0.3648
		6	0.4051	0.3648
	NB	-	0.3022	0.2570
		1	0.3970	0.3686
		2	0.3869	0.3562
		3	0.3950	0.3645
		4	0.3970	0.3614
		5	0.3930	0.3545
		6	0.3930	0.3545

Table 3.1: DCF and minDCF for models and PCA values ($\tilde{\pi} = 0.1$)

$\tilde{\pi}$	**Model**	**PCA**	**DCF**	**minDCF**
0.5	MVG	-	0.1399	0.1302
		1	0.1850	0.1769
		2	0.1760	0.1731
		3	0.1759	0.1734
		4	0.1609	0.1537
		5	0.1419	0.1331
		6	0.1419	0.1331
	TC	-	0.1860	0.1812
		1	0.1870	0.1769
		2	0.1850	0.1789
		3	0.1850	0.1830
		4	0.1850	0.1821
		5	0.1860	0.1812
		6	0.1860	0.1812
	NB	-	0.1439	0.1311
		1	0.1850	0.1769
		2	0.1770	0.1710
		3	0.1799	0.1746
		4	0.1770	0.1717
		5	0.1750	0.1737
		6	0.1750	0.1737

Table 3.2: DCF and minDCF for models and PCA values ($\tilde{\pi} = 0.5$)

$\tilde{\pi}$	**Model**	**PCA**	**DCF**	**minDCF**
0.9	MVG	-	0.4001	0.3423
		1	0.4775	0.4342
		2	0.4434	0.4384
		3	0.4680	0.4392
		4	0.4598	0.4150
		5	0.3980	0.3512
		6	0.3980	0.3512
	TC	-	0.4626	0.4421
		1	0.4806	0.4342
		2	0.4785	0.4352
		3	0.4565	0.4342
		4	0.4615	0.4441
		5	0.4626	0.4451
		6	0.4626	0.4451
	NB	-	0.3893	0.3510
		1	0.4775	0.4342
		2	0.4424	0.4323
		3	0.4591	0.4343
		4	0.4630	0.4313
		5	0.4660	0.4340
		6	0.4660	0.4340

Table 3.3: DCF and minDCF for models and PCA values ($\tilde{\pi} = 0.9$)

3.2 Best-performing models

For each application, and for each model, we look for the PCA configuration that provides the best performance of *minDCF* . We report our findings in this table:

$\tilde{\pi}$	**Model**	**PCA**	**DCF**	**minDCF**
0.1	MVG	-	0.3051	0.2629
	NB	**-**	**0.3022**	**0.2570**
	TC	4	0.4031	0.3610
0.5	**MVG**	**-**	**0.1399**	**0.1302**
	NB	-	0.1439	0.1311
	TC	1	0.1870	0.1769
0.9	**MVG**	**-**	**0.4001**	**0.3423**
	NB	-	0.3893	0.3509
	TC	1	0.4806	0.4342

Table 3.4: Best Performing Models (MVG, NB, TC) for Each Application Prior

- For $\tilde{\pi} = 0.1$: Naive Bayes (NB) model without PCA achieved the lowest minimum DCF of 0.2570.
- For $\tilde{\pi} = 0.5$: Multivariate Gaussian (MVG) model without PCA performed the best with a minimum DCF of 0.1302, and NB without PCA performed almost as good, with a minimum DCF of 0.1311.
- For $\tilde{\pi} = 0.9$:Multivariate Gaussian (MVG) model without PCA again showed superior performance with a minimum DCF of 0.3423.

These results indicate that the choice of model varies depending on the application prior, with NB and MVG generally performing well across different scenarios. There is consistency in the relative performance for different applications, NB and MVG consistently outperforming the TC model.

3.3 Calibration

As for calibration, the models show varying degrees of calibration. For prior 0.5, we observe low calibration losses for the models, while for prior 0.1 and 0.9, calibration loss is moderate, suggesting the models are reasonably but not perfectly calibrated. Let us visualize these results:

$\tilde{\pi}$	**Model**	**PCA**	**DCF**	**minDCF**	**Calibration Loss (%)**
0.1	MVG	-	0.3051	0.2629	15.98
	NB	-	0.3022	0.2570	17.56
	TC	4	0.4031	0.3610	11.65
0.5	MVG	-	0.1399	0.1302	7.44
	NB	-	0.1439	0.1311	9.79
	TC	1	0.1870	0.1769	5.73
0.9	MVG	-	0.4001	0.3423	16.88
	NB	-	0.3893	0.3509	10.96
	TC	1	0.4806	0.4342	10.69

Table 3.5: Best Performing Models (MVG, NB, TC) for Each Application Prior with Calibration Loss (%)

For all application priors, the TC model seems to be the most calibrated model. While TC has higher $minDCF$ values than MVG and NB, suggesting slightly inferior discrimination performance in terms of minimizing errors, its superior calibration indicates that when it makes a decision (genuine or fake), the probabilities assigned to those decisions are more reliable. This reliability is crucial in applications where the cost of mis-classification (false positives or false negatives) needs to be balanced with the confidence in the model's predictions. Classification becomes a trade-off problem between accurate classification (low $minDCF$) and reliable confidence in the predictions (low calibration loss).

3.4 Bayes Error Plot

We have determined that our primary application setting is $\tilde{\pi} = 0.1$. Based on our analysis, it appears that the models perform better without applying PCA, so we will proceed with the "no PCA" configuration. Another important approach we will employ to evaluate our models is the Bayes Error Plot. This method allows us to visually assess the performance of different models when varying the application (changing the application prior), by plotting their *actualDCF* and *minDCF* for each application. The results are demonstrated:

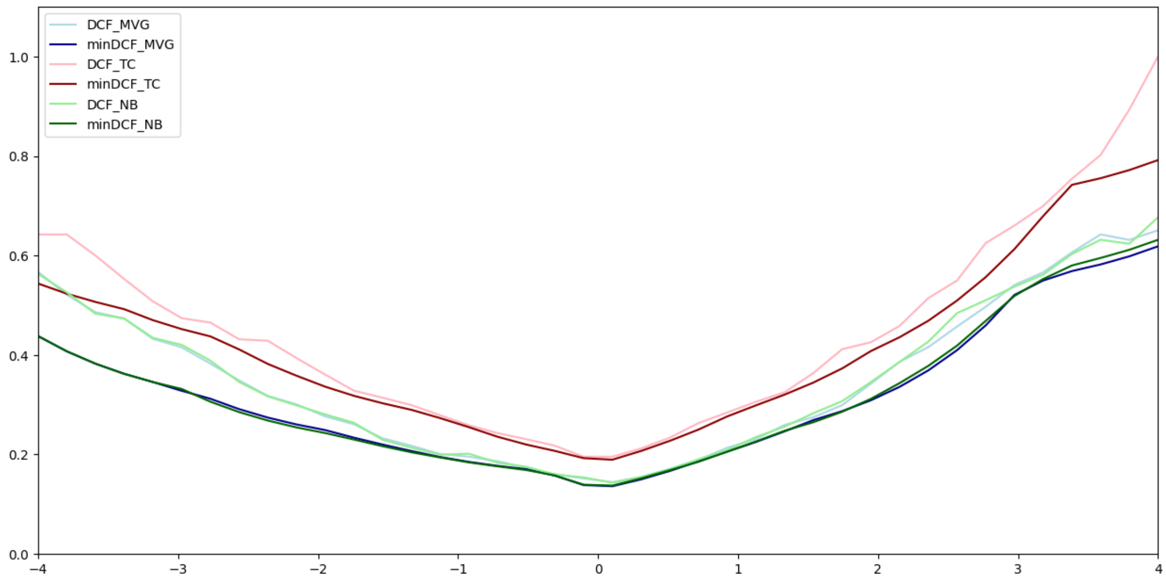


Figure 3.1: Bayes Error Plot of different models

From this figure, we can see that the MVG and NB models are very similar in performance, both their $DCFs$ and $minDFCs$ almost overlapping. We also confirm that model rankings are consistent across applications, with MVG and NB outperforming TC models for all applications in terms of $minDCF$. All considered models have decent calibration across these applications, with the calibration loss reaching its minimum for applications $-1 < \tilde{\pi} < 1$.

4. Logistic Regression

With this approach, rather than modeling the distribution of observed samples like in generative models, we directly model the class posterior distribution. The Logistic Regression model is obtained by minimizing the average cross-entropy between the model predictions and the observed labels.

4.1 Logistic Regression Model on full dataset

In this section, we obtain the log-likelihood scores using the Logistic Regression model varying the values of the regularization hyper-parameter λ. We make a prediction by applying Bayes optimal decisions on the obtained log-likelihood scores for each value of λ, and we compute the DCF and $minDCF$ to assess the performance of the models as a function of λ.

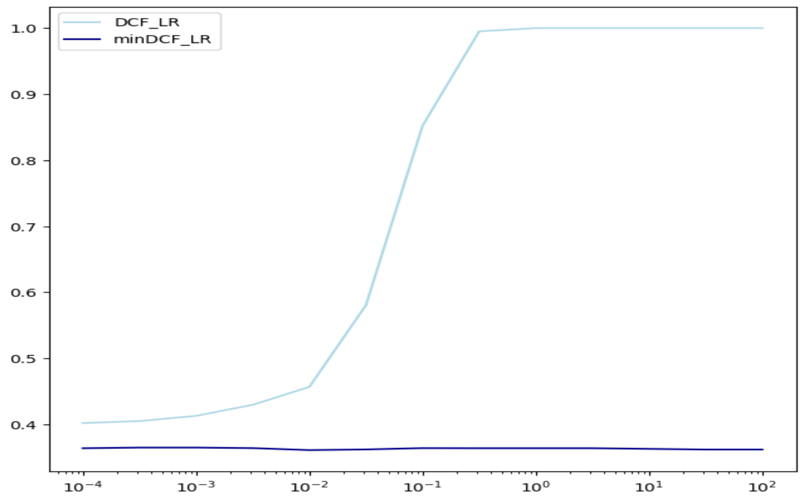


Figure 4.1: Evaluating Logistic Regression models for different values of λ

The observed trends indicate that regularization has a significant impact on the actual DCF but minimal effect on the minimum DCF which remains stable for all values of λ. Specifically, for small values of λ, the actual DCF is low, suggesting that the model is well-calibrated and performs effectively. However, as λ increases, the actual DCF increases dramatically, suggesting that the regularization is counterproductive, leading to a loss of the probabilistic interpretability of the scores. This degradation happens because strong regularization can overly simplify the model and hinder its ability to capture underlying data patterns. So in this case, regularization seems ineffective. In the best case of regularization, we achieve $minDCF = 0.3611$.

4.2 Logistic Regression Model on a subset of dataset

In this section, we conduct the same experiment, but we use only 50 samples from the training dataset.

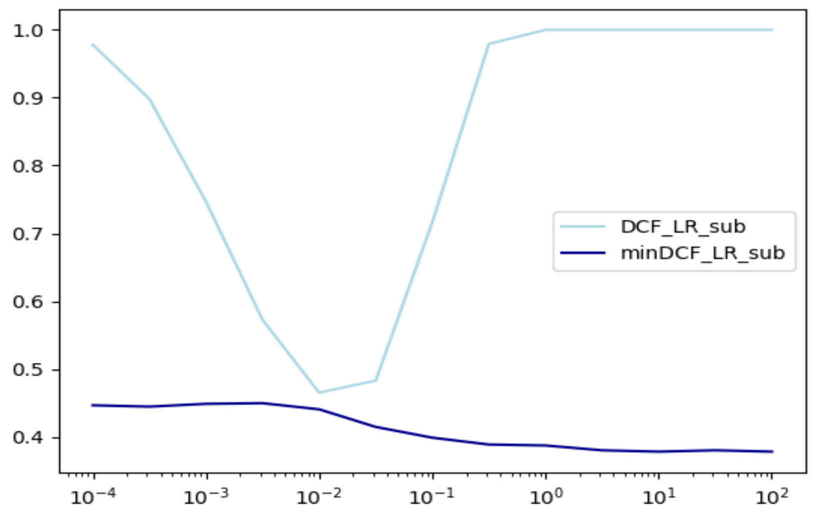


Figure 4.2: Evaluating Logistic Regression models trained on a subset of data for different values of λ

With the reduction in the number of training samples, the impact of regularization on model performance becomes more visible. In particular, for very small values of λ, the model likely overfits the limited training data, leading to a high actual DCF, and poor calibration which is evident in the large gap between the DCF and $minDCF$. As λ increases, regularization helps in mitigating over-fitting by constraining the model parameters. This results in a decrease in the actual DCF, reaching an optimal point around $\lambda = 10^{-2}$. Beyond this optimal point, increasing λ further induces underfitting, where the model becomes too simple to capture the underlying data patterns, causing the actual DCF to rise again. We conclude that with fewer training samples, regularization plays a vital role in preventing overfitting and maintaining model performance. Nevertheless, excessive regularization can lead to underfitting, causing the model to lose its predictive power and probabilistic confidence.

4.3 Prior-weighted Logistics Regression Model

In this section, we train the logistic regression model by taking into consideration the prior of the class genuine from the validation dataset.

The prior-weighted model shows similar trends to the non-prior-weighted model in terms of the impact of regularization on actual and minimum DCF. The prior-weighted model is particularly useful in scenarios where the target prior is significantly different from the empirical prior in the training data, allowing the model to be specifically tuned to the expected distribution of the target data. Since, the performance is similar to the non prior-weighted model, we conclude that for our particular application, prior-weighing is not actually advantageous. In the best case of regularization, we achieve $minDCF = 0.3611$.

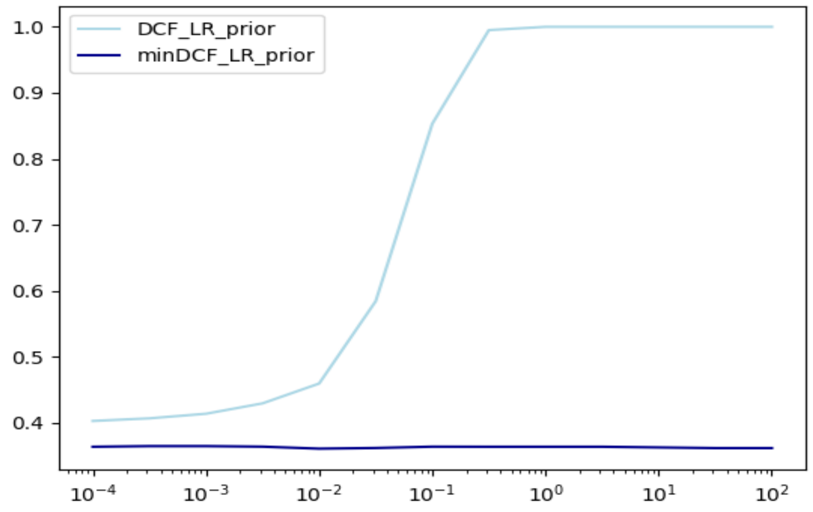


Figure 4.3: Evaluating prior-weighted Logistic Regression models for different values of λ

4.4 Quadratic Logistic Regression

In this part, we expand our features in the quadratic space, and we repeat the experiment.

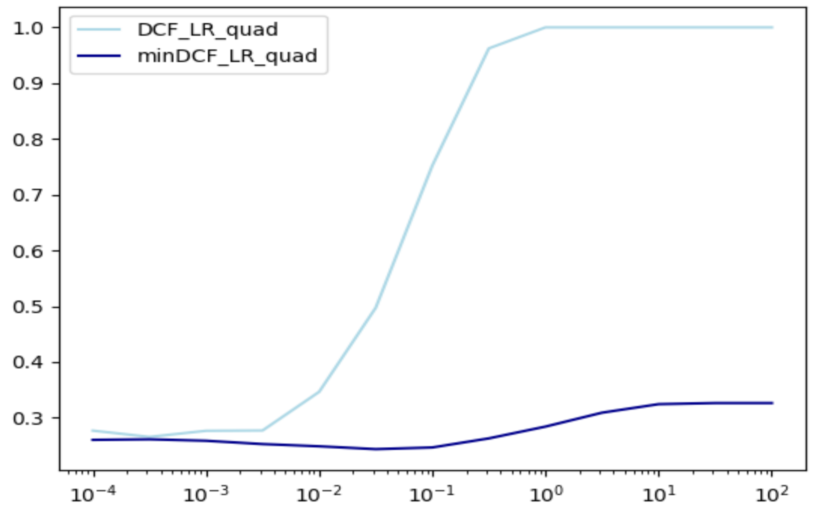


Figure 4.4: Evaluating Quadratic Logistic Regression models for different values of λ

With the quadratic model, both the DCF and $minDCF$ decrease with respect to the linear model (from 0.3611 to 0.243 for the $minDCF$). Moreover, the observed smaller gap between the actual DCF and minDCF for small λ values in the quadratic model compared to the linear model suggests that the quadratic features provide a better fit to the data, enhancing the model's calibration and predictive power when regularization is minimal. However, the slight increase in minDCF for large λ values in the quadratic model indicates that even the best possible performance is affected by excessive regularization. In short, The quadratic logistic regression model demonstrates that regularization can be both beneficial and detrimental, depending on its extent, but it generally fits the data better than a linear regression model. In the best case of regularization, we achieve $minDCF = 0.2436$.

4.5 Pre-processing the training data for Logistic Regression

In this section, we apply data pre-processing. In particular, we center the data, then we apply z-normalization, and we whiten the covariance matrix. Then we apply the linear non-prior-weighted Logistic Regression model. The results are demonstrated below.

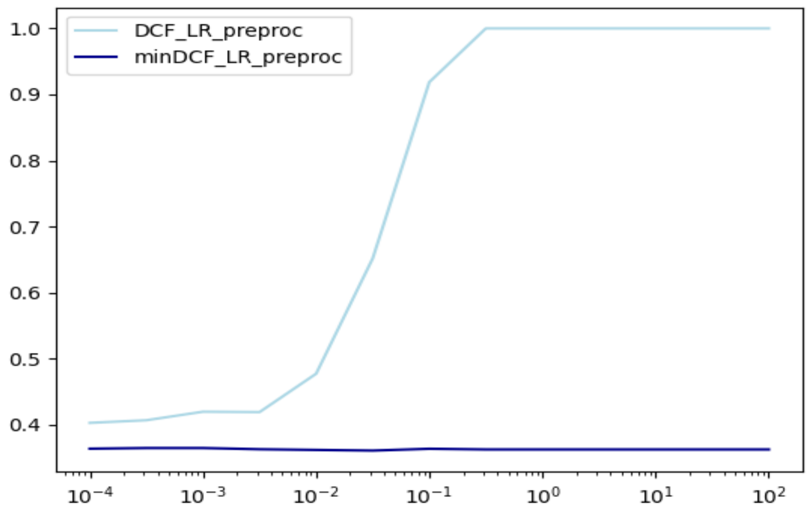


Figure 4.5: Evaluating Logistic Regression models for different values of λ with data pre-processing

The variations with respect to the base-model are minor, almost negligible. This is due to the fact that the original features were already almost standardized. In the best case of regularization, we achieve $minDCF = 0.3611$.

4.6 Model Comparison

Although the best models in terms of minimum DCF are not necessarily those that provide the best actual DCF, we will use the $minDCF$ as a metric to compare model performances, regardless of the calibration. Among the Logistic Regression models, the quadratic model performs by far better than all the other models, reaching a $minDCF$ of 0.2436 at optimal regularization. We now compare the best models for the application with prior $\tilde{\pi} = 0.1$ obtained with every algorithm in the table below:

$\tilde{\pi}$	**Model**	**PCA**	**DCF**	**minDCF**	**Calibration Loss (%)**
0.1	MVG	-	0.3051	0.2629	15.98
	NB	-	0.3022	0.2570	17.56
	TC	4	0.4031	0.3610	11.65
	Quadratic LR($\lambda = 10^{-1.5}$)	-	0.4972	0.2436	104.02

- The MVG (full covariance) model assumes that the data follows a multivariate normal distribution and uses quadratic decision boundaries. It achieves a $minDCF$ of 0.2629.
- The Naive Bayes model assumes independence among features and results in a quadratic separation surface. It achieves a $minDCF$ of 0.2570.
- The Tied Covariance model assumes that the classes share a common covariance matrix but have different means. It uses linear decision boundaries. With PCA reduction to 4 dimensions, it achieves a $minDCF$ of 0.3610.
- The Quadratic LR model uses a quadratic decision boundary which allows for more flexibility in modeling complex relationships in the data. With $\lambda = 010^{-1.5}$, the model achieves the lowest $minDCF$ of 0.2436, making it the best among all models so far.

The **Quadratic Logistic Regression** model with $\lambda = 10^{-1.5}$ provides the lowest $minDCF$ and can be considered the best model for our application so far, despite the fact that it has the worst calibration. Its superior performance can be due to its ability to capture complex, non-linear relationships between features. The other models, while useful, are limited by their assumptions (Gaussian distribution, independence of features, or linearity), leading to higher *minDCF* values. Given that the TC model with a linear separation rule performs the worst, the dataset likely contains non-linear patterns and dependencies that are better captured by more flexible models like Quadratic LR. We choose this model as the best model, the decision metric being only the $minDCF$.

5. Support Vector Machines

Support Vector Machines are linear classifiers that look for maximum margin separation hyperplanes. They are geometric models that provide a geometric interpretation to the regularization term.

5.1 Linear SVM

We first consider a linear SVM model. For this experiment, we set $K = 1$, and we vary the values of C by a logarithmic scale. We compare the obtained $minDCF$ and DCF values with respect to C.

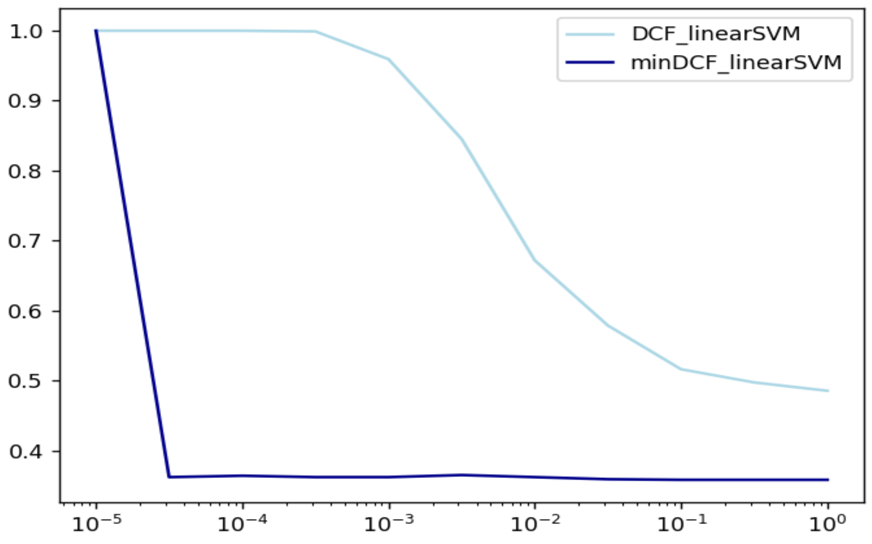


Figure 5.1: Linear SVM models with different C

First, we analyze the regularization impact. The regularization coefficient C has a significant impact on the performance of the linear SVM model within the tested range, since the DCF values show a gradual decrease as C increases. The values of the DCF decrease from 1 to 0.485. The values of the $minDCF$, on the other hand, show a small range of variation (we note that the abrupt decrease in the beginning is due to computation limitations). Thus, weaker regularization, corresponding to higher values of C, leads to better performance, i.e, lower values of the actual DCF. The linear SVM model's performance does indeed depend on regularization.

As for calibration, we find that our DCF and $minDCF$ values are not quite close, especially for lower values of C. This indicates that the model's scores are not well-calibrated when strong regularization is applied. The calibration improves as C increases, with the calibration loss decreasing to 13.0$\%$ at best, indicating better calibration with weaker regularization.

With a $minDCF$ of 0.3581, and a calibration loss of 13$\%$, the linear SVM model cannot compete with previously analyzed linear models, since MVG and NB both outperform it in terms of $minDCF$.

5.2 Linear SVM with centered data

We re-train the same linear SVM model varying the regularization coefficient C, but we use centered data for training. The results are demonstrated below:

The model trained with centered data shows very similar performance to the model trained with non-centered data, with no significant difference worth commenting. Thus, centering the data does not provide any advantage to the linear SVM model.

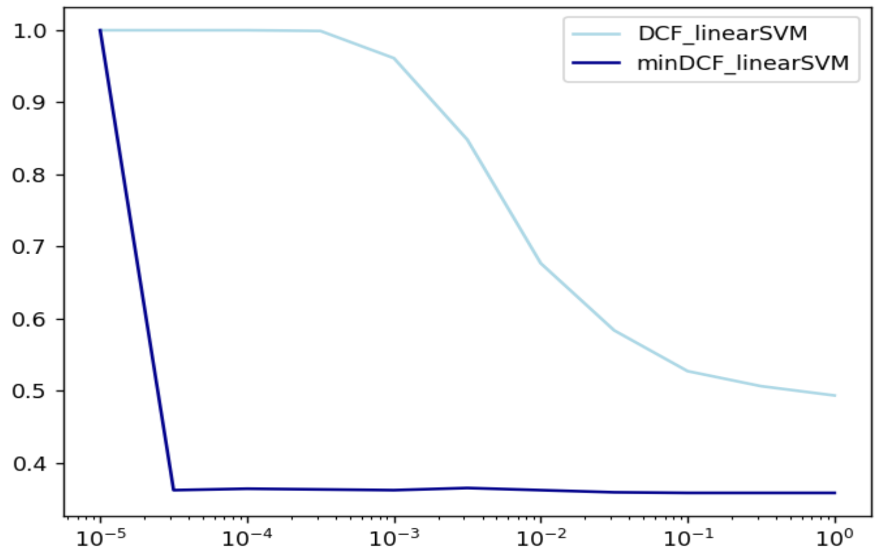


Figure 5.2: Linear SVM models with with different C trained with centered data

5.3 Polynomial Kernel SVM

If we use a kernel function in the derivation of the SVM objective, we compute a linear separation surface in the expanded space, which corresponds to a non–linear separation surface in the original feature space. This allows us to perform a non-linear classification through an implicit expansion of the features in a higher-dimensional space. In this section, we use a polynomial kernel function with $d = 2, c = 1, \epsilon = 0$. We demonstrate the results:

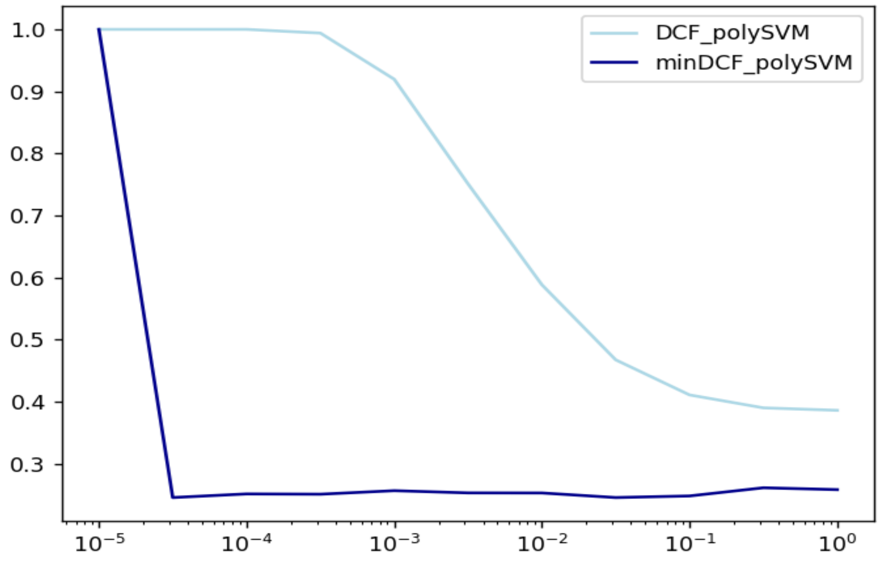


Figure 5.3: Polynomial Kernel SVM models with with different C

The $minDCF$ and DCF show similar trends with respect to C as with the linear SVM model, but they reach lower values. The minimum values reached are $minDCF = 0.245$ and $DCF = 0.386$, indicating that the quadratic SVM is more effective in capturing the underlying patterns of the dataset and its features.

As C increases, both $minDCF$ and DCF generally decrease, indicating better performance with less regularization. This trend suggests that the quadratic SVM benefits from a higher C value, leading to better fitting to the data.

From the point of view of calirbation, this model continues to not have the best-calibrated results, with the calibration loss decreasing as we weaken the regularization, similarly to the linear SVM case.

The quadratic SVM with polynomial kernel $(d = 2, c = 1)$ performs better in terms of both $minDCF$ and DCF compared to the linear models considered, and its performance is quite similar to that of the Quadratic Logistic Regression model in terms of $minDCF$. This reassures us that our classes have better separability with a quadratic surface.

5.4 Radial Basis Function Kernel SVM

We choose another kernel function, the RBF. For this kernel, we need to optimize the parameter γ. Since we have two parameters to optimize, γ and C, we perform a grid search. We set $\epsilon = 1$. For each value of $\gamma \in \{e^{-4}, e^{-3}, e^{-2}, e^{-1}\}$, we train a model for every C in the logarithmic scale. We visualize the results below:

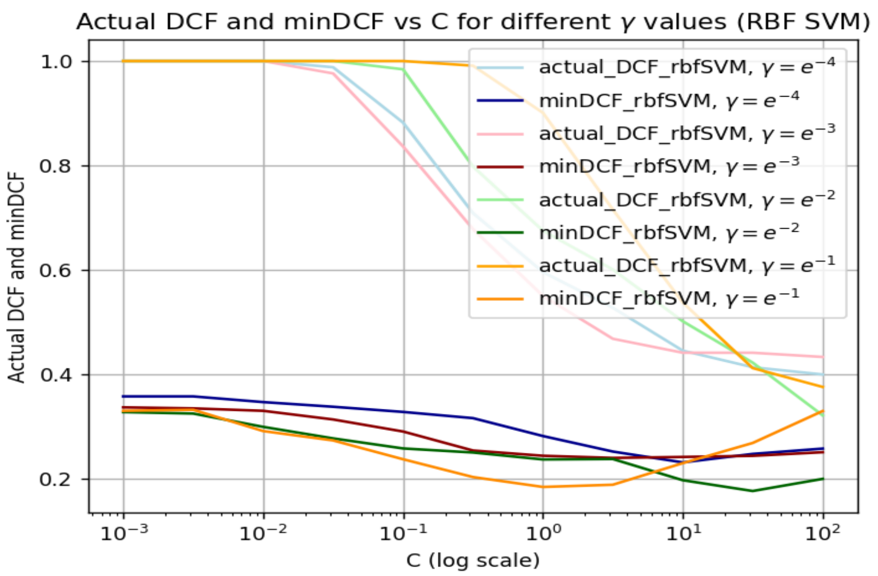


Figure 5.4: RBF Kernel SVM models with with different C and γ

It is directly visible that for larger values of γ, the RBF SVM models perform better because they reach lower values for $minDCF$. In particular, the best performance is observed with $\gamma = e^{-2}$ and C values in the range $10^{1.5} \leq C \leq 10^2$, achieving a $minDCF$ of 0.1755. However, the calibration loss for

RBF SVM models is quite high, although the models seem to be more calibrated for higher values of C, i.e, less regularization. This indicates that while this model captures underlying patterns in the data effectively (low $minDCF$), it requires further calibration for practical application (to reduce the gap between DCF and $minDCF$).

In terms of $minDCF$, the RBF kernel SVM significantly outperforms other models such as linear and polynomial SVMs, as well as LR and MVG, demonstrating its capability to model complex, non-linear relationships in the dataset. This suggests that the dataset benefits from the non-linear decision boundaries provided by the RBF kernel, highlighting once again the non-linear nature of the relationships between features and classes.

6. Gaussian Mixture Models

6.1 Full and Diagonal-Covariance GMM models

When the features of a dataset cannot be adequately described by a single Gaussian distribution, they may be more effectively represented by a combination of multiple Gaussian distributions. In such cases, we model the data using a weighted sum of these Gaussian distributions, known as a Gaussian Mixture Model (GMM). In this section, we will determine the optimal number of Gaussian components for each class, genuine and fake. We will start by training GMMs with full covariance matrices, which capture relationships between all pairs of features. We will also train GMMs with diagonal covariance matrices, which assume that features are uncorrelated. The results are shown below:

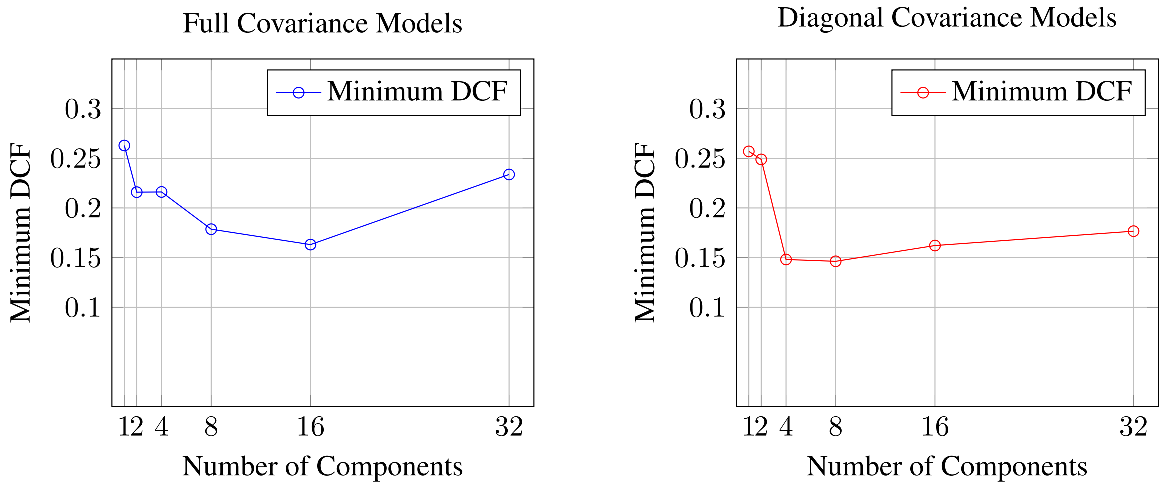


Figure 6.1: Minimum DCF vs Number of Components for Full and Diagonal Covariance Models

From the figures, we can see that for full covariance models, the performance improves ($minDCF$ decreases) as the number of components increases up to 16 components, after which the performance deteriorates with 32 components. On the other side, for diagonal covariance models, the performance improves significantly up to 8 components, after which there is a slight increase in $minDCF$ at 16 and 32 components. For full covariance models, the optimal number of components is 16, providing a $minDCF$ of 0.1613, while for diagonal covariance models, the optimal number of components is 8, with a $minDCF$ of 0.1462.

We thus note that the diagonal covariance models perform significantly better than the full-covariance ones, which aligns with the properties of our features that we discovered earlier. In particular, since our features are not correlated, it makes sense that the diagonal models perform better, as they assume feature independence.

However, it is somewhat surprising that the full covariance model with 32 components performs worse than the one with 16 components. This might be because the increased model complexity does not add value due to the independence of features, leading to over-fitting. **From here, we can choose the GMM with diagonal covariances and 8 components as the best performing GMM model.**

6.2 Best Models Comparison

We now compare the best performances we obtained with LR models, SVM models, and GMM models, omitting MVG models from this comparison because we have already established that they perform worse. From this table, it seems that the GMM model outperforms LR and SVM, providing the lowest $minDCF$. We now proceed to plot the Bayes Error to evaluate performance over wide range of applications. The results are demonstrated below:

Table 6.1: Performance of Different Models with Varying Parameters

$\tilde{\pi}$	**Model**	**PCA**	**DCF**	**minDCF**	**Calibration Loss (%)**
0.1	Quadratic LR($\lambda = 10^{-1.5}$)	-	0.4972	0.2436	104.02
	RBF kernel SVM ($\gamma = e^{-2}, C = 10^{1.5}$)	-	0.4216	0.1755	140.12
	GMM (Diagonal Covariance, c=8)	-	0.1808	0.1462	23.65

From this Bayes Error plot, we confirm that the ranking of the models across the range of application priors (-4,4) is consistent with the ranking established in the table above for our application prior, $\pi = 0.1$. In particular, GMM remains the best performing model, followed by the RBF-kernel SVM model, and finally by the Quadratic LR model, in terms of $minDCF$ of course.

Moreover, the GMM model stands out because it provides the lowest $minDCF$, but also the best calibration among the three models, having a minimal gap between the actual DCF and $minDCF$ across all the range of considered application priors. The RBF-kernel SVM model on the other side has the worst calibration, and is indeed very harmful for applications with priors in the range outside [-1,1], as the mis-calibration is quite significant outside this interval. Finally, the Quadratic LR model has a slightly higher $minDCF$, and non-optimal calibration . It can be also considerably harmful for appli-

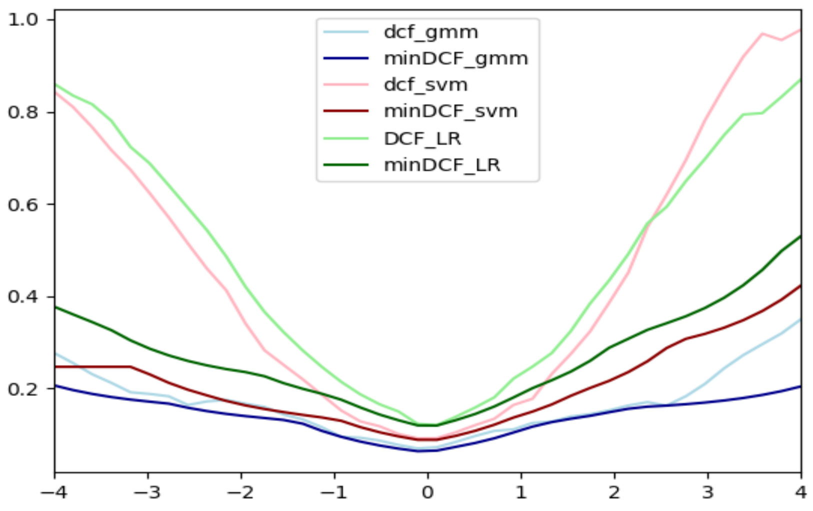


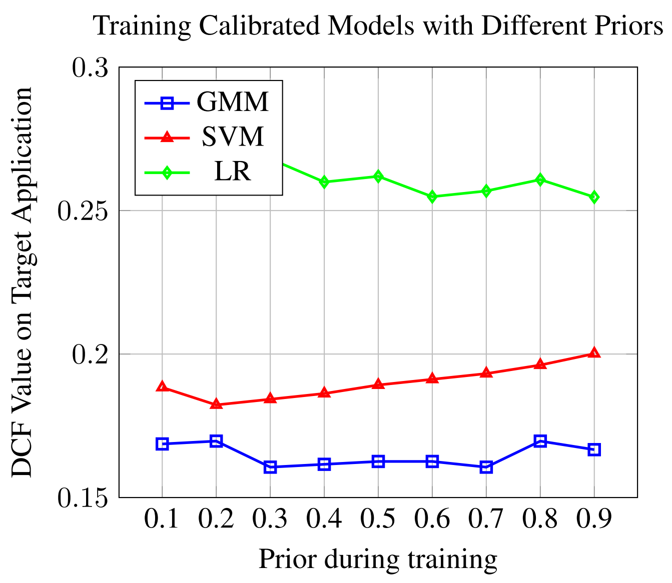
Figure 6.2: Performance of best LR, SVM, and GMM models over a range of application priors

cation priors outside [-2,2].

7. Calibration

7.1 K-Fold Approach

To close the gap between $actualDCF$ and $minDCF$, we calibrate the scores using a K-Fold approach. We proceed as follows: for each model in table 6.1, we compute a calibration transformation trained on the validation set. During training, we **shuffle** the scores to ensure that the data is randomly distributed across different folds, which aligns with the assumption that the data points are independent and identically distributed. We experiment training with the priors in the set [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]. Each of the obtained models is evaluated on the target application prior 0.1 in terms of *actual DCF*. Then, the training prior providing the smallest $actualDCF$ is chosen as the best calibrated model. The final model is obtained by training again a calibration transformation using the entire validation set, and using the training prior chosen by means of K-Fold.



From the above graph, we choose the best calibrated transformations as follows:

- GMM: calibration model trained with $prior = 0.3$. Evaluated on target application, $actualDCF = 0.16058$, $minDCF = 0.1462$
- SVM: calibration model trained with $prior = 0.2$. Evaluated on target application, $actualDCF = 0.18226$, $minDCF = 0.1794$
- LR: calibration model trained with $prior = 0.9$. Evaluated on target application, $actualDCF = 0.25468$, $minDCF = 0.2538$

The final calibrated models for the 3 approaches are computed by training again a calibration transformation with the above chosen training priors, on the entire validation set.

We summarize the new best results:

Table 7.1: Performance of Different Models after Calibrating the Scores

$\tilde{\pi}$	**Calibrated Model**	**PCA**	**DCF**	**minDCF**	**Calibration Loss (%)**
0.1	Quadratic LR($\lambda = 10^{-1.5}$)	-	0.2456	0.24363	0.81
	RBF kernel SVM ($\gamma = e^{-2}, C = 10^{1.5}$)	-	0.18325	0.17546	4.43
	GMM (Diagonal Covariance, c=8)	-	0.16058	0.14626	9.79

We observe considerable improvements in performance when comparing Table 6.1 and Table 7.2. In particular, the calibration loss is much lower than it was before calibration, because thr $actualDCF$ has decreased considerably for all models.

Apart from the target application, we also evaluate the performance of the new models with calibrated scores over a range of applications, to see the effect of calibration over a wider range of priors. The results are demonstrated in the below Bayes Error Plot:

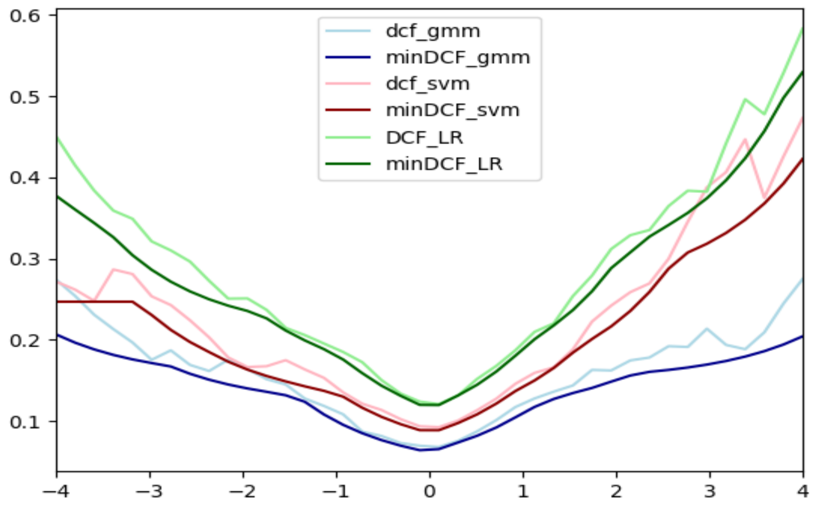


Figure 7.1: Performance of best LR, SVM, and GMM models over a range of application priors after calibration

The new Bayes Error Plot shows a much smaller gap between the $actualDCF$ and $minDCF$ than in Figure 6.2. This shows that calibration has indeed been effective, and there is no harm in using any model for any target application anymore. All 3 models are well-calibrated more or less over the entire range of applications.

7.2 Score-level Fusion

In this section, we fuse the scores of all the three models in one feature matrix, and use these fused scores to compute a calibration transformation. As in the previous section, we experiment with different training priors, and we set $prior = 0.2$ as it provides the smallest $actualDCF$ for our target application of $prior = 0.1$. The final fused and calibrated model with the calibration trained over the entire validation set with $prior = 0.2$, provides $actualDCF = 0.16058$, $minDCF = 0.14541$. We add this result to the table of final models:

Table 7.2: Performance of Different Models after Calibrating the Scores

$\tilde{\pi}$	**Calibrated Model**	**PCA**	**DCF**	**minDCF**	**Calibration Loss (%)**
0.1	Quadratic LR($\lambda = 10^{-1.5}$)	-	0.25468	0.24660	3.27
	RBF kernel SVM ($\gamma = e^{-2}, C = 10^{1.5}$)	-	0.18226	0.17433	4.54
	GMM (Diagonal Covariance, c=8)	-	0.16058	0.14220	12.93
	Score Fusion	-	0.16058	0.14541	10.43

The score fusion performs well. It provides an $actualDCF$ lower than that of the LR and SVM models, and equal to that of the GMM model. This model also has a very small calibration loss of 10.43%, so the fused scores have a decent calibration. Overall, using a model based on score-level fusion would be a reasonable choice since its performance is very competitive and sufficient. This model combines the strengths of the different models to enhance the performance.

However, as the final chosen and delivered system, I would opt to use the GMM model with diagonal covariance and $c = 8$, with calibrated scores, the calibration transformation trained on $prior = 0.3$. I have chosen this model because it provides the smallest possible $actualDCF$ for our target application, is well-enough calibrated, and would be the best fit to capture the variability of the features of our dataset. The Score Fusion model is also a great candidate, but its results in comparison with the GMM model do not justify the difference in computation cost between them, so we choose the computationally lighter model.

8. Evaluation

We now move to evaluating our final chosen and delivered model on the evaluation dataset of our project, and comparing it to the performance of other variants and other models, to verify whether our choice has indeed been optimal.

8.1 Performance of the Chosen Model

Our chosen model, GMM with diagonal covariance, $c = 8$, and scores calibrated using $prior = 0.3$ for training, results in an $actualDCF = 0.2124$, and a $minDCF = 0.2025$ for the target application of prior = 0.1. To measure its performance on a wider range of applications, we consider the below Bayes Error Plot:

We can immediately see that the scores of our chosen model are well calibrated for our target application, and for other applications as well. This is visible from the very small, sometimes barely visible, gap between the $actualDCF$ and $minDCF$ in the Bayes Error Plot.

8.2 Evaluating other Best Models

We now proceed to evaluating the performance of the other candidate best models on the evaluation set, by calculating the corresponding $actualDCF$ and $minDCF$:

From the above table, it seems that the Score Fusion model would have been a more optimal choice, as it provides the lowest $actualDCF$, lowest $minDCF$, and even lowest calibration loss. It by far outperforms all the other models.

Now let's consider the DCF Error Plots for these 4 models obtained by varying the decision threshold

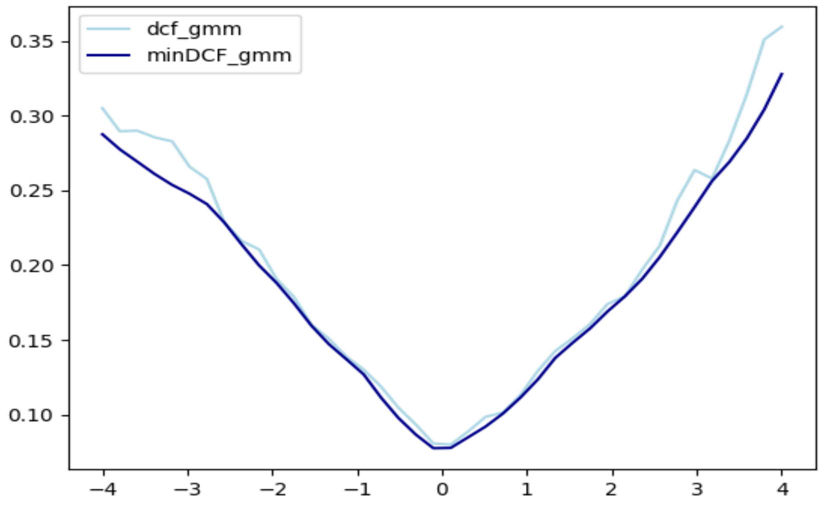


Figure 8.1: Bayes Error Plot of the Delivered System

Table 8.1: Performance of Different Models on the Evaluation Set

$\tilde{\pi}$	**Calibrated Model**	**PCA**	**DCF**	**minDCF**	**Calibration Loss (%)**
0.1	Quadratic LR ($\lambda = 10^{-1.5}$)	-	0.36587	0.35147	4.10
	RBF kernel SVM ($\gamma = e^{-2}, C = 10^{1.5}$)	-	0.26586	0.26224	1.38
	GMM (Diagonal Covariance, c=8)	-	0.2124	0.2025	4.88
	Score Fusion	-	0.18951	0.18783	0.89

and thus the false positive rates and false negative rates:

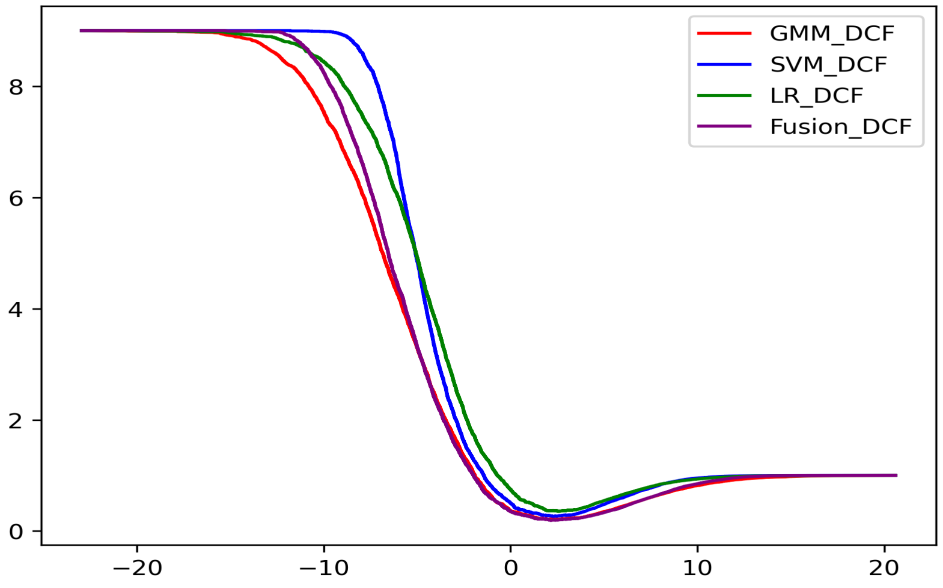


Figure 8.2: DCF Error Plot of the Different Models

The DCF error plot reveals that the GMM model consistently shows the lowest values across a range of score thresholds, indicating it has the lowest detection costs. This means that the GMM model effectively balances false positives and false negatives, making it a strong candidate when the goal is minimizing errors. On the other hand, the Fusion model performs nearly as well as the GMM, with its DCF values very close to those of the GMM model. The Fusion model combines the strengths of multiple models, offering robustness and flexibility across different operating conditions. This makes the Fusion model also a viable choice, especially since we have seen in Table 8.1 that it produces the best results.

Finally, we also compare the performance of the models across a range of applications. We visualize this using the below Bayes Error Plot:

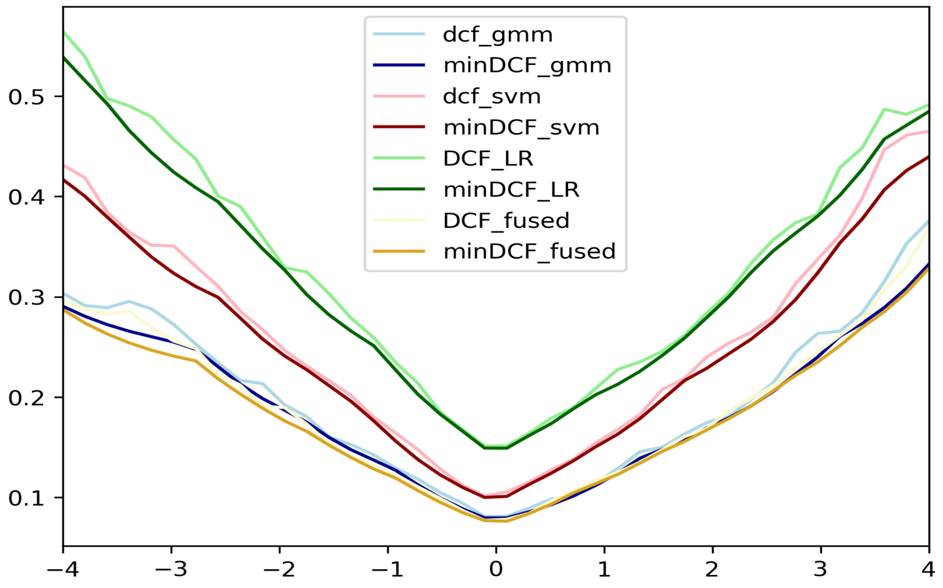


Figure 8.3: Bayes Error Plot of the Different Models

First, we note that the calibration strategy was effective for all models, as the gap between the $actualDCF$ and $minDCF$ for each model is minimal. This means that all models perform well, and there is no harm in using any of them for any application. Second, from the Bayes Error Plot, we can see that although the GMM and Fused models have very close values for the cost functions, the Fused model does indeed outperform all other models, including the GMM model. It has the lowest values of $actualDCF$ and $minDCF$ across the entire range of applications, and it would be the optimal choice.

8.3 Evaluating various RBF-kernel SVM models

In this last section, we compare the different RBF-kernel SVM models with different hyper-parameters, by computing their $minDCF$ on the evaluation dataset. In particular, we will perform a grid-search by varying γ and C, and plotting the $minDCF$. We suffice with $minDCF$ for comparison, since comparing the $actualDCF$ would require calibrating the scores of all these models, which requires excessive training time and computation power. This will allow us to verify whether our best chosen RBF-kernel SVM model with $\gamma = e^{-2}, C = 10^{1.5}$, was indeed the best choice, or whether other hyper-parameter choices perform better on the evaluation dataset. The results are shown in the below graph:

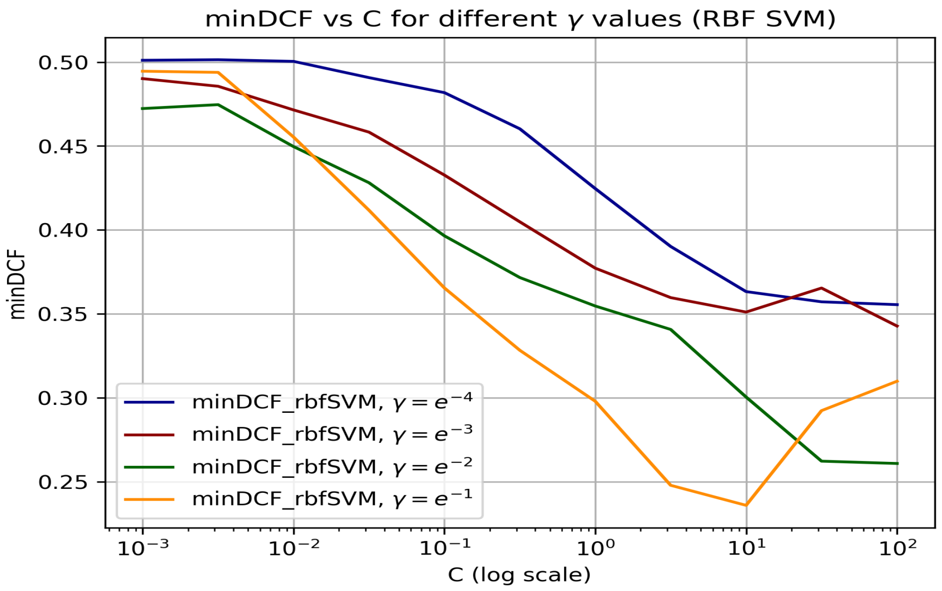


Figure 8.4: $minDCF$ of various RBF-SVM models on the evaluation dataset

From the above plot, the lowest value is achieved with $\gamma = e^{-1}$, and $C = 10$. This means our chosen RBF-SVM model is not the best-performing model in terms of $minDCF$ on the evaluation set.

Table 8.2: Performance Comparison

$\tilde{\pi}$	**Model**	γ	C	$minDCF$
0.1	Our chosen model	e^{-2}	$10^{1.5}$	0.262236
	Best-performing model	e^{-1}	10	0.235914

9. Conclusion

In this project, we evaluated various classification models with different covariance structures and decision boundaries, specifically focusing on full, tied, and diagonal covariances, as well as linear and quadratic separation surfaces. Through comprehensive experiments, we observed that models employing either diagonal or full covariances, combined with quadratic decision boundaries, consistently delivered superior classification performance on our dataset.

This outcome suggests that the underlying data distribution and feature interactions are better captured by models that allow for more flexibility in representing the covariance structure of the data. The diagonal and full covariance models, with their ability to account for varying degrees of feature correlations, combined with quadratic decision boundaries that can model complex relationships, appear to align well with the characteristics of our dataset.

Given these findings, we conclude that the two primary approaches for classification of our dataset are Gaussian Mixture Models (GMMs) and Score Fusion Models.

Gaussian Mixture Models (GMMs) are well-suited to model complex, multi-modal distributions which allows for effective classification, particularly when using models with diagonal or full covariances that reflect the intricate relationships between features.

Score Fusion Models, by combining predictions from multiple classifiers, can leverage the strengths of various individual models and enhance classification performance through a more robust and comprehensive approach.

Overall, our findings highlight the importance of selecting appropriate covariance structures and decision boundaries based on the specific characteristics of the dataset. The flexibility and adaptability of GMMs, coupled with the strategic use of score fusion techniques, offer promising pathways for achieving high classification accuracy for the Fingerprint Spoofing problem.

The code for this project can be found at: GitHub Repository