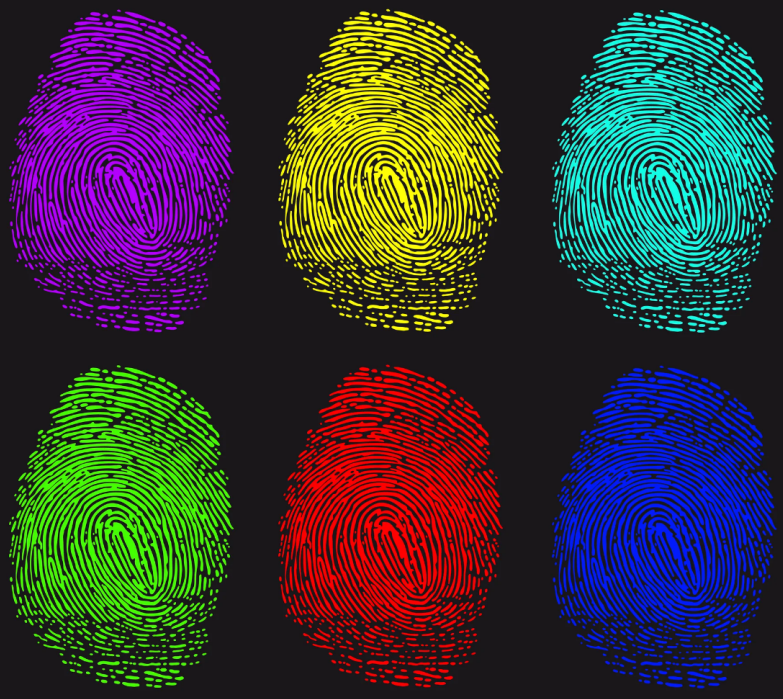
Federico Mustich

Machine Learning and Pattern Recognition

Fingerprint Spoofing Detection

Project Report

Academic Year 2022/2023



Politecnico di Torino

Contents

1 Abstract **1**

2 Introduction **2**
2.1 Dataset Considerations . . . 2
2.2 Feature Analysis . . . 3
2.2.1 Feature Correlation . . . 3
2.2.2 Feature Distribution . . . 5

3 Classifiers **7**
3.1 Nearest Neighbors Classifier . . . 8
3.1.1 K-Nearest Neighbors Classifier . . . 8
3.1.2 Weighted KNN Classifier . . . 9
3.1.3 Considerations . . . 10
3.2 Multivariate Gaussian Classifiers . . . 11
3.2.1 Full Covariance . . . 11
3.2.2 Diagonal Covariance . . . 12
3.2.3 Full Tied Covariance . . . 13
3.2.4 Diagonal Tied Covariance . . . 14
3.2.5 Considerations . . . 14
3.3 Gaussian Mixture Models . . . 16
3.3.1 Full Covariance . . . 17
3.3.2 Diagonal Covariance . . . 18
3.3.3 Full Tied Covariance . . . 19
3.3.4 Considerations . . . 19
3.4 Logistic Regression . . . 22
3.4.1 Linear Logistic Regression . . . 22
3.4.2 Quadratic Logistic Regression . . . 24
3.4.3 Considerations . . . 26
3.5 Support Vector Machines . . . 27
3.5.1 Linear SVM . . . 27
3.5.2 Non-Linear SVMs . . . 29
3.5.2.1 Polynomial Kernel SVM . . . 29
3.5.2.2 Radial Basis Function Kernel SVM . . . 30
3.5.3 Considerations . . . 30
3.6 Best Classifiers . . . 31

4 Score Calibration **32**

5 Evaluation **34**
5.1 Target Application . . . 34
5.2 Different Applications . . . 36
5.3 Final Evaluations . . . 37

6 Conclusions **38**

1 Abstract

This project assesses the efficacy of various machine learning supervised algorithms in distinguishing between genuine and spoofed fingerprint images. Specifically, it focuses on detecting fingerprints that have been maliciously replicated, posing a potential security threat. A diverse range of classification techniques will be employed and their performance compared.

First, a quick analysis of the dataset properties and distribution has been conducted. Results hint at how a decent separation between the two classes present in the dataset should be realistically achievable.

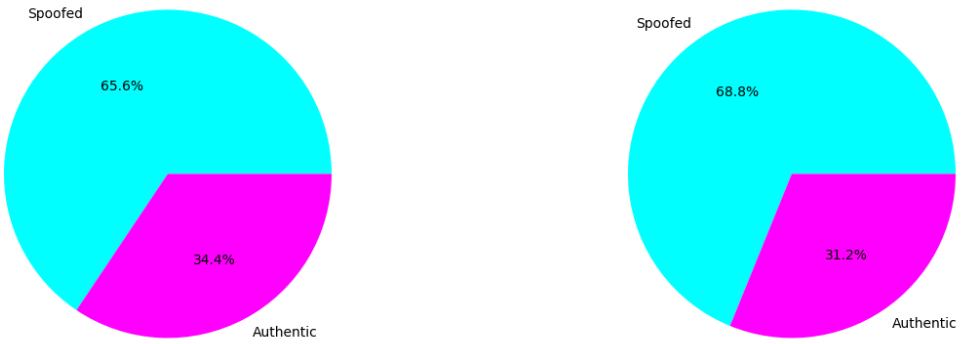
The evaluation begins with the **K-Nearest Neighbor Classifier**, a simple , non-parametric and computationally efficient method which does not employ any training process. This approach leverages the inherent similarity between data points to assign new fingerprints to their appropriate class, relying on the knowledge of their "nearest neighbors" within the training dataset.
Next, the investigation delves into the realm of generative classifiers. The **Multivariate Gaussian Classifiers** and **Gaussian Mixture Models** will be utilized, each constructing probabilistic models of both authentic and spoofed fingerprints. These models assess the likelihood of a new fingerprint belonging to either category, ultimately issuing a classification verdict.
The focus then shifts towards discriminative classifiers. The **Logistic Regression** model employs linear functions to separate genuine and spoofed fingerprints, while the **Support Vector Machines** seeks the optimal hyperplane that maximizes the margin between the two classes. These models directly learn separation rules from the labeled training data, identifying the subtle patterns that differentiate real from fabricated fingerprints.

All models have been trained on the training set, possibly performing cross-validation for hyperparameter estimation, and their performance is then evaluated on the test set, a collection of unseen fingerprints, where their true classification capabilities are exposed.
Results show how the analyzed dataset can be successfully classified by the previously said machine learning techniques.

2 Introduction

2.1 Dataset Considerations

The project utilizes a dataset comprised of synthetic fingerprint images. These images are represented by 10-dimensional embeddings, effectively continuous-valued vectors with no direct physical interpretation. The dataset has been already pre-divided into two distinct sets: a training set containing 2,325 samples and a test set with 7,704 samples. It is noteworthy that the dataset exhibits class imbalance, with the spoofed fingerprint class significantly outnumbering the genuine class at a ratio of about 2:1. While the spoofed fingerprints are categorized into six sub-classes representing distinct spoofing methods from which they have been generated, the specific sub-class label corresponding to each spoofed sample remains unavailable.



(a) Training set

(b) Test set

Figure 1: *Pie charts showing the relative ratios for the two classes represented in the dataset.*

2.2 Feature Analysis

2.2.1 Feature Correlation

We start by analyzing the properties of the dataset by assessing the **feature correlation** between each of the 10 embedding values with all the others. We do this by means of the Pearson Correlation coefficient:

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

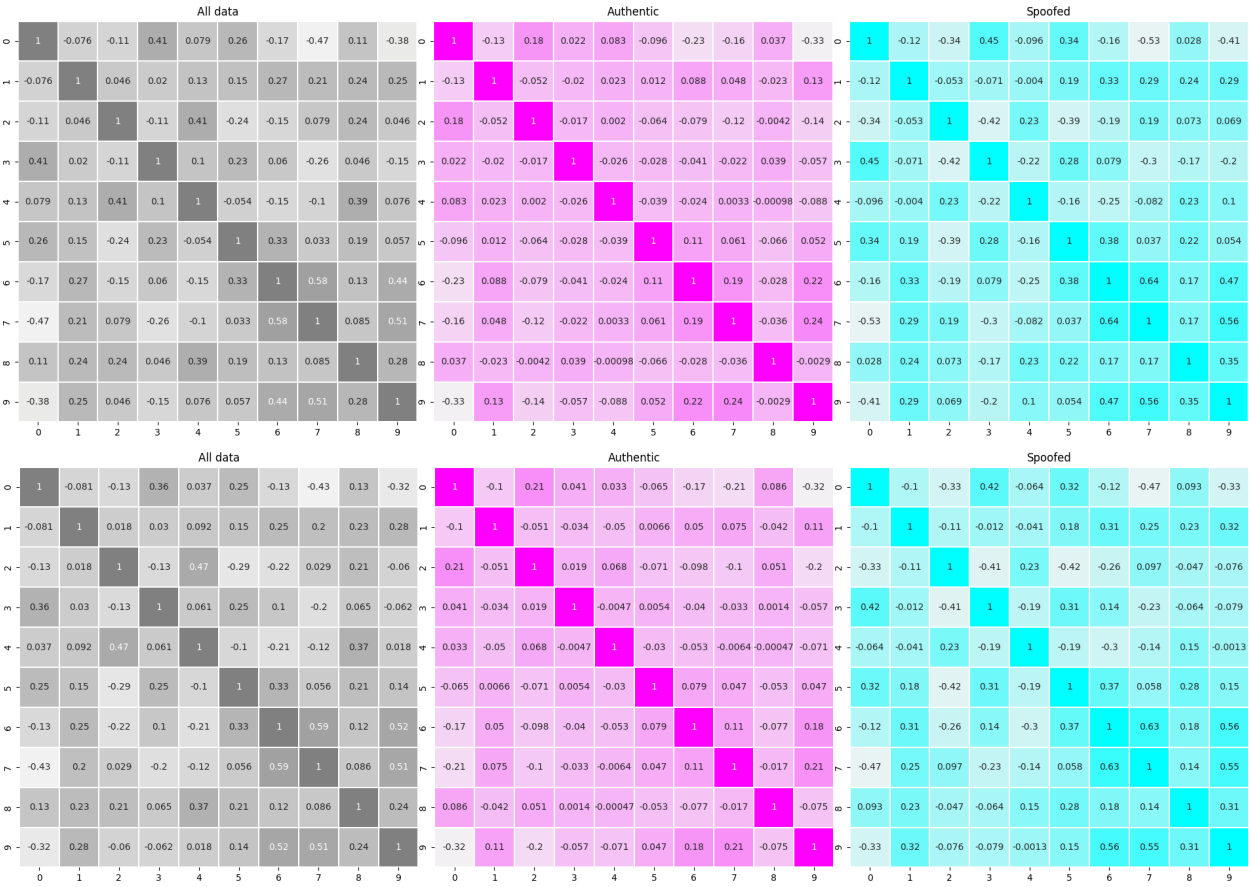


Figure 2: *Correlation heatmaps between the features. Left: all data. Center: authentic data. Right: spoofed data*

Our analysis of the feature correlation heatmaps reveals some insight with potential implications for improving classification performance. While the **authentic class exhibits no remarkable inter-feature relationships**, the spoofed class paints a different picture. Here, **feature 6 appears to be positively correlated with both features 7 and 9**. Additionally, a **subtle negative correlation might exist between features 0 and 7**. This observed disparity in correlation patterns between the classes suggests that dimensionality reduction techniques could yield fruitful results.

By reducing the data dimensionality, we may achieve several desirable outcomes. Firstly, the presence of correlated features, potentially introducing redundancy and hindering model generalization, could be effectively addressed. Secondly, focusing on the most informative features through dimensionality reduction

could streamline the classification process, leading to potentially improved efficiency and accuracy.

Therefore, as we proceed we will explore dimensionality reduction techniques such as **Principal Component Analysis (PCA)** or **Linear Discriminant Analysis (LDA)** and their effects on the classifiers' performances. LDA in general is not suggested to be applied to binary classifications problems, so we expect to observe worse results when it is applied. We will check if this expectations stand to the experimental results.

If fact, from the below plot we can immediately appreciate how much information (or variance) we can retain from the original data by applying PCA with different values of principal components.

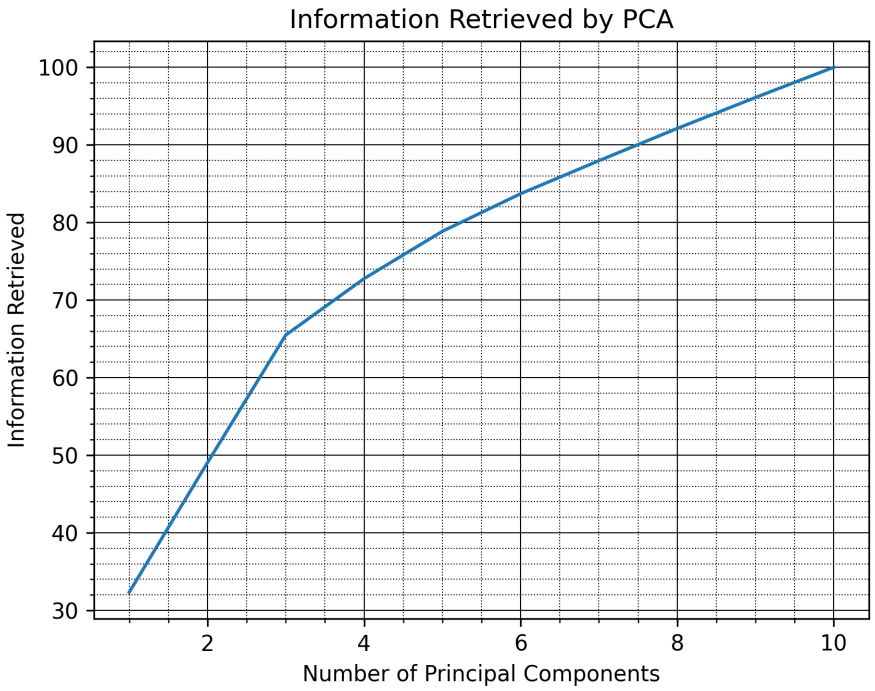


Figure 3: *Plot showing the percentage of variance preserved by applying PCA.*

In particular, we notice how applying PCA with $m = 7$ allow us to preserve almost 90% of the original information content.

2.2.2 Feature Distribution

We now plot the distributions of each one of the 10 embedded features, evidencing the diverse class distributions, and look for noticeable patterns:

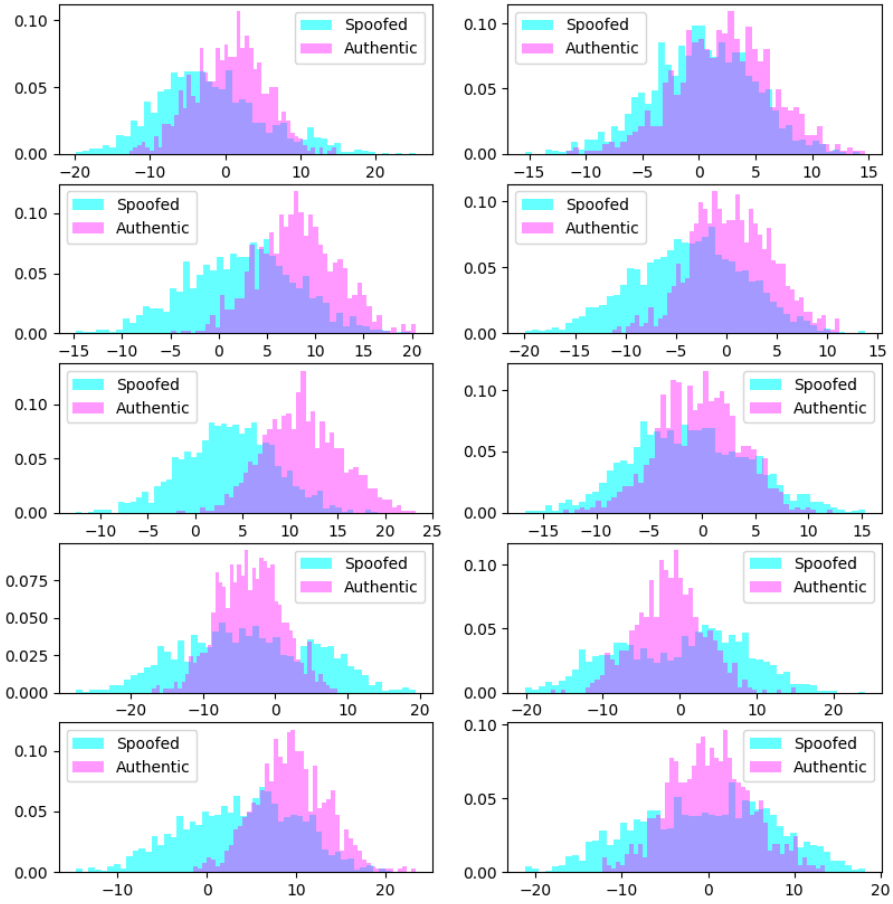


Figure 4: *Distribution of the embedded features.*

Most of the features, particularly those associated with the authentic class, exhibit well-behaved Gaussian distributions. This aligns with prior expectations and suggests a relatively well-structured data representation. However, within the spoofed class, notable deviations emerge: features 6 and 7 stand out by displaying a peculiar "double bell" structure. This pattern, while unable to be definitively confirmed solely through distribution plots, is possibly due the distinct methodologies employed in spoofing fingerprint generation.

Moving beyond individual features, the degree of overlap between the two classes across features varies considerably. Feature 1 showcases nearly identical distributions for both classes, indicating limited discrim-

inatory power. Conversely, feature 4 demonstrates commendable performance in separating authentic and spoofed fingerprints. Features 2, 3, 6, 7, 8, and 9 present a more nuanced picture. While exhibiting significant overlap, they also reveal distinct distribution patterns.

Since our features do not have a physical interpretation but are instead embeddings probably generated by some automatic algorithm, showing already similar scales and means, **we decide not to proceed with standardization**. This could, however, result in some issue when applying dimensionality reduction techniques.

Finally, we show the cross-feature scatter plots for each the 10 features over the remaining ones:

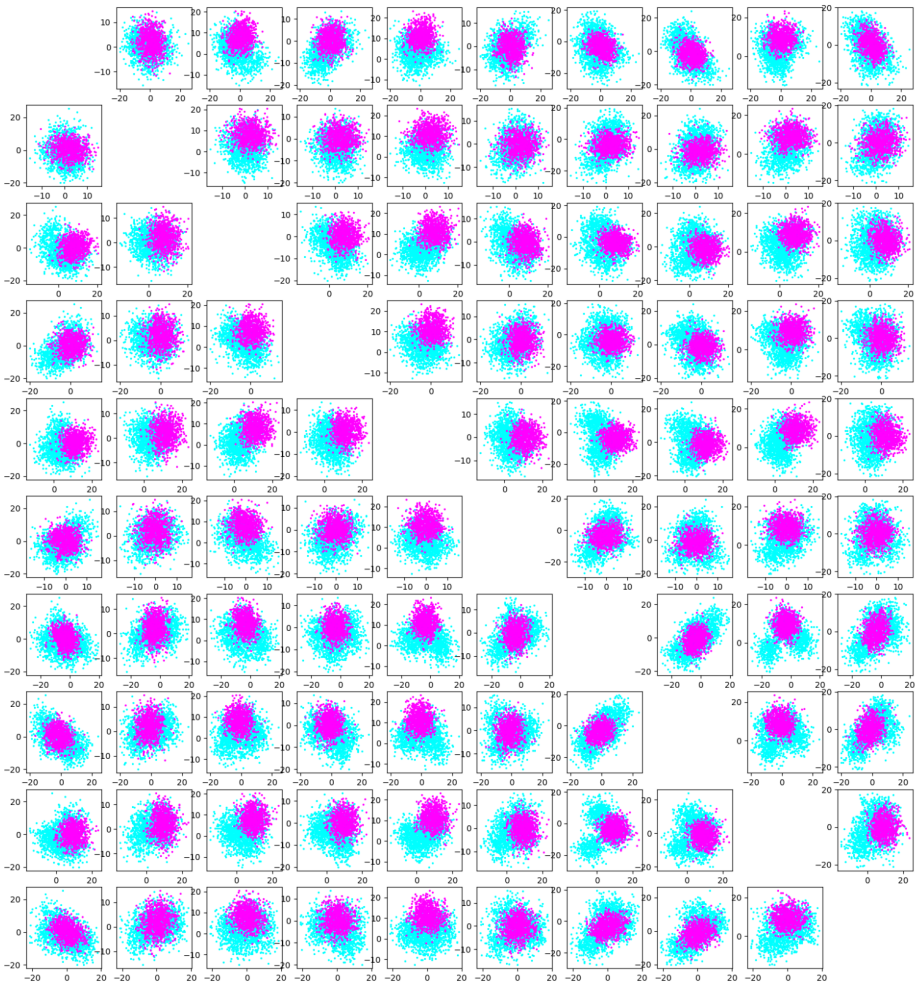


Figure 5: *Cross-feature scatter plots.*

In many of the scatter plots the authentic class sampled are organized in clusters which make it somewhat straight-forward to visually separate them from the spoofed class samples. We can also notice how also in this case there are some noticeable case in which the spoofed class looks organized in different "components". This finding aligns with our earlier observations regarding potential heterogeneity within the spoofed class, hinting at the possible use of different methods for fingerprint spoofing.

3 Classifiers

We now start evaluating and comparing the performance of various classifiers on the spoofed fingerprint authentication task. The primary metric employed is the **minimum Detection Cost Function (minDCF)**, which incorporates the relative costs associated with False Negative (FN) and False Positive (FP) errors. Additionally, **error rate** is assessed through cross-validation to provide a complementary perspective on model performance.

Although interrelated, these metrics evaluate classifier efficacy from distinct viewpoints. Error rate simply calculates the percentage of correctly classified samples within the predicted set. In contrast, minDCF accounts for the varying costs attributed to FN and FP classifications.

The minimum DCF is usually computed by selecting an optimal operating point for a binary classifier by balancing the trade-off between false negatives and false positives, defined as:

$$DCF = \pi_T * C_{Fn} * FNR + (1 - \pi_T) * C_{Fp} * FPR$$

where:

- π_T is the prior probability of class True
- C_{Fn} and C_{Fp} are the costs for False Negative and False Positive Errors
- FNR is the False Negative Rate, computed as $FN/(FN + TP)$ via a confusion matrix
- FPR is the False Positive Rate, computed as $FP/(FP + TN)$ via a confusion matrix

The idea is to compute this value over varying values of thresholds against which to compare the scores and perform class predictions. The predictions are then organized in a confusion matrix and the DCF value is computed. The minDCF is the value corresponding to the threshold which minimizes the DCF.

In the context of fingerprint authentication, accurately distinguishing genuine and spoofed fingerprints is paramount. False positives carry a significantly higher consequence compared to false negatives. While a user requiring re-authentication (potentially employing an alternative identification method) may be considered acceptable, allowing an unauthorized individual with fabricated credentials to bypass security measures poses a grave risk. Consequently, while more abstract, **minDCF serves as the primary performance metri**c. Notably, a higher error rate solely consisting of false negatives might be preferable to a lower error rate accompanied by a substantial number of false positives. The **error rate metric will be utilized as a tie-breake**r should any model with equal minDCF emerge.

In this phase, **we will consider a target application where prior probabilities of authentic and spoofed classes are the same while the false positive cost is tenfold** compared to the false negative one, defined by the triplet $(\pi_T = 0.5, C_{Fn} = 1, C_{Fp} = 10)$.

3.1 Nearest Neighbors Classifier

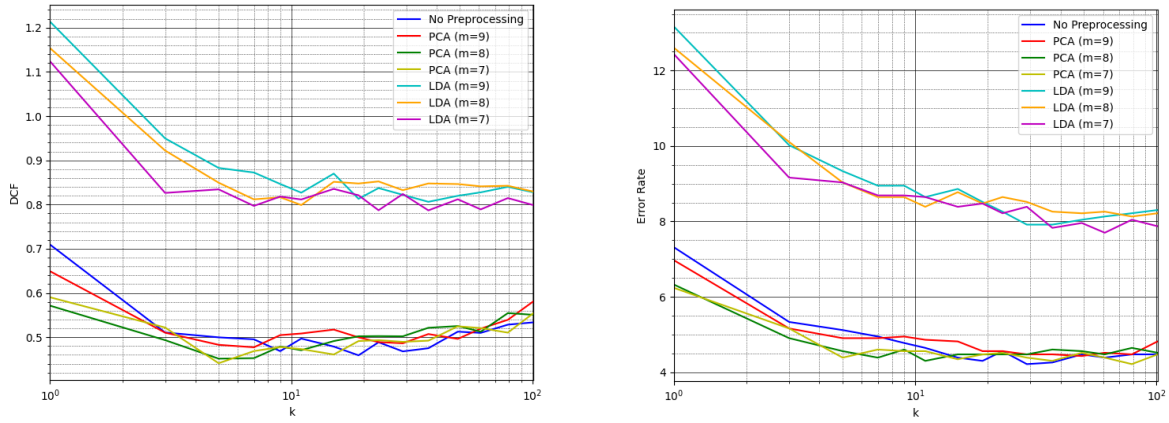
3.1.1 K-Nearest Neighbors Classifier

Our first considered classifier is the **K-Nearest Neighbors Classifier (KNN)**. KNN is a very simple, non-parametric, classifier which does not need any training process. Instead, without making any assumption on the underlying data distribution, it naively assigns to the new samples to be predicted the class most common among its k nearest neighbors (computing the Euclidean distance) in the n-dimensional space sampled from the "training" data, although no explicit training step occurs.

Even if very simple, as the amount of data approaches infinity, the two-class KNN algorithm is guaranteed to yield an error rate no worse than twice the Bayes error rate (the minimum achievable error rate given the distribution of the data)[1].

However, the KNN classifier in non-probabilistic, as it only outputs hard predictions (e.g. it outputs directly binary labels and not scores). Consequently, **it is not possible to measure its performance through minDCF**, since there are no scores to be compared against a threshold. The decision threshold is essentially fixed, which means the trade-off between false positives and false negatives is also fixed. Luckily, we can still evaluate the cost of the predictions using the DCF formula.

The KNN classifier relies on one hyperparameter to be tuned, k, which expresses the number of neighbours to be considered when assigning a sample to a class. In order to select a proper value for k we will proceed with **6-folds cross-validation** on the training dataset, and compare the results on the raw dataset and after having applied PCA and LDA.



(a) Detection Cost

(b) Error Rate

Figure 6: *Results for the K-Nearest Neighbor Classifier on the training set.*

[1] T. Cover and P. Hart, "Nearest neighbor pattern classification," in IEEE Transactions on Information Theory, vol. 13, no. 1, pp. 21-27, January 1967, doi: 10.1109/TIT.1967.1053964.

Model	DCF	Error Rate
KNN (No Preprocessing), k = 19	0.459	**4.30%**
KNN (PCA m=9), $k = 7$	0.477	4.90%
KNN (PCA m=8), $k = 7$	0.451	4.56%
KNN (PCA m=7), $k = 5$	**0.441**	4.39%
KNN (LDA m=9), $k = 37$	0.806	7.91%
KNN (LDA m=8), $k = 11$	0.799	8.39%
KNN (LDA m=7), $k = 37$	0.787	7.83%

Table 1: *DCF and Error Rate results for the best k values for the Nearest Neighbors Classifier.*

It is immediate to notice how LDA is detrimental to the classifier's performance, as opposed to PCA which allows to reduce the dimensionality of the data while preserving overall the same performances observed on the the raw dataset. It is also worth noting the decent overlap between the two metrics. This indicates how, with the considered classifier, reducing the error on the prediction corresponds to a decrease on the detection cost, regardless of the unbalanced classes and different error costs. We will check if this pattern will also hold on the following parametric classifiers.

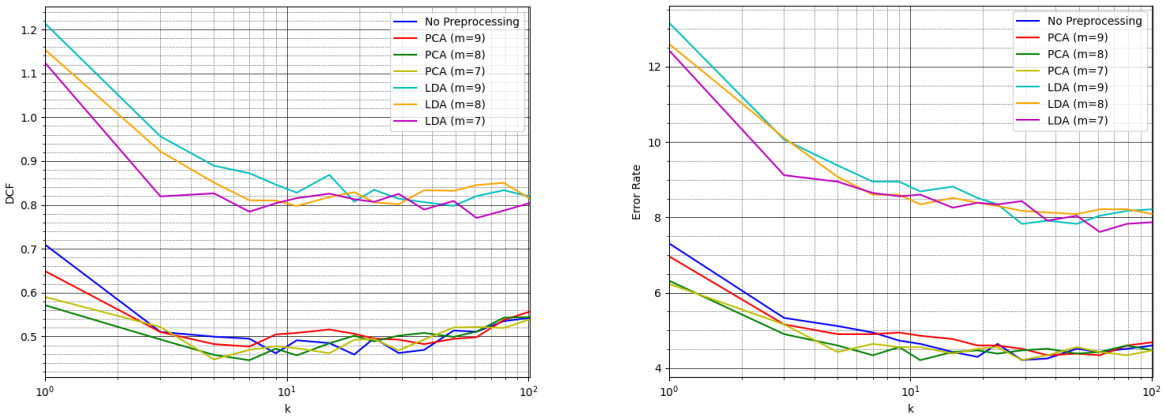
If we were to be particularly precise we could actually point out a small trend in the DCF plot to increase as the number of selected k-neighbors increases after having touched a minimum in the region of $[5-7]$, at least for PCA processed data, where instead the error rate seem to saturate and remain constant. This could be caused by the unbalanced classes: being the spoofed class more represented in the dataset, by increasing the number of considered neighbors it is more likely to have a majority of spoofed samples participating in the classification of a new sample.

3.1.2 Weighted KNN Classifier

Usually, a challenge associated with basic "majority voting" classification in KNN classifiers arises from class imbalance. When one class significantly outnumbers others, its representatives tend to dominate the nearest neighbor pool of a new data point, potentially biasing the prediction towards the majority class.

To mitigate this effect, a common approach involves **Weighted KNN** classification. This method assigns weights to each neighbor based on its distance to the test point. Closer neighbors, considered more informative, receive higher weights, effectively amplifying their influence on the final classification. Typically, the weight function takes the form of an inverse distance metric, scaling down the impact of distant neighbors. By incorporating distance-based weighting, weighted KNN effectively compensates for skewed class distributions, leading to more balanced and accurate classifications.

For completeness, we also experimented with a Weighted KNN classifier, with the same procedure seen previously.



(a) Detection Cost (b) Error Rate

Figure 7: *Results for the Weighted K-Nearest Neighbor Classifier on the training set.*

Model	DCF	Error Rate
Weighted KNN (No Preprocessing), k = 29	0.462	**4.22%**
Weighted KNN (PCA m=9), k = 7	0.477	4.90%
Weighted KNN (PCA m=8), k = 7	**0.446**	4.34%
Weighted KNN (PCA m=7), k = 5	0.448	4.43%
Weighted KNN (LDA m=9), k = 49	0.798	7.83%
Weighted KNN (LDA m=8), k = 11	0.798	8.34%
Weighted KNN (LDA m=7), k = 61	0.770	7.61%

Table 2: *DCF and Error Rate results for the best k values for the Weighted Nearest Neighbors Classifier.*

Quite surprisingly, the results show an almost identical performance with respect to the non-weighted KNN classifier. We could hypothesize that this unexpected result is due to the two classes being already decently separated in distinguished clusters, maybe also as a consequence of the different spoofing methodologies utilized. However, we must not fall into the "hindsight bias" trap and we should avoid too much *a posteriori* reasoning.

3.1.3 Considerations

Model	DCF	Error Rate
KNN (PCA m=7), k = 5	**0.441**	4.39%
Weighted KNN (PCA m=8), k = 7	0.446	**4.34%**

Table 3: *Best KNN classifiers.*

The two KNN classifiers considered show a very similar performance. The best model achieves a minDCF of **0.441**. We will consider the Nearest Neighbors classifiers as a benchmark to compare their results with the next, more sophisticated, parametric models.

3.2 Multivariate Gaussian Classifiers

We now start considering **Multivariate Gaussian Classifiers (MVG)**. MVGs is a family of supervised classifiers which, from a general point of view, work by assuming that the data is distributed following a gaussian distribution of mean $\mu_{\mathbf{c}}$ and covariance matrix $\mathbf{\Sigma_c}$ for each class c, or in other words, that the data, given the class, can be described as a gaussian distribution like:

$$(\boldsymbol{X}|C=c) \sim \mathcal{N}(\boldsymbol{\mu_c}, \mathbf{\Sigma_c})$$

which means that, if we knew $\boldsymbol{\mu_c}, \mathbf{\Sigma_c}$, the likelihood for a given sample $\boldsymbol{x_t}$ to belong to a class c could be mapped to a certain value depending on $\boldsymbol{x_t}$ itself and on $\boldsymbol{\mu_c}$ and $\mathbf{\Sigma_c}$, and we could compute it as:

$$f(\boldsymbol{x_t}|c) = \mathcal{N}(\boldsymbol{\mu_c}, \mathbf{\Sigma_c})$$

However, we do not know *a priori* the values for $\boldsymbol{\mu_c}$ and $\mathbf{\Sigma_c}$, which become the model parameters $\boldsymbol{\theta} = [(\boldsymbol{\mu_1}, \mathbf{\Sigma_1}), ..., (\boldsymbol{\mu_k}, \mathbf{\Sigma_k})]$, with $k = 2$ for binary problems, to be learned during the training phase by means of an estimator which maximizes the data log-likelihood (Maximum Likelihood estimator):

$$\boldsymbol{\theta}^*_{ML} = \underset{\theta}{\operatorname{argmax}}\, \mathcal{L}(\boldsymbol{\theta})$$

We can regard the ML solution as the parameters that best explains the observed dataset (i.e. the value for which it is most likely that we observe the dataset samples).

Depending on how they treat the class covariance matrix/matrices, we can distinguish four types of Gaussian Classifiers, which we considered.

These models do not have any hyperparameter that needs to be tuned. We proceeded evaluating them by applying 6-fold cross-validation on the training dataset.

3.2.1 Full Covariance

The **Full Covariance Gaussian Classifier** is the one that assumes the least amount of hypothesis on the data. It allows classes not only to have different means µc but also makes no assumption on the covariance matrices $\mathbf{\Sigma_c}$ of each class. As a result, it should be able to capture correlations in data and it should output better results, at the cost of increased computational costs.

The ML solution, which gives our estimated parameters, is given by:

$$\boldsymbol{\mu_c^*} = \frac{1}{N_c} \sum_{i|c_i=c} \boldsymbol{x_i}$$

$$\mathbf{\Sigma_c^*} = \frac{1}{N_c} \sum_{i|c_i=c} (\boldsymbol{x_i} - \boldsymbol{\mu_c^*})(\boldsymbol{x_i} - \boldsymbol{\mu_c^*})^T$$

It is worth noting that the Full Covariance classifier has a **quadratic** decision function.

Model	minDCF	Error Rate
MVG Full Covariance (No Preprocessing)	0.294	**5.51%**
MVG Full Covariance (PCA m=9)	0.292	5.68%
MVG Full Covariance (PCA m=8)	**0.288**	5.68%
MVG Full Covariance (PCA m=7)	0.294	**5.51%**
MVG Full Covariance (LDA m=9)	0.408	8.95%
MVG Full Covariance (LDA m=8)	0.414	8.60%
MVG Full Covariance (LDA m=7)	0.428	8.65%

Table 4: *DCF and Error Rate results for the Full Covariance Gaussian Classifier*

3.2.2 Diagonal Covariance

The **Diagonal Covariance Gaussian Classifier** is also called Naive Bayes Classifier because it relies on the Naive Bayes assumption: **it supposes that features are independently distributed and also uncorrelated**. Since the off-diagonal terms of $\boldsymbol{\Sigma_c}$ represent the covariances of the different components of the data feature space, such features will lead to a covariance matrix $\boldsymbol{\Sigma_c}$ which is diagonal, having all the elements off-diagonal equal to 0. Based on what we discovered in the Feature Correlation section (Figure 2), we expect this assumption to hold true for our authentic fingerprint data, while it could be too strong for the spoofed class. For these reasons, we expect this model to perform a bit worse when compared to the Full Covariance one.

The ML solution is given by:

$$\boldsymbol{\mu_c^*} = \frac{1}{N_c} \sum_{i|c_i=c} \boldsymbol{x_i}$$

$$\sigma^2_{c,[j]} = \frac{1}{N_c} \sum_{i|c_i=c} (x_{i,[j]} - \mu_{c,[j]})^2$$

with $\boldsymbol{\Sigma_c^*}$ equal to a diagonal square matrix with the $\sigma^2_{c,[1]}, ..., \sigma^2_{c,[D]}$ elements on its diagonal.

The Diagonal Covariance classifier is yet again a **quadratic** classifier.

Model	minDCF	Error Rate
MVG Diagonal Covariance (No Preprocessing)	0.421	7.23%
MVG Diagonal Covariance (PCA m=9)	0.349	5.98%
MVG Diagonal Covariance (PCA m=8)	**0.345**	5.98%
MVG Diagonal Covariance (PCA m=7)	0.352	**5.94%**
MVG Diagonal Covariance (LDA m=9)	0.439	8.99%
MVG Diagonal Covariance (LDA m=8)	0.449	8.99%
MVG Diagonal Covariance (LDA m=7)	0.445	9.03%

Table 5: *DCF and Error Rate results for the Diagonal Covariance Gaussian Classifier*

3.2.3 Full Tied Covariance

The **Full Tied Covariance Gaussian Classifier** still assumes for each class to have its own mean $\boldsymbol{\mu_c}$ but it is different from the previous classifiers because it assumes a *tied* covariance matrix, e.g. the covariance matrix $\boldsymbol{\Sigma}$ is the same for both the classes.

$$f(\boldsymbol{x_t}|c) = \mathcal{N}(\boldsymbol{\mu_c}, \boldsymbol{\Sigma})$$

The selected covariance matrix is full and not diagonal. **This models consequently assumes that, even if the samples have been generated by two different gaussian distributions, these distributions differ only by the mean.** Considering the scatter plots in the Feature Distribution section (Figure 5), we expect this assumption to not hold true and this model to show a lower performance with respect to the non-tied models.

The ML solution is given by:

$$\boldsymbol{\mu_c^*} = \frac{1}{N_c} \sum_{i|c_i=c} \boldsymbol{x_i}$$

$$\boldsymbol{\Sigma^*} = \frac{1}{N} \sum_{c} \sum_{i|c_i=c} (\boldsymbol{x_i} - \boldsymbol{\mu_c^*})(\boldsymbol{x_i} - \boldsymbol{\mu_c^*})^T$$

This model has a **linear** decision function, so it will be interesting to assess how linear models compare with respect to quadratic ones.

Model	minDCF	Error Rate
MVG Full Tied Covariance (No Preprocessing)	0.450	**8.90%**
MVG Full Tied Covariance (PCA m=9)	0.445	9.03%
MVG Full Tied Covariance (PCA m=8)	0.444	8.95%
MVG Full Tied Covariance (PCA m=7)	**0.442**	9.08%
MVG Full Tied Covariance (LDA m=9)	0.449	8.95%
MVG Full Tied Covariance (LDA m=8)	0.448	**8.90%**
MVG Full Tied Covariance (LDA m=7)	0.445	8.95%

Table 6: *DCF and Error Rate results for the Full Tied Covariance Gaussian Classifier*

3.2.4 Diagonal Tied Covariance

The last type of gaussian classifier considered is the **Diagonal Tied Covariance Gaussian Classifier**. This model assumes both that the features are independently distributed and uncorrelated (diagonal covariance) and that the distributions generating the samples have same covariance and only differ by the mean. These are the strongest assumptions posing the most constraints on the classifier, so we do not expect this model to perform significantly good.

The ML solution is given by:

$$\boldsymbol{\mu_c^*} = \frac{1}{N_c} \sum_{i|c_i=c} \boldsymbol{x_i}$$

$$\sigma^2_{[j]} = \frac{1}{N} \sum_{c} \sum_{i|c_i=c} (x_{i,[j]} - \mu_{c,[j]})^2$$

with $\boldsymbol{\Sigma^*}$ equal to a diagonal square matrix with the $\sigma^2_{[1]}, ..., \sigma^2_{[D]}$ elements on its diagonal.

This model also is **linear**.

Model	minDCF	Error Rate
MVG Diagonal Tied Covariance (No Preprocessing)	0.514	9.33%
MVG Diagonal Tied Covariance (PCA m=9)	0.515	9.76%
MVG Diagonal Tied Covariance (PCA m=8)	0.513	9.76%
MVG Diagonal Tied Covariance (PCA m=7)	0.509	9.76%
MVG Diagonal Tied Covariance (LDA m=9)	0.444	**8.77%**
MVG Diagonal Tied Covariance (LDA m=8)	0.445	**8.77%**
MVG Diagonal Tied Covariance (LDA m=7)	**0.442**	**8.77%**

Table 7: *DCF and Error Rate results for the Diagonal Tied Covariance Gaussian Classifier*

3.2.5 Considerations

Model	minDCF	Error Rate
MVG Full Covariance (PCA m=8)	**0.288**	**5.68%**
MVG Diagonal Covariance (PCA m=8)	0.345	5.98%
MVG Full Tied Covariance (PCA m=7)	0.442	9.08%
MVG Diagonal Tied Covariance (LDA m=7)	0.442	8.77%

Table 8: *Best MVG classifiers.*

Plotting the results of our reference metrics for all the Gaussian Classifiers on an histogram plot we obtain Figure 8. At a first glance it looks like, same as for KNN classifiers, LDA in general degrades the performances of the classifiers. PCA pre-processed data instead preserves similar performances to the unprocessed data over the whole range of dimensionality tested, with the notable exception of the Diagonal Covariance model, which seems to particularly appreciate PCA pre-preprocessing, with more than 20% performance gain on the minDCF primary metric.

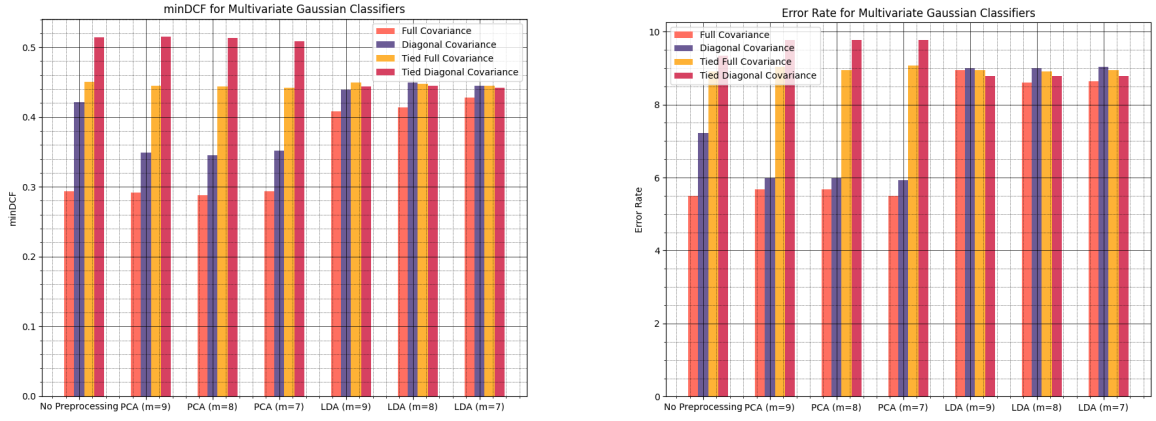
Quadratic models generally perform much better than the linear ones, and **the Full Covariance classifier results to be best classifier** in all the combinations considered. but could be worth mentioning how, even if in general sub-optimal, linear models perform about the same as (or sometimes slighty better than) quadratic ones when data is pre-processed with LDA. This could be explained by the fact that LDA

reduces the dimensionality of the data by maximizing the between-class covariance over the within-class covariance to better separate classes. A linear model could have an easier time at correctly classify samples with such pre-processed data.

Anyway, the performance on both the metrics of LDA-fed quadratic models is significantly impaired with respect to when no pre-processing or PCA is applied, while the Tied Full Covariance model perform consistently regardless of the preprocessing method applied or not applied and, notably, the Tied Diagonal Covariance model improves a bit its, not great, performance when LDA is applied.

As observed on the KNN classifiers, also in this case we notice an almost perfect match between the detection cost metric and the error rate.

The prior expectations we had on the classifiers are generally confirmed and overall results show that the **Full Covariance model is the one with best performance**, topping at a minDCF of **0.288**, a considerable improvement over the KNN classifiers.



(a) Detection Cost (b) Error Rate

Figure 8: *Results for the Gaussian Classifiers on the training set.*

3.3 Gaussian Mixture Models

Gaussian Classifiers are able to solve classification problems by building generative models that assume that class-conditional distribution of the data can be described as gaussian distributions. Although common, however, this assumption can in many cases turn out to be quite inaccurate. **Gaussian Mixture Models (GMM)** overcome this issue by identifying those distributions as *mixtures* of gaussian distributions. Each class distribution is approximated by n gaussian components, each with its own mean $\boldsymbol{\mu_c}$ and covariance matrix $\boldsymbol{\Sigma_c}$ and an assigned weight w_c, to a desired degree. In this regard, GMMs can be viewed as a generalization of the concept behind Gaussian Classifiers. More in general, a GMM is a weighted combination of gaussian distributions:

$$f_{\boldsymbol{X}}(\boldsymbol{x}) = \sum_{c=1}^{K} w_c \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu_c}, \boldsymbol{\Sigma_c})$$

Where K is the total number of components The distribution parameters $\boldsymbol{\theta}$are:

- The component means.
$$\boldsymbol{M} = [\boldsymbol{\mu_1}, ..., \boldsymbol{\mu_K}]$$
- The component covariance matrices.
$$\boldsymbol{\mathcal{S}} = [\boldsymbol{\Sigma_1}, ..., \boldsymbol{\Sigma_K}]$$
- And the weights.
$$\boldsymbol{w} = [w_1, ..., w_k]$$
with
$$\sum_{c=1}^{K} w_c = 1$$

For each sample, we can compute how much a given component is responsible for its generation as a posterior probability called *responsibility* as a ratio of the probability for that sample to have been generated from said component over the remaining components:

$$\gamma_{c,i} = \frac{w_c \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu_c}, \boldsymbol{\Sigma_c})}{\sum_{c'} w_{c'} \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu_{c'}}, \boldsymbol{\Sigma_{c'}})}$$

A GMM classifier is trained by means of the **Expectation-Maximization (EM)** algorithm, an iterative process which estimates the responsibilities $\gamma_{c,i}$ given the parameters $\boldsymbol{\theta}$ and then recomputes the parameters $\boldsymbol{\theta}$ given the resposibilities $\gamma_{c,i}$, until a desired degree of approximation is reached.

As per the gaussian classifier, the same hypothesis on the covariance matrices can be assumed for the GMMs, leading to four different kinds of classifier. We discarded the Diagonal Tied Covariance given its low performance seen on the gaussian classifier and we proceeded with the other variants.

We once again train and evaluate our models through 6-folds cross-validation, on the raw data and on the PCA/LDA processed data, with the exception of the 16-components model which took too long to be trained and has only been evaluated on raw data.

3.3.1 Full Covariance

Model	minDCF	Error Rate
GMM Full Covariance 2 components (No Preprocessing)	**0.218**	**4.21%**
GMM Full Covariance 2 components (PCA m=9)	0.235	**4.21%**
GMM Full Covariance 2 components (PCA m=8)	0.224	4.47%
GMM Full Covariance 2 components (PCA m=7)	0.223	4.64%
GMM Full Covariance 2 components (LDA m=9)	0.406	7.95%
GMM Full Covariance 2 components (LDA m=8)	0.404	8.69%
GMM Full Covariance 2 components (LDA m=7)	0.424	8.90%
GMM Full Covariance 4 components (No Preprocessing)	**0.188**	**3.31%**
GMM Full Covariance 4 components (PCA m=9)	0.191	3.91%
GMM Full Covariance 4 components (PCA m=8)	0.203	3.44%
GMM Full Covariance 4 components (PCA m=7)	0.209	4.08%
GMM Full Covariance 4 components (LDA m=9)	0.379	7.78%
GMM Full Covariance 4 components (LDA m=8)	0.381	8.04%
GMM Full Covariance 4 components (LDA m=7)	0.402	8.55%
GMM Full Covariance 8 components (No Preprocessing)	**0.151**	**2.75%**
GMM Full Covariance 8 components (PCA m=9)	0.158	3.35%
GMM Full Covariance 8 components (PCA m=8)	0.179	3.01%
GMM Full Covariance 8 components (PCA m=7)	0.180	3.39%
GMM Full Covariance 8 components (LDA m=9)	0.329	6.66%
GMM Full Covariance 8 components (LDA m=8)	0.356	7.09%
GMM Full Covariance 8 components (LDA m=7)	0.388	7.78%
GMM Full Covariance 16 components (No Preprocessing)	**0.138**	**2.57%**
GMM Full Covariance 16 components (PCA m=9)	-	-
GMM Full Covariance 16 components (PCA m=8)	-	-
GMM Full Covariance 16 components (PCA m=7)	-	-
GMM Full Covariance 16 components (LDA m=9)	-	-
GMM Full Covariance 16 components (LDA m=8)	-	-
GMM Full Covariance 16 components (LDA m=7)	-	-

Table 9: *DCF and Error Rate results for the Full Covariance Gaussian Mixture Model*

3.3.2 Diagonal Covariance

Model	minDCF	Error Rate
GMM Diagonal Covariance 2 components (No Preprocessing)	0.294	6.83%
GMM Diagonal Covariance 2 components (PCA m=9)	0.255	**5.07%**
GMM Diagonal Covariance 2 components (PCA m=8)	0.252	**5.07%**
GMM Diagonal Covariance 2 components (PCA m=7)	**0.251**	5.29%
GMM Diagonal Covariance 2 components (LDA m=9)	0.425	8.90%
GMM Diagonal Covariance 2 components (LDA m=8)	0.423	8.90%
GMM Diagonal Covariance 2 components (LDA m=7)	0.423	9.16%
GMM Diagonal Covariance 4 components (No Preprocessing)	0.224	4.34%
GMM Diagonal Covariance 4 components (PCA m=9)	0.221	4.51%
GMM Diagonal Covariance 4 components (PCA m=8)	**0.211**	**4.12%**
GMM Diagonal Covariance 4 components (PCA m=7)	0.217	4.34%
GMM Diagonal Covariance 4 components (LDA m=9)	0.444	9.03%
GMM Diagonal Covariance 4 components (LDA m=8)	0.414	8.73%
GMM Diagonal Covariance 4 components (LDA m=7)	0.465	9.07%
GMM Diagonal Covariance 8 components (No Preprocessing)	**0.176**	3.95%
GMM Diagonal Covariance 8 components (PCA m=9)	0.204	4.04%
GMM Diagonal Covariance 8 components (PCA m=8)	0.220	**3.65%**
GMM Diagonal Covariance 8 components (PCA m=7)	0.210	3.95%
GMM Diagonal Covariance 8 components (LDA m=9)	0.400	8.64%
GMM Diagonal Covariance 8 components (LDA m=8)	0.409	8.77%
GMM Diagonal Covariance 8 components (LDA m=7)	0.432	8.90%
GMM Diagonal Covariance 16 components (No Preprocessing)	**0.159**	**3.78%**
GMM Diagonal Covariance 16 components (PCA m=9)	-	-
GMM Diagonal Covariance 16 components (PCA m=8)	-	-
GMM Diagonal Covariance 16 components (PCA m=7)	-	-
GMM Diagonal Covariance 16 components (LDA m=9)	-	-
GMM Diagonal Covariance 16 components (LDA m=8)	-	-
GMM Diagonal Covariance 16 components (LDA m=7)	-	-

Table 10: *DCF and Error Rate results for the Diagonal Covariance Gaussian Mixture Model*

3.3.3 Full Tied Covariance

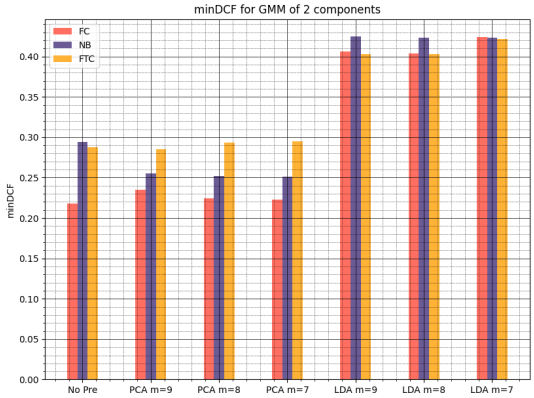
Model	minDCF	Error Rate
GMM Full Tied Covariance 2 components (No Preprocessing)	0.288	**5.29%**
GMM Full Tied Covariance 2 components (PCA m=9)	**0.285**	5.41%
GMM Full Tied Covariance 2 components (PCA m=8)	0.293	5.33%
GMM Full Tied Covariance 2 components (PCA m=7)	0.295	5.54%
GMM Full Tied Covariance 2 components (LDA m=9)	0.403	10.15%
GMM Full Tied Covariance 2 components (LDA m=8)	0.403	9.59%
GMM Full Tied Covariance 2 components (LDA m=7)	0.422	9.72%
GMM Full Tied Covariance 4 components (No Preprocessing)	**0.210**	**4.43%**
GMM Full Tied Covariance 4 components (PCA m=9)	0.214	4.81%
GMM Full Tied Covariance 4 components (PCA m=8)	0.223	4.90%
GMM Full Tied Covariance 4 components (PCA m=7)	0.217	4.73%
GMM Full Tied Covariance 4 components (LDA m=9)	0.397	8.12%
GMM Full Tied Covariance 4 components (LDA m=8)	0.410	8.21%
GMM Full Tied Covariance 4 components (LDA m=7)	0.420	8.86%
GMM Full Tied Covariance 8 components (No Preprocessing)	**0.189**	4.21%
GMM Full Tied Covariance 8 components (PCA m=9)	0.199	4.21%
GMM Full Tied Covariance 8 components (PCA m=8)	0.205	**3.91%**
GMM Full Tied Covariance 8 components (PCA m=7)	0.199	4.25%
GMM Full Tied Covariance 8 components (LDA m=9)	0.388	7.39%
GMM Full Tied Covariance 8 components (LDA m=8)	0.379	8.25%
GMM Full Tied Covariance 8 components (LDA m=7)	0.412	8.86%
GMM Full Tied Covariance 16 components (No Preprocessing)	**0.171**	**3.91%**
GMM Full Tied Covariance 16 components (PCA m=9)	-	-
GMM Full Tied Covariance 16 components (PCA m=8)	-	-
GMM Full Tied Covariance 16 components (PCA m=7)	-	-
GMM Full Tied Covariance 16 components (LDA m=9)	-	-
GMM Full Tied Covariance 16 components (LDA m=8)	-	-
GMM Full Tied Covariance 16 components (LDA m=7)	-	-

Table 11: *DCF and Error Rate results for the Full Tied Covariance Gaussian Mixture Model*

3.3.4 Considerations

Model	minDCF	Error Rate
GMM Full Covariance 16 components (No Preprocessing)	**0.138**	**2.57%**
GMM Diagonal Covariance 16 components (No Preprocessing)	0.159	3.78%
GMM Full Tied Covariance 16 components (No Preprocessing)	0.171	3.91%

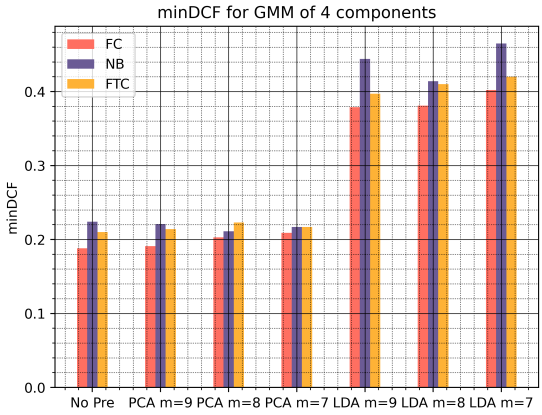
Table 12: *Best GMM classifiers.*



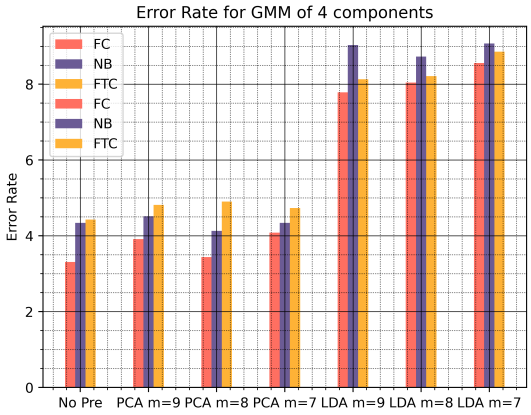
(a) Detection Cost

(b) Error Rate

Figure 9: *Results for the Gaussian Mixture Models with 2 components on the training set.*

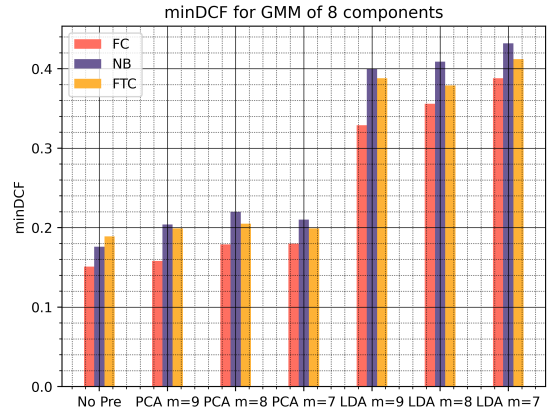


(a) Detection Cost

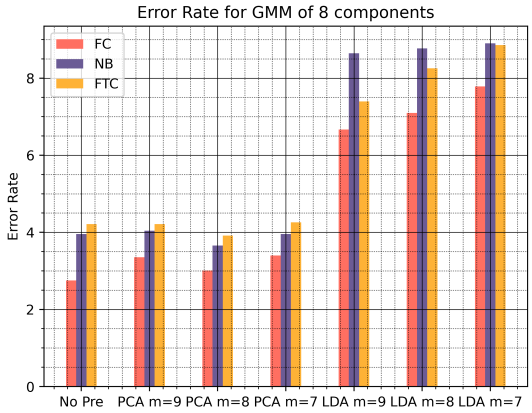


(b) Error Rate

Figure 10: *Results for the Gaussian Mixture Models with 4 components on the training set.*

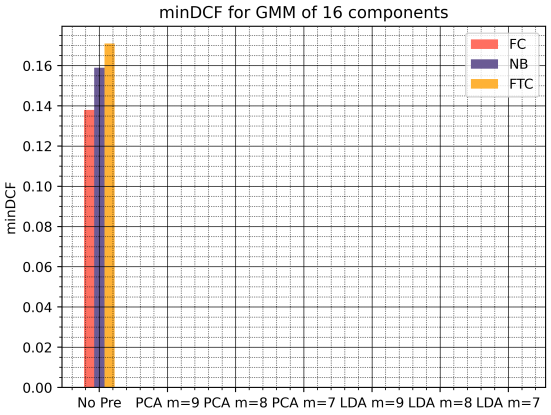


(a) Detection Cost

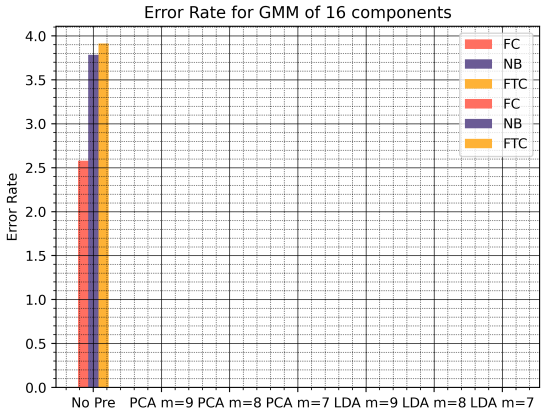


(b) Error Rate

Figure 11: *Results for the Gaussian Mixture Models with 8 components on the training set.*



(a) Detection Cost



(b) Error Rate

Figure 12: *Results for the Gaussian Mixture Models with 16 components on the training set.*

Overall, **GMMs show consistently better performances than MVGs**. This is expected since GMMs are able to more describe data with more sophisticate distribution, and as a consequence they are able to capture more nuanced variations. **This comes at the cost of an increased compute time** for the training process, due to the considerably higher number of parameters.

As observed in MVGs, applying LDA is again detrimental, while PCA allows to reduce training time without losing too much performance. Predictably, the performance of the classifiers consistently improves as the selected number of gaussian components k increases, and the models that seem to better describe the data are once again the Full Covariance ones. Again, we note an optimal match between the considered metrics.

A **Full Covariance GMM with 16 components is the new best classifier observed,** with an impressive **0.034** minimum detection cost.

3.4 Logistic Regression

We now switch our attention from generative models to discriminative ones. The difference is, where generative models model the underlying distribution of each class and attempt to understand how the data for each class is generated, discriminative models model the decision boundary between classes directly. They essentially learn how to tell classes apart, without explicitly modeling the distribution of the features.

The first discriminative models we consider are those from the **Logistic Regression (LR)** family.

3.4.1 Linear Logistic Regression

We start considering regularized Linear Logistic Regression.

Linear LR is a linear classifier that models the relationship between the features and the class labels using a linear function: the *logistic* function. The goal of linear logistic regression is to find the parameters of the linear function that best separates the positive class from the negative class in the training data.

In general, the Logistic Regression minimization problem is the following:

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

Since classes are unbalanced, we re-balance the costs of the different classes by minimizing:

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

The model parameters are $(\boldsymbol{w}, b)$. The model assumes that decision rules are linear hyperplanes orthogonal to $\boldsymbol{w}$. The hyperparameter λ, contributing to the regularization term:

$$\frac{\lambda}{2}||\boldsymbol{w}||^2$$

allow us to select an optimal value with respect to the Bias-Variance trade-off:

1. Poor separation of classes when $\lambda \gg 0$
2. Poor generalization on new data when $\lambda \approx 0$

We estimate the optimal value for the hyperparameter λ via 6-folds cross-validation.

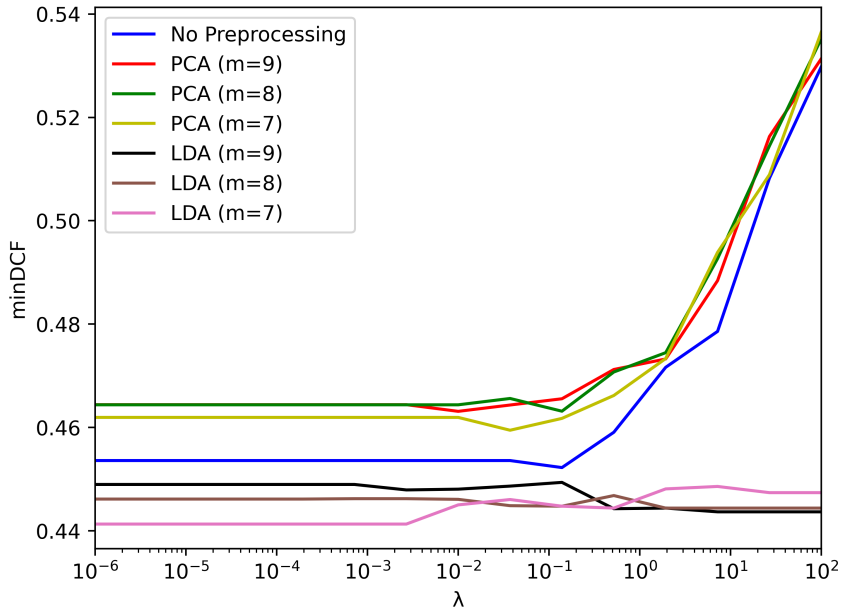
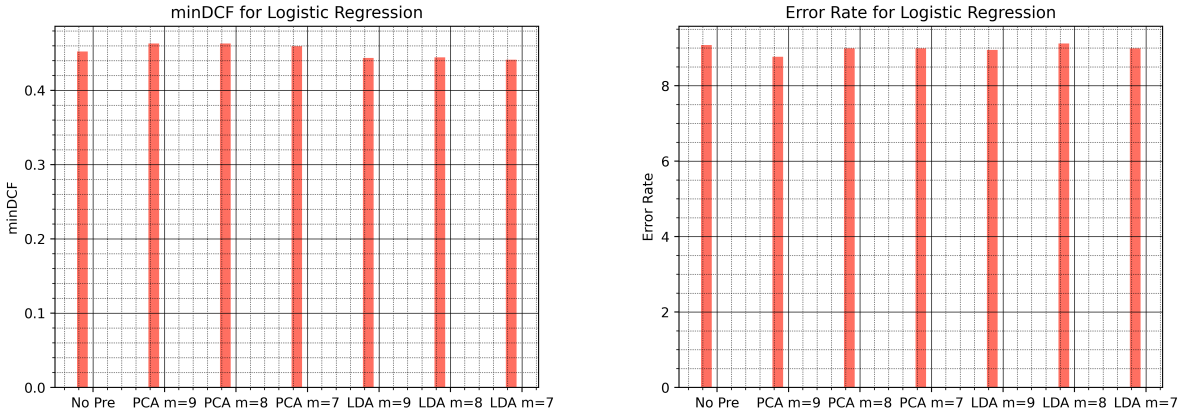


Figure 13: *minDCF across different values of λ for Linear Logistic Regression.*



(a) Detection Cost (b) Error Rate

Figure 14: *Results for the Linear Logistic Regression model on the training set.*

Model	minDCF	Error Rate
Linear LR (No Preprocessing), $\lambda = 0.139$	0.452	9.08%
Linear LR (PCA m=9), $\lambda = 0.01$	0.463	**8.77%**
Linear LR (PCA m=8), $\lambda = 0.139$	0.463	8.99%
Linear LR (PCA m=7), $\lambda = 0.037$	0.459	8.99%
Linear LR (LDA m=9), $\lambda = 7.20$	0.444	9.29%
Linear LR (LDA m=8), $\lambda = 1.93$	0.444	9.20%
Linear LR (LDA m=7), $\lambda = 0.000001$	**0.441**	8.99%

Table 13: *DCF and Error Rate results for the best λ values for the Linear Logistic Regression Classifier.*

3.4.2 Quadratic Logistic Regression

The quadratic version of the LR Classifier is based on the mapping between the data feature space on an *expanded* feature space.

$$\phi(\boldsymbol{x}) = \begin{bmatrix} vec(\boldsymbol{x}\boldsymbol{x}^T) \\ \boldsymbol{x} \end{bmatrix}$$

with $vec(\boldsymbol{M})$ defined as the operator that stacks the columns of a matrix $\boldsymbol{M}$.

With the log-likelihood defined as

$$log\frac{P(C = h_1|\boldsymbol{x})}{P(C = h_0|\boldsymbol{x})} = s(\boldsymbol{x}, \boldsymbol{A}, \boldsymbol{b}, c) = \boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}^T\boldsymbol{x} + c$$

and defining

$$\boldsymbol{w} = \begin{bmatrix} vec(\boldsymbol{A}) \\ \boldsymbol{b} \end{bmatrix}$$

then the class log-posterior ratio (the scores computed by the classifier) can be expressed as:

$$s(\boldsymbol{x}, \boldsymbol{w}, c) = \boldsymbol{w}^T\phi(\boldsymbol{x}) + c$$

If we train an LR model using feature vectors $\phi(\boldsymbol{x})$ rather than $\boldsymbol{x}$ we obtain a model that has linear separation surface in the space defined by the mapping ϕ (the expanded feature space), hence linear separation rules for the transformed features $\phi(\boldsymbol{x})$, but since the expression $\boldsymbol{w}^T\phi(\boldsymbol{x}) + c$ corresponds to quadratic forms in the original feature space, **we are actually estimating quadratic separation surfaces in the original space.**

Summing up, Quadratic LR works by expanding the original dimensionality of the data and by finding linear separation rules on the expanded space which translate to quadratic separation rules on the original space. A similar trick is also employed by Support Vector Machines, covered in the next section.

Due to the heavy computational load required for training Quadratic LR models, 3-folds cross-validation has been employed and fewer variations of pre-processing methods have been explored.

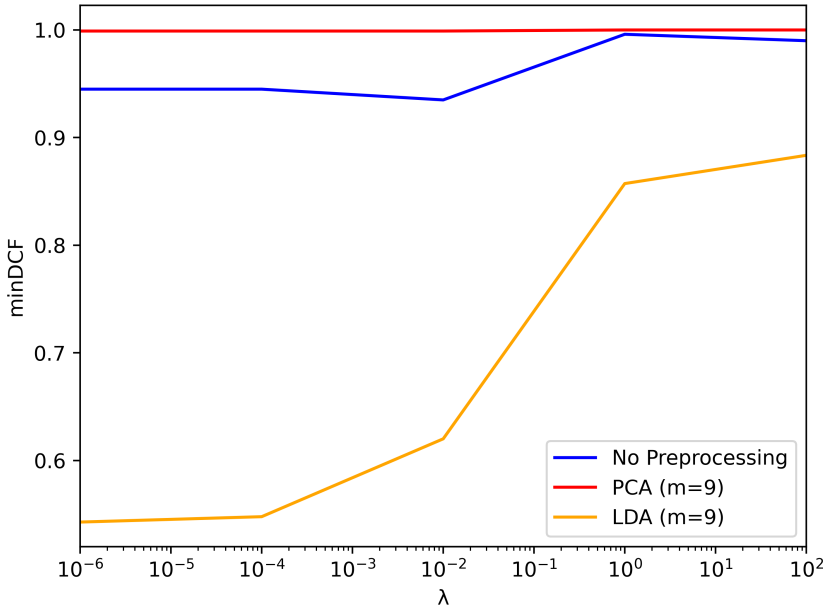
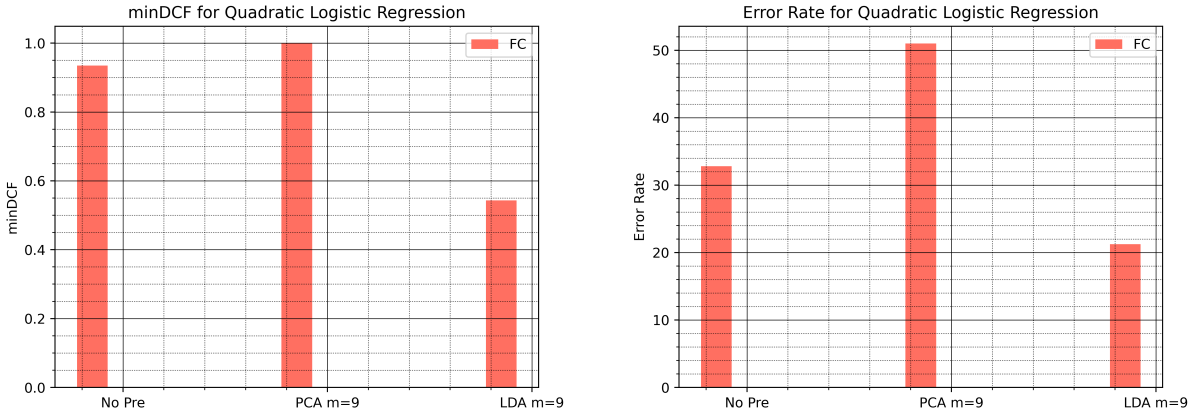


Figure 15: *minDCF across different values of λ for Quadratic Logistic Regression.*



(a) Detection Cost

(b) Error Rate

Figure 16: *Results for the Quadratic Logistic Regression model on the training set.*

Model	minDCF	Error Rate
Quadratic LR (No Preprocessing), $\lambda = 0.01$	0.935	32.82%
Quadratic LR (PCA m=9), $\lambda = 0.01$	0.999	51.01%
Quadratic LR (LDA m=9), $\lambda = 0.000001$	**0.543**	**21.25%**

Table 14: *DCF and Error Rate results for the best λ values for the Quadratic Logistic Regression Classifier.*

3.4.3 Considerations

Model	minDCF	Error Rate
Linear LR (LDA m=7), $\lambda = 0.000001$	**0.441**	**8.99%**
Quadratic LR (LDA m=9), $\lambda = 0.000001$	0.543	21.25%

Table 15: *Best LR classifiers.*

Both the **LR models perform considerably worse** than the best previously considered classifiers on the training set. In particular, Quadratic LR in practically unusable on every case analyzed. This comes quite as a surprise, since from what emerged on generative classifiers it seems that our data is more easily separable by quadratic separation boundaries. We could hypothesize that at least part of the reasons for this disappointing results are due to the wrong assumption of the target application on the prior probability of the true class, considered at 0.5 when the actual ratio of true labels in the dataset is $\approx 30\%$.

Linear LR shows better results, but still overall subpar when compared to previous best encountered models. On both metrics it achieves results similar to the linear MVGs models. If compared to the KNN classifiers it scores similar on the primary metric, minDCF, but on the error rate it ranks considerably worse. All of the analyzed versions show similar results, without significant variance.

Best results tops at a minDCF of **0.441**.

3.5 Support Vector Machines

Finally, we consider **Support Vector Machines (SVM)**.

SVMs are a type of supervised learning algorithm that are used to find the **decision boundary**, which is an hyperplane that separates different classes in a dataset. The goal of an SVM is to find the hyperplane that maximizes the distance (the *margin*) between the hyperplane and the closest data points from each class, also known as **support vectors**. SVMs are particularly useful when the data is not linearly separable in the original input space.

Due to the computational load required to train these models, coupled with the number of combinations their hyperparameters can assume, SVMs have only been trained on single-fold approaches. This means that the following results may represent the true performance of the classifiers not as reliably as those obtained on k-folds cross validation approaches. In general, we noticed that single-fold approach tends to lead to slightly pessimistic results.

3.5.1 Linear SVM

For a **Linear SVM**, we need to tune the hyper-parameter C. We start with a model that does not balance the two classes. To solve the SVM problem we consider the dual formulation:

$$J^D(\boldsymbol{\alpha}) = -\frac{1}{2}\boldsymbol{\alpha}^T \boldsymbol{H} \alpha + \boldsymbol{\alpha}^T \mathbf{1}$$

subject to:

$$0 \leq \alpha_i \leq C, \forall i \in \{1, ..., n\}$$

and

$$\sum_{i=1}^{n} \alpha_i z_i = 0$$

where n is the number of training samples and $\mathbf{1}$ is a n-dimensional vector of ones and the hyperparameter C is a real number with the role to add a penalty for each misclassified sample. For low values of C, hence low misclassification penalties, the decision boundary will be chosen with a large margin at the expense of a loss in accuracy and a greater number of misclassifications, while for high values of C the penalty for each misclassification will be higher and this will result in greater accuracy at the expense of a relevant increase in the training time and it could also lead to overfitting problems. z_i is the class label, which in the binary classification case for the SVMs are usually encoded as +1 for positive samples (authentic fingerprints) and -1 for negative samples (spoofed fingerprints).
Finally $\boldsymbol{H}$ is a matrix whose elements are:

$$\boldsymbol{H}_{i,j} = z_i z_j \boldsymbol{x}_i^T \boldsymbol{x}_j$$

The constraint

$$\sum_{i=1}^{n} \alpha_i z_i = 0$$

derives from the presence of the bias term in the SVM primal formulation. In our implementation we exploit the L-BFGS-B algorithm from the scipy library, that is only able to handle box-constraints. Therefore we need to slightly modify our dual formulation to apply it and taking in account also the constraint above. The modified dual formulation is:

$$\widehat{J}^D(\boldsymbol{\alpha}) = -\frac{1}{2}\boldsymbol{\alpha}^T \widehat{\boldsymbol{H}} \alpha + \boldsymbol{\alpha}^T \mathbf{1}$$

subject to:

$$0 \leq \alpha_i \leq C, \forall i \in \{1, ..., n\}$$

and since we use the mapping: $\widehat{x_i} = \begin{bmatrix} x_i \\ K \end{bmatrix}$, when K = 1, the matrix $\widehat{\boldsymbol{H}}$ is modified such as:

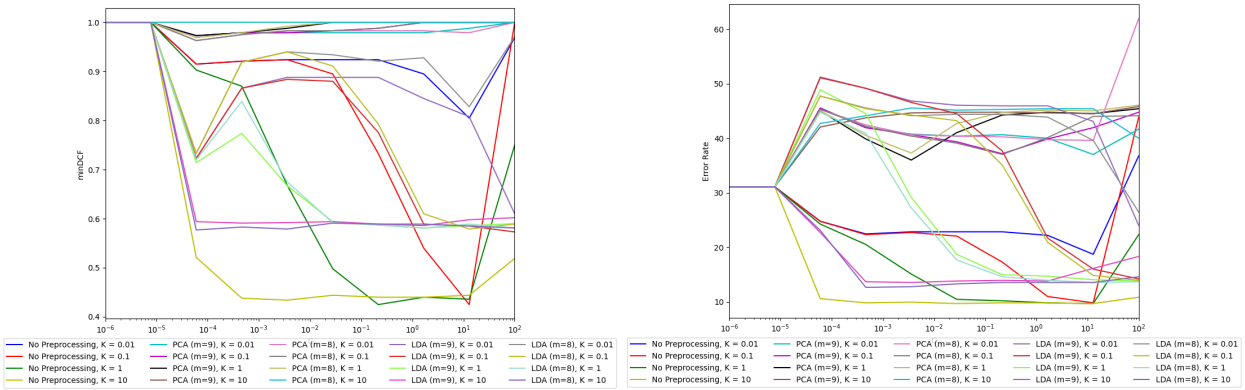
$$\widehat{\boldsymbol{H}}_{\boldsymbol{i,j}} = z_i z_j (\boldsymbol{x}_i^T \boldsymbol{x}_j + 1)$$

To compute the score of a test sample we need to compute the sum over all the training samples such as:

$$s(\boldsymbol{x}_t) = \sum_{i=1|\alpha_i>0}^{n} \alpha_i z_i \boldsymbol{x}_i^T \boldsymbol{x}_t + b$$

where b is the bias term. Differently to the classifiers encountered up until this point, **SVMs scores do not have a probabilistic interpretation**.

Here follow the results for the single-fold cross validation on the training set across increasing values for the hyperparameter C and considering four different values for K.



(a) Detection Cost (b) Error Rate

Figure 17: *Results for the Linear Support Vector Machine on the training set.*

3.5.2 Non-Linear SVMs

SVMs allow for non-linear classification through an implicit expansion of the features in an higher dimensional space. The SVM dual objective depends on the training samples only through dot-products. By this method, called also *kernel trick*, SVMs do not require to explicitly compute the feature expansion: it is sufficient to compute the scalar product between the expanded feature through a *kernel function* k.

$$k(\boldsymbol{x}_1, \boldsymbol{x}_2) = \boldsymbol{\Phi}(\boldsymbol{x}_1)^T \boldsymbol{\Phi}(\boldsymbol{x}_2)$$

So that we can rewrite $\widehat{H}$:

$$\widehat{\boldsymbol{H}}_{\boldsymbol{i,j}} = z_i z_j k(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

A kernel function allows training a SVM in a large dimensional Hilbert space $\mathcal{H}$, without requiring explicit computation of the mapping. The complexity only depends on the number of training points. In practice, we are computing a linear separation surface in the expanded space, which corresponds to a non–linear separation surface in the original feature space.

To compute the score of a test sample we need to compute the sum over all the training samples such as:

$$s(\boldsymbol{x}_t) = \sum_{i=1|\alpha_i>0}^{n} \alpha_i z_i \boldsymbol{\Phi}(\boldsymbol{x}_1)^T \boldsymbol{\Phi}(\boldsymbol{x}_2) + b = \sum_{i=1|\alpha_i>0}^{n} \alpha_i z_i k(\boldsymbol{x}_i, \boldsymbol{x}_t) + b$$

where b is the bias term.

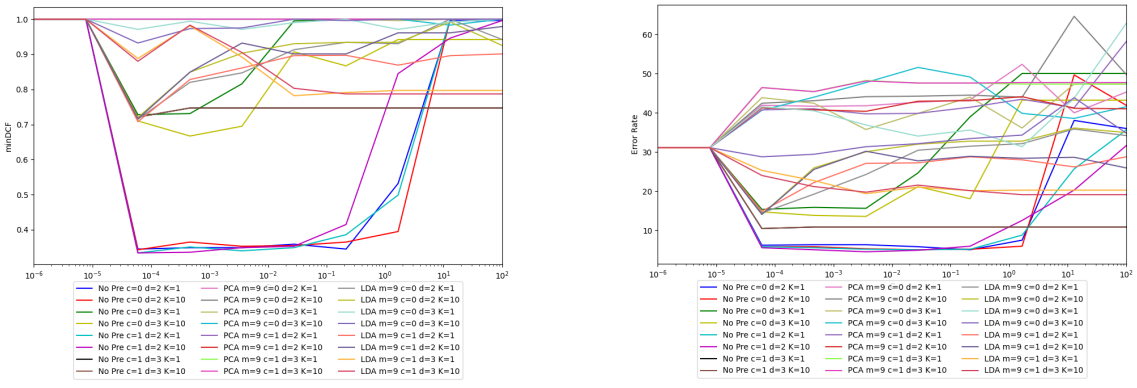
We will analyze two different kernel functions, the Polynomial function and the Radial Basis Function.

3.5.2.1 Polynomial Kernel SVM

We will first consider a **Polynomial** kernel of degree d such as:

$$k(\boldsymbol{x}_1, \boldsymbol{x}_2) = (\boldsymbol{x}_1^T \boldsymbol{x}_2 + c)^d$$

we will consider both quadratic and cubic formulations with $d = 2$ and $d = 3$.
We still need to determine an optimal value for the hyperparameters C and K, but this time we need to perform a grid search to jointly optimize also the parameters c and d.



(a) Detection Cost (b) Error Rate

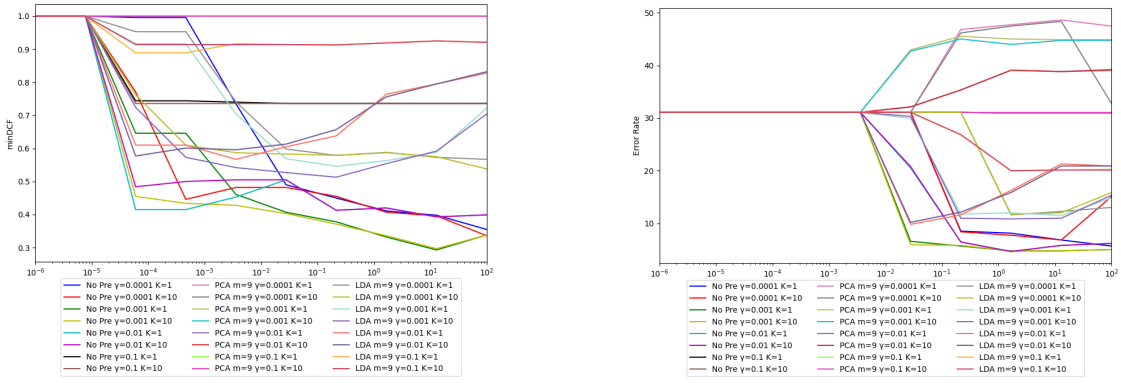
Figure 18: *Results for the Polynomial Support Vector Machine on the training set.*

3.5.2.2 Radial Basis Function Kernel SVM

The last model we consider is the SVM with the **Radial Basis Function (RBF)** kernel formulation:

$$k(\boldsymbol{x_1}, \boldsymbol{x_2}) = e^{-\gamma||\boldsymbol{x_i}-\boldsymbol{x_j}||^2}$$

This time we need to jointly estimate optimal values for the hyperparameters C and γ in the same fashion as above.



(a) Detection Cost (b) Error Rate

Figure 19: *Results for the RBF Support Vector Machine on the training set.*

3.5.3 Considerations

Model	minDCF	Error Rate
Linear SVM (No Preprocessing), $K = 10, C \approx 0.003$	0.434	9.94%
Polynomial SVM (No Preprocessing), $c = 1, d = 2, K = 10, C \approx 0.00006$	0.334	5.55%
RBF SVM (No Preprocessing), $\gamma = 0.001, K = 10, C \approx 13$	**0.297**	**4.77%**

Table 16: *Best SVMs classifiers*

In general, SVMs behave well on the considered dataset. Once again, non-linear models significantly outperform linear models, with the Linear SVM achieving comparable performance to that of Linear LR and Linear MVGs. Polynomial and RBF SVMs, on the other hand, rank as the best discriminative models tried so far. In particular, **RBF SVM** achieves a minimum DCF of **0.297**, still higher than those observed on the gaussian-based generative models. It must be taken into account that SVMs had only been considered on single-fold cross-validation approach, whether all previous models have been evaluated via 6-folds cross-validation, so the **performance measured might be considered pessimistic** w.r.t their true potential.

3.6 Best Classifiers

Having completed the training phase for all our considered models, it is now time to select those to be fatherly evaluated and to propose one (or more) candidate solution(s) to our classification tasks.

In the following table we list the best model evaluated for each of the five model families encountered, among which the best classifiers will be selected. We decide to select only one classifier for each model type, even if most of the top performing classifiers evaluated are all belonging to the GMM family, both because we believe that the best performing GMM is representative enough of its whole family and to present a vary enough selection of models with different properties, more interesting to compare with each other.

Model	**minDCF**	**Error Rate**
KNN (PCA m=7), $k = 5$	0.441	4.39%
MVG Full Covariance (PCA m=8)	0.288	5.68%
GMM Full Cov (No Preprocessing), $K = 16$	**0.138**	**2.57%**
Linear LR (LDA m=7), $\lambda = 0.000001$	0.441	8.99%
RBF SVM (No Preprocessing), $\gamma = 0.001, K = 10, C \approx 13$	0.297	4.77%

Table 17: *Best classifiers.*

We consider Linear LR results as unacceptable so we discard the model a will not consider it anymore. KNN classifiers, although showing promising results, due to their non parametric nature are harder to compare with the other models. For this reason these models will be discarded for now but we will use them as a benchmark on the final evaluations.

The GMM is by far the top performing model on both the primary and secondary metric, so **we select the GMM with 16 components as our candidate solution.** However, we decide to bring along to the next evaluation phase also the MVG Full Cov model and the RBF SVM.

4 Score Calibration

Until now we used as the performance metric the Minimum Detection Cost. As already discussed, MinDCF measures the cost we would pay if we made optimal decisions for the validation set using the scores of the recognizer. The cost that we actually pay however, is different. We call **Actual Detection Cost (ActDCF)** the actual cost that we pay, which depends on the goodness of the threshold we use to perform class assignment. For this reason we will start considering the ActDCF metric.
We could start by assuming that our scores are already calibrated. If this was the case, the optimal threshold that optimizes the Bayes risk is described by:

$$t = -log\frac{\widetilde{\pi}}{1-\widetilde{\pi}}$$

where $\widetilde{\pi}$ is the *effective prior* probability of $\mathcal{H}_\mathcal{T}$, which is a prior that, if the class prior for $\mathcal{H}_\mathcal{T}$ was $\widetilde{\pi}$ and we assumed uniform costs, hence considering an equivalent application ($\widetilde{\pi}$, 1, 1), we would obtain the same decisions as for our original application considered (0.5, 1, 10).

We can compute the effective prior as:

$$\frac{\widetilde{\pi}}{1-\widetilde{\pi}} = \frac{\pi_T C_{fn}}{(1-\pi_T)C_{fp}}$$

hence, for $\pi_T = 0.5, C_{fn} = 1$ and $C_{fp} = 10$, we get $\widetilde{\pi} = 1/110$.

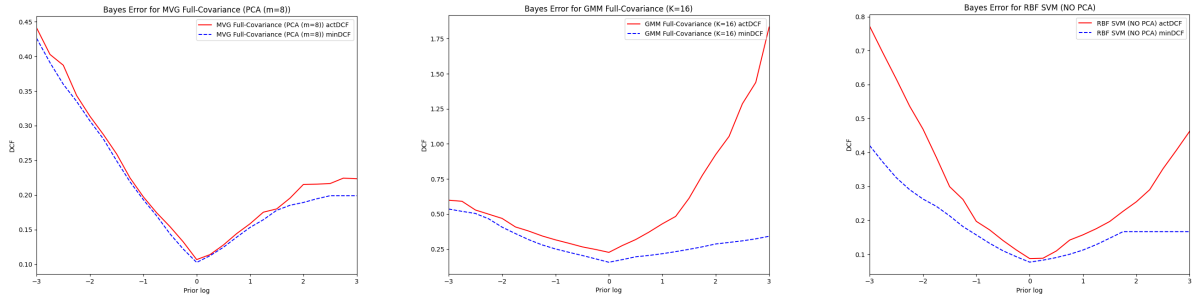
We can now, assuming that our scores are already well-calibrated, evaluate the actual DCF of our best tested classifiers to measure how good the model would behave if we were using the theoretical threshold.

Model	minDCF	actDCF
MVG Full Covariance (PCA m=8)	0.288	0.671
GMM Full Cov (No Preprocessing), $K = 16$	0.138	0.250
RBF SVM (No Preprocessing), $\gamma = 0.001, K = 10, C \approx 13$	0.238*	0.442

Table 18: *Gap between minimum DCF and actual DCF. *This value is much better than the 0.297 obtained in the previous section. This is because we are now utilizing again 6-folds cross validation. The corresponding value on the Error Rate metric is 4.39%*

From the results we can observe that none of our scores is actually already well calibrated.

This analysis can be further verified through Bayes Error Plots:



(a) Full Cov MVG (b) GMM 16 Components (c) RBF SVM

Figure 20: *Bayes Error Plots for the considered classifiers*

The Bayes Error Plots highlight the poor calibration which affects in particular our GMM and RBF SVM models. The Full Cov MVG model instead look already calibrated pretty well. **We therefore resort to score calibration on the RBF SVM and GMM models** and we double check it on the Full Cov MVG model.

Our approach will consist in computing a transformation function $f(s)$ which will map the scores s of each classifier to well-calibrated scores $s_{cal} = f(s)$. We assume that the function $f(s)$ is linear, such as:

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

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

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

so that the class posterior probability for a prior $\widehat{\pi}$ can be defined as:

$$log\frac{P(C = H_T|s)}{P(C = H_F|s)} = \alpha s + \beta + log\frac{\widehat{\pi}}{1 - \widehat{\pi}}$$

We notice if we interpret the scores s as features this expression will be similar to the log-posterior ratio of the Logistic Regression model. Let's define:

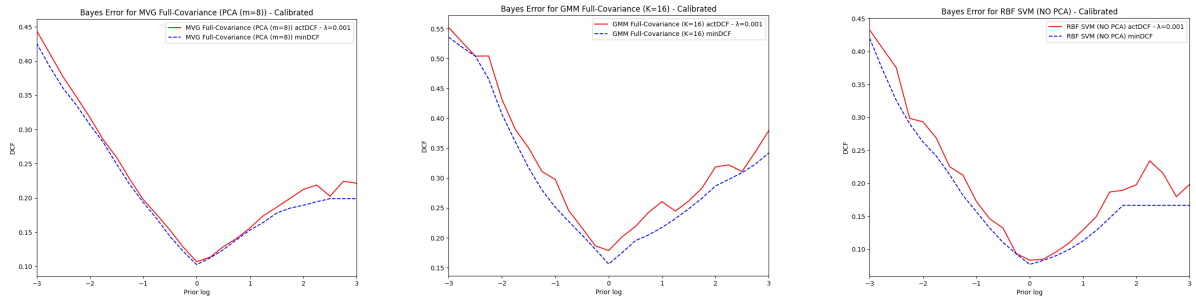
$$\beta' = \beta + log\frac{\widehat{\pi}}{1 - \widehat{\pi}} \Rightarrow \beta = \beta' - log\frac{\widehat{\pi}}{1 - \widehat{\pi}} \Rightarrow log\frac{P(C = H_T|s)}{P(C = H_F|s)} = \alpha s + \beta'$$

and we have exactly the same model. This means that we can use the prior-weighted Logistic Regression model to learn the parameters α and β' over our training scores.

Finally, we can recover the calibrates scores $f(s)$ computing:

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

We can plot again the Bayes Error Plots with the learned parameters.



(a) Full Cov MVG (b) GMM 16 Components (c) RBF SVM

Figure 21: *Bayes Error Plots for the calibrated classifiers*

It is evident from the plots that, as expected, score calibration proven to be effective in rectifying our models. By looking at these plots we can also make assumptions on how our best models would behave on different working points.

5 Evaluation

5.1 Target Application

Finally, it is now time to conclude our analysis by **evaluating our classifiers on the test set**. Our main goal is to assess the performance of the selected classifiers on the test set, for the target application, in terms of the primary metric.

We are not going to resort to K-folds validation now, since we are going to use the whole training dataset to train our models and the whole test dataset to evaluate them.

Model	minDCF	Error Rate
MVG Full Covariance (PCA m=8)	0.276	4.10%
GMM Full Cov (No Preprocessing), $K = 16$	0.307	5.15%
RBF SVM (No Preprocessing), $\gamma = 0.001, K = 10, C \approx 13$	**0.243**	**3.69%**

Table 19: *Performance of the selected models on the test set.*

Looks like our choice to bring along with ourselves in our analysis not only the best performing classifier during the training phase but also two other models apparently less performing was a wise one because, against our expectations, the results show that the GMM Full Cov with 16 components model is actually the worse among those tested against the test set. It ranks significantly lower than the Full Cov MVG model which is basically and minimal version of itself. This is most certainly due to **overfitting**, a problem we pretended to ignore until now but that is actually a very common issue in machine learning. Overfitting happens when the model learns the detail and noise in the training data to the extent that it negatively impacts the performance on new data, fitting the noise instead of the underlying pattern. This can happen, as we assume is our case, for models which are too complex, i.e. too many features/parameters relative to the number of observations. We can try to address this issue by considering a simpler version of the GMM Full Covariance model.

Model	minDCF	Error Rate
MVG Full Covariance (PCA m=8)	0.276	4.10%
GMM Full Cov (No Preprocessing), $K = 4$	**0.238**	3.89%
RBF SVM (No Preprocessing), $\gamma = 0.001, K = 10, C \approx 13$	0.243	**3.69%**

Table 20: *Performance of the selected models on the test set.*

As expected, by selecting a simpler GMM among the ones analyzed earlier, we get as result a considerable increase in performance, **with a minDCF of 0.238 the GMM Full Cov with K=4 is the best performing classifier on the test set**, among those selected.

Overall, models perform accordingly to what we observed in the training phase, if not even better. We can compare the obtained classifiers through a ROC plot.

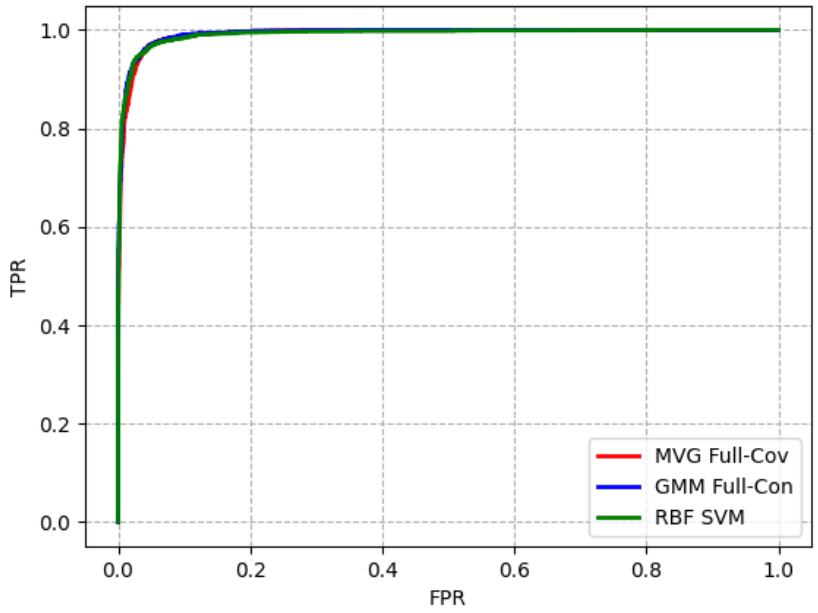


Figure 22: *ROC Plot for comparison of our best models after score calibration.*

5.2 Different Applications

We will now assess the performance of our selected models on applications different from the target application considered up until this point.

First, we will consider an application which tries to simulate more accurately the performance of our classifiers if **deployed on a real application**. On an actual use case we may assume the amount of trials to access a system protected by biometric authentication (e.g. a smartphone) with spoofed credentials to be much lower with respect to the amount of legitimate accesses. For this reason, we evaluate our classifiers on a application defined by the triplet ($\pi_T = 0.9, C_{Fn} = 1, C_{Fp} = 10$). We measure only the primary metric since the error rate is not affected by priors and costs.

Model	minDCF
MVG Full Covariance (PCA m=8)	**0.082**
GMM Full Cov (No Preprocessing), $K = 4$	0.086
RBF SVM (No Preprocessing), $\gamma = 0.001, K = 10, C \approx 13$	0.083

Table 21: *Performance of the selected models on the test set considering a ($\pi_T = 0.9, C_{Fn} = 1, C_{Fp} = 10$) application.*

On a similar application all the models would perform very good and somewhat at the same level.

We now consider a different but still plausible application in which the system is deployed to protect some very valuable good which is not accessed very often, like for example could be a domestic safe. Such system should penalize heavily false positive costs without making much assumptions on the prior probability for an authorized or unauthorized access. We therefore evaluate an application defined by the triplet ($\pi_T = 0.5, C_{Fn} = 1, C_{Fp} = 100$).

Model	minDCF
MVG Full Covariance (PCA m=8)	0.551
GMM Full Cov (No Preprocessing), $K = 4$	0.529
RBF SVM (No Preprocessing), $\gamma = 0.001, K = 10, C \approx 13$	**0.484**

Table 22: *Performance of the selected models on the test set considering a ($\pi_T = 0.5, C_{Fn} = 1, C_{Fp} = 100$) application.*

All of our models experience a decrease in performance in such scenario. Therefore, they would not be an optimal choice for this kind of applications.

5.3 Final Evaluations

Finally, we benchmark our models on the considered target application against the models we discarded earlier, to assess that our choices were indeed optimal. In particular, we reconsider the K-Nearest Neighbors models and the Linear Logistic Regression, evaluating them on the test set.

Model	minDCF	Error Rate
KNN, $k = 5$	0.402	4.34%
Weighted KNN $k = 7$	**0.369**	**3.99%**
Linear LR (LDA m=7), $\lambda = 0.000001$	0.460	8.98%

Table 23: *Performance of the discarded models on the test set.*

We can confirm that our choices were indeed motivated. The models performance align with what we observed in the training section. **The Linear LR is particularly suboptimal on both the measured metrics.** The KNN classifiers' performance however, even if subpar when compared to those of our selected models, is in general not so bad, especially if we consider that these models do not need any training step and are very computationally lightweight. Although not indicated for safety-centered applications, **KNN models could constitute a valuable resource in resource-constrained environments.**

6 Conclusions

The goal of this project was to assess the efficacy of various machine learning supervised algorithms in distinguishing between genuine and spoofed fingerprint images. To achieve this goal, we employed a diverse range of classification techniques, such as Nearest Neighbors, Multivariate Gaussian Classifiers, Gaussian Mixture Models, Logistic Regression, and Support Vector Machines, and compared their performance using the minimum detection cost function as primary metric and the error rate metric as tie-breaker.

Our results showed that the analyzed dataset can be successfully classified by the machine learning techniques we considered, with a Full Covariance Gaussian Mixture Model with 4 components achieving the best performance, followed by a Radial Basis Function Support Vector Machine and by a Full Covariance Multivariate Gaussian Classifiers. In general, the dataset has been more easily classified by non-linear models as opposed to linear ones, we observed that quadratic models generally performed better, and that the full covariance models outperformed the diagonal or tied covariance models. Moreover, we confirmed our expectations on how in general applying LDA to the data lead to worse classification performances.

Further analysis could be conducted by considering a wider range of hyperparameters during training and by allocating more computational resources and time, which we found to be main bottleneck and hardship to overcome during the development of this project.