Report Machine Learning & Pattern Recognition Project

s314913 Gabriele Lorenzo

July 2024

Abstract

In this report, we present the project about fingerprint spoofing detection that we realized for the course of Machine Learning and Pattern Recognition.
The dataset consists of labeled samples corresponding to the genuine (True, label 1) class and the fake (False, label 0) class. We will analyze the features of the dataset, perform dimensionality reduction, train and validate the classifiers on the training dataset, and finally test the classifiers on the testing dataset.
We will use different classifiers: Gaussian Classifiers, Logistic Regression, Support Vector Machine, and Gaussian Mixture Model. We will evaluate them based on the error rate and the Detection Cost Function (DCF).

Contents

1 Overview **2**

2 Feature Analysis **3**
2.1 Preliminary Analysis 3
2.2 Gaussian Feature Analysis 6

3 Dimensionality Reduction **7**

4 Training & Validation **11**
4.1 Introduction 11
4.2 Multivariate Gaussian Classifier 12
4.3 Logistic Regression Classifier 18
4.4 Support Vector Machine Classifier 22
4.5 Gaussian Mixture Model Classifier 24
4.6 Model Comparison 26
4.7 Score Calibration 28
4.8 Model Fusion 31

5 Experimental Results **35**

6 Conclusions **42**

1. Overview

The main project task is to create a classifier that allow us to reliably do what is called fingerprint spoofing detection. We can view this task as a **binary classification** problem where we have to distinguish genuine and counterfeit fingerprints.

The dataset consists of labeled samples corresponding to the genuine (True, label 1) class and the fake (False, label 0) class. Samples are computed by a feature extractor that summarizes high-level characteristics of a fingerprint image. Both training and testing dataset are 6-dimensional, with the training data having 6000 samples and the testing data having also 6000 samples.

2. Feature Analysis

2.1 Preliminary Analysis

The first step is to analyze the features of the dataset.

We will plot the histograms of the features for the genuine and fake classes.

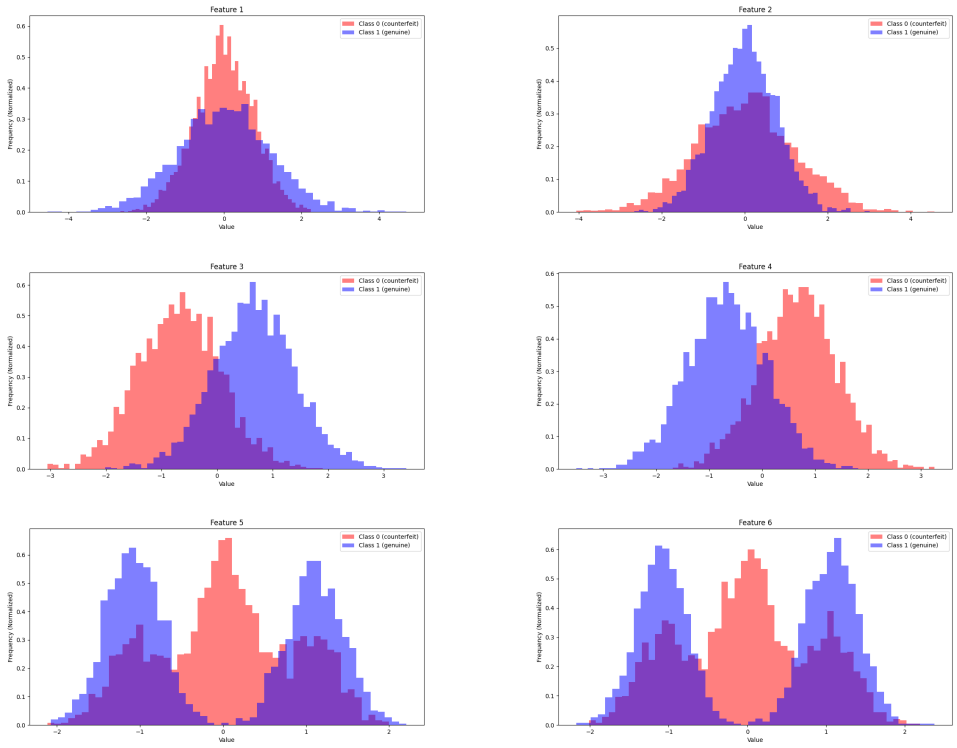

Figure 2.1: Histograms of the features for the genuine and fake classes.

As we can see the classes for features 1 and 2 have a similar mean, while their variance shows some differences: class 1 has higher variance than class 0 for feature 1 and lower variance than class 0 for feature 2. Both of the

features are approximately gaussian distributed and the two classes overlap quite heavily on the middle part of the distribution. Since we can observe one main peak in our distributions (only one mode) we expect gaussian models to perform poorly.

We observe that also features 3 and 4 are gaussian distributed and posses a similar variance, while their mean is different. The two classes overlap only in a portion of the distribution (since the means are quite different the overlap is only on the respective sides of the distributions). Here we can see two modes and gaussian models should clearly identify the two classes when using these features.

Feature 5 and 6 are more complex. We observe three modes and the two features are not gaussian distributed. For both features class 1 can be modeled by two distinct gaussians, while class 0 can be modeled by a three gaussians (we can hence see two clusters for class 1 and three clusters for class 0). The two distributions overlap in the intervals corresponding to the two modes of class 1. We expect gaussian mixture models to perform better than simple gaussian models for these features.

We can also observe the pairwise plot of the features for the genuine and fake classes.

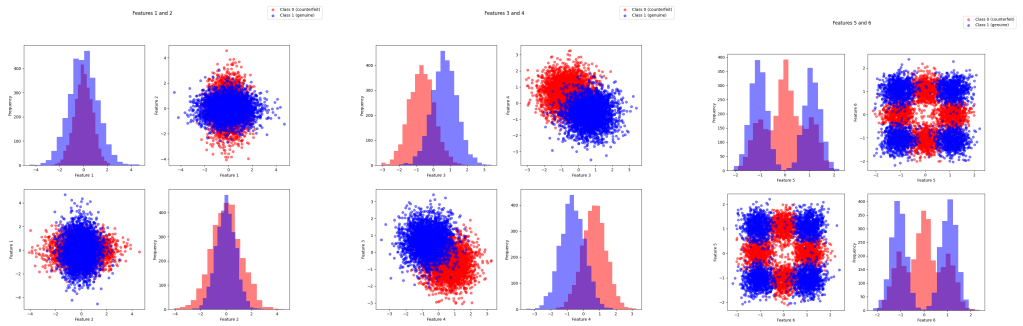

Figure 2.2: Pairwise plot of the features for the genuine and fake classes.

The analysis previous analysis is confirmed by the pairwise plot of the features. We can see a string overlap for both classes in features 1 and 2, while we can see a separation in features 3 and 4. Features 5 and 6 show a more

complex structure with more clusters for each class.

Finally we can see the pairwise plot of every feature with every other feature.

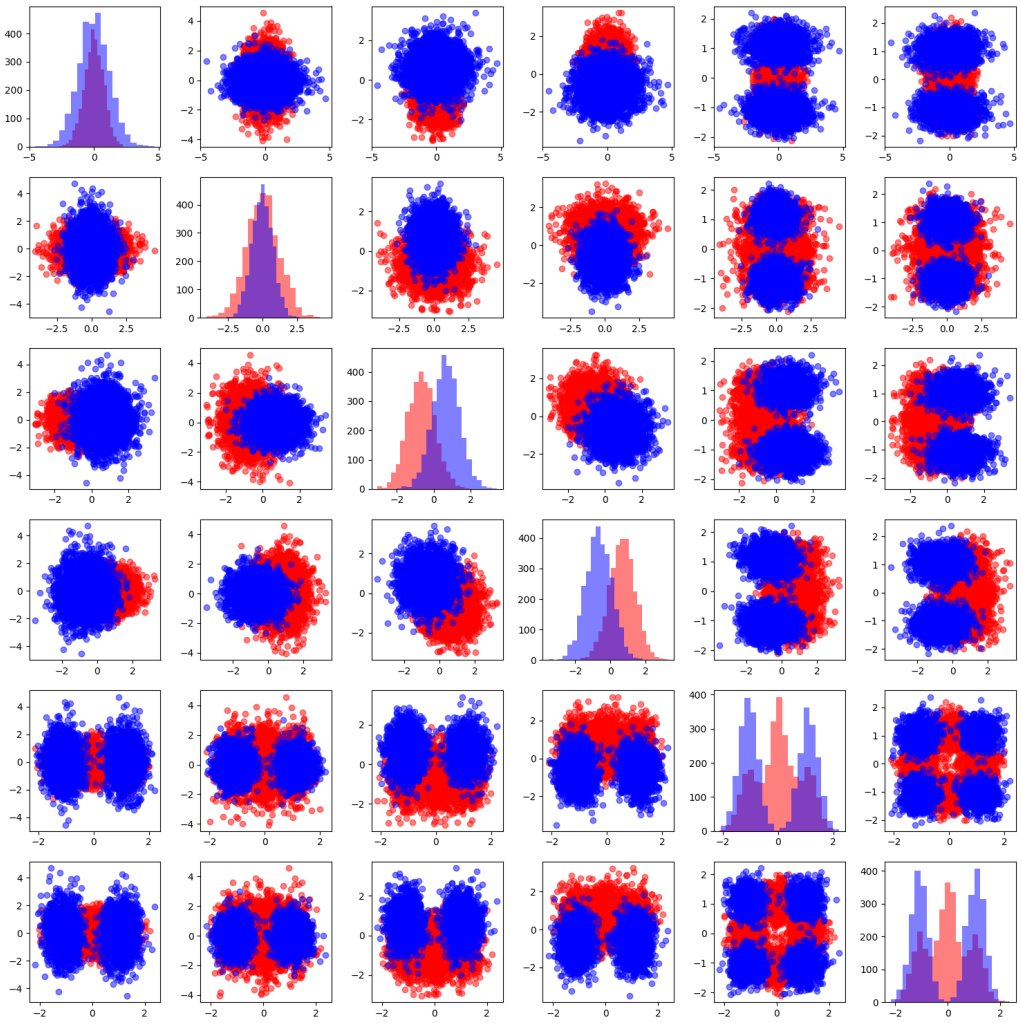

Figure 2.3: Pairwise plot of every feature with every other feature.

From this preliminary analysis we can see that the features are not perfectly linearly separable and that we might to use more complex models to classify the samples.

2.2 Gaussian Feature Analysis

We proceed to refine our analysis by fitting a gaussian model to the data.

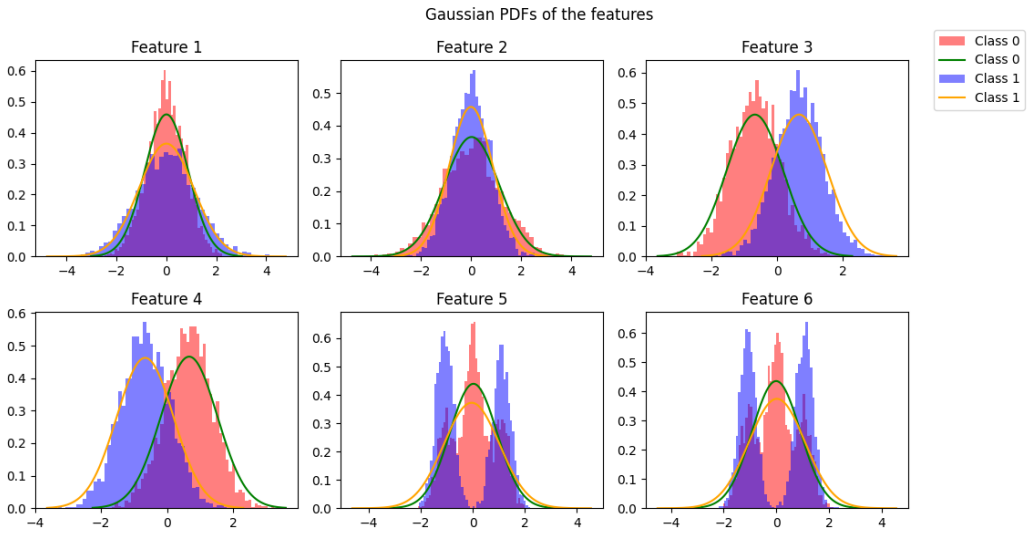

Figure 2.4: Gaussian model fitted on each feature.

As previously stated, there are discrepancies between the distributions of the two classes for each feature.
We can see that features 3 and 4 are the most promising for a gaussian model, while features 5 and 6 are the least promising. We will probably need to use a gaussian mixture model or other more complex non linear models to classify the samples.

3. Dimensionality Reduction

To gain more insight into the data we can perform dimensionality reduction using PCA and LDA.

We will first start by using PCA and project our data on the 6 PCA directions, starting from the direction with the highest variance. In the following we will refer to PCA components as the projection of data on the PCA directions.

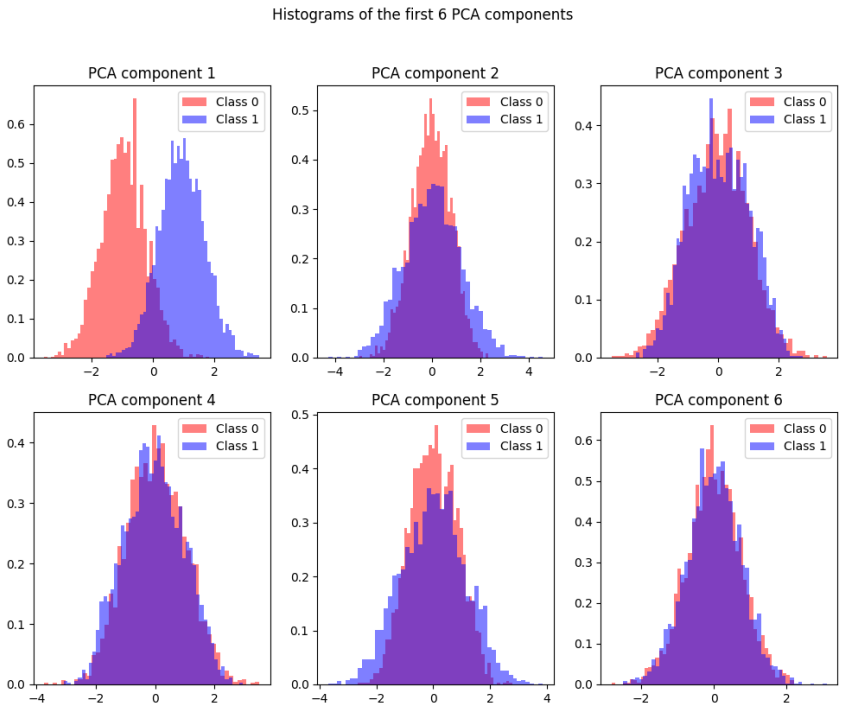

Figure 3.1: PCA projection of the data.

As we can see, the first PCA component is able to separate the two classes

quite well, while the rest of the components do worst at separating the two classes.

Now we can use LDA to project the data on the first LDA direction.

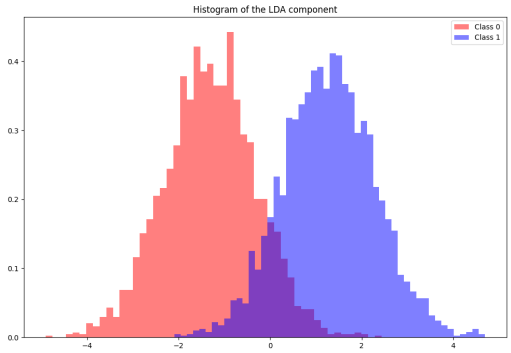

Figure 3.2: Histogram of LDA projection of data.

After applying LDA the classes overlap only on a partial portion of the distribution. This is a good result since we want to have a clear separation between the two classes.
Compared to Figure 2.1 we can see that the LDA projection is able to separate the two classes better than the original features.

In the following part, we will attempt our first classification, using LDA as a classifier.
We first divide our training dataset in a training and validation part. From now on, if not stated otherwise, we will always do a single split with a training/validation ratio of 2:1.

Here is the histogram plot of the validation data and the threshold given by the average of the projected class means:

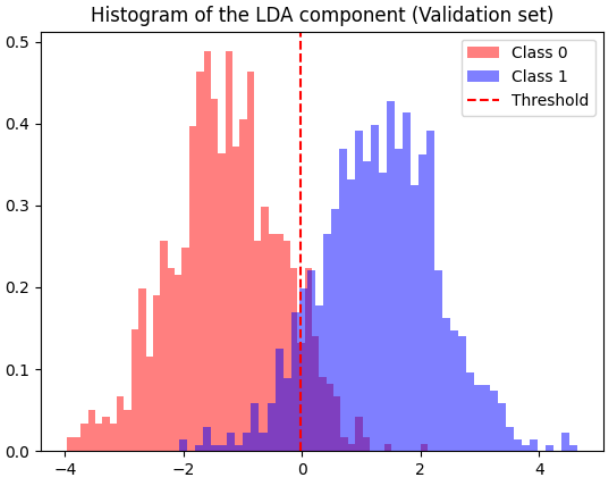

Figure 3.3: Histogram of LDA projection of data and simple threshold.

Here instead we use a optimized threshold, calculated by iterating over all the possible values ad finding the best, that minimizes the error rate:

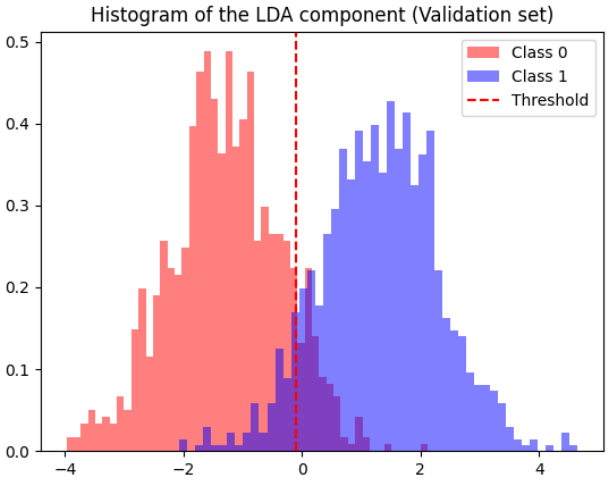

Figure 3.4: Histogram of LDA projection of data and optimized threshold.

As we can see the two thresholds are not too far apart. We can expect the errors rates to be similar for both thresholds.
The error rates are presented in this table:

	Simple Threshold	**Optimized Threshold**
Error Rate	9.10%	9.10%

Table 3.1: Error rates for LDA classifier.

We can improve the classification by applying preprocessing to the data by means of PCA. As before we are going to try using first the simple threshold and then the optimized threshold for the classification.
The final results are presented in this table:

PCA m components	**Simple Threshold**	**Optimized Threshold**
m=1	90.65%	55%
m=2	9.25%	**8.95%**
m=3	9.25%	9.15%
m=4	9.25%	9.15%
m=5	9.30%	9.05%
m=6	9.30%	9.10%

Table 3.2: Error rates for LDA classifier with PCA preprocessing.

The best results in terms of error rate are obtained by using 2 PCA components and the optimized threshold, giving a lower error rate than the LDA classifier without PCA preprocessing.

4. Training & Validation

4.1 Introduction

The main goal of this part is to train and validate the classifiers on the training dataset.
The classifiers we are going to use are:

- Gaussian Classifiers
  - Multivariate Gaussian (MVG)
  - MVG with Naive Bayes assumption (Naive MVG)
  - MVG with diagonal covariance matrix (Tied MVG)
- Logistic Regression (LR)
- Support Vector Machine (SVM)
- Gaussian Mixture Model (GMM)

As previously stated, we will use a single split of the training dataset in a training and validation part, with a ratio of 2:1.
In the beginning we will validate the classifiers based on the error rates they produce. We will then use a more complex metric, the Detection Cost Function (DCF), to evaluate the classifiers.
The DCF is a metric that takes into account the cost of false positives and false negatives.
We define the un-normalized DCF as:

$$DCF_u = \pi_T C_{fn} P_{fn} + (1 - \pi_T) C_{fp} P_{fp} \tag{4.1}$$

Where π_T is the prior probability of the target class (1), C_{fn} is the cost of a false negative, P_{fn} is the false negative rate ($P_{fn} = \frac{TN}{FN+TP}$), C_{fp} is the cost of a false positive, and P_{fp} is the false positive rate ($P_{fp} = \frac{FP}{FP+TN}$).
In particular we notice that the parameter π_T, C_{fn}, and C_{fp} depend only on

the application we are considering. We can summarize these parameters by computing what is called the effective prior:

$$\tilde{\pi} = \frac{\pi_T C_{fn}}{\pi_T C_{fn} + (1 - \pi_T) C_{fp}} \tag{4.2}$$

We can thus rewrite the un-normalized DCF as:

$$DCF_u = \tilde{\pi} P_{fn} + (1 - \tilde{\pi}) P_{fp} \tag{4.3}$$

To have a more meaningful metric we can normalize the DCF by the DCF of a dummy classifier:

$$DCF = \frac{DCF_u(\pi_T, C_{fn}, C_{fp})}{min(\pi_T C_{fn}, (1 - \pi_T) C_{fp})} \tag{4.4}$$

The normalized DCF is invariant to scaling, so it has the same value whether we use or no we use the effective prior.

Now that we have clearly defined the framework we are going to use, we can proceed with the training and validation of the classifiers.

4.2 Multivariate Gaussian Classifier

Here are the results of the Gaussian classifiers and LDA classifier in terms of error rate:

Classifier	Error Rate
MVG	**7.00%**
Naive MVG	7.20%
Tied MVG	9.30%
LDA	9.10%

Table 4.1: Error rates for MVG classifiers and LDA classifier.

The MVG classifier performs the best in terms of error rate, followed by the Naive MVG classifier. Both the LDA and Tied MVG classifiers have a higher

error rate than the other models. This could suggest that the features are somewhat independent.

To confirm this hypothesis we can compute the correlation matrix of the features:

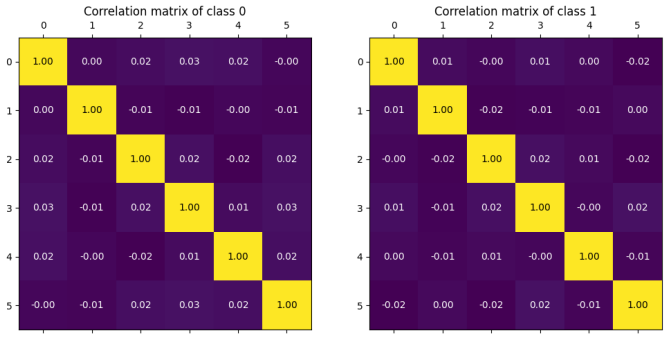

Figure 4.1: Correlation matrix of the features.

As we can see the features are not correlated: the values of covariance among the different features are close to 0, confirming our hypothesis. We can thus see why the Naive MVG classifier performs better, since this classifier assumes that the features are independent.

As discussed in section 2.2, features 3 and 4 are the most promising for a gaussian model, while the others will presumably need a more complex model to classify the samples.

We will now try to classify the samples using a subset of features.
We will start by using the first 4 features:

Classifier	Error Rate
MVG	7.95%
Naive MVG	**7.65%**
Tied MVG	9.50%
LDA	9.15%

Table 4.2: Error rates for MVG classifiers and LDA classifier using the first 4 features.

We observe that the performances are roughly the same (but slightly worse) than using all the features. This indicates that features 5 and 6 are not degrading the performances of the classifiers, but at the same time they offer few relevant information for the classification.
We'll proceed by classifying the samples using features 1 and 2 and then features 3 and 4 jointly.

Classifier	Error Rate
MVG	36.50%
Naive MVG	**36.30%**
Tied MVG	49.45%
LDA	44.45%

Table 4.3: Error rates for MVG classifiers and LDA classifier using features 1 and 2.

Classifier	Error Rate
MVG	9.45%
Naive MVG	9.40%
Tied MVG	9.45%
LDA	**9.10%**%

Table 4.4: Error rates for MVG classifiers and LDA classifier using features 3 and 4.

These results confirm our previous analysis. Features 1 and 2 are not able to separate the two classes, while features 3 and 4 are able to do so in a more reliable way.

In particular, we can see that using features 1 and 2 the MVG and Naive MVG classifiers have the lowest error rate (for the naive case this can be explained by the low correlation between features).
The Tied MVG model performs the worst as expected. This is due to the fact that this model assumes that the covariance matrix is the same for both classes, but as discussed in section 2.2 the two classes have different variance for features 1 and 2.
A different outcome is observed for features 3 and 4: they have similar variance but different mean. Moreover these features have a weak correlation. All these factors can explain the good performances of the MVG classifiers wrt only using features 1 and 2.

As a final step we will try to classify the samples using PCA as a preprocessing step.

PCA m components	MVG	Naive MVG	Tied MVG	LDA
m=1	9.25%	9.25%	9.35%	55.00%
m=2	8.80%	8.85%	9.25%	8.95%
m=3	8.80%	9.00%	9.25%	9.15%
m=4	8.05%	8.85%	9.25%	9.15%
m=5	7.10%	8.75%	9.30%	9.05%
m=6	**7.00%**	8.90%	9.30%	9.10%

Table 4.5: Error rates for MVG classifiers and LDA classifier with PCA preprocessing.

PCA doesn't seem to be effective with this dataset. The best results are obtained by using 6 PCA components, but the error rate is the same than using the original features.

After this preliminary analysis we can say that the best model is the MVG classifier (without preprocessing of data) with an error rate of 7.00%.

Detection Cost Function analysis

As discussed before, error rate can sometimes be misleading. We will now use a more robust metric, the DCF, to evaluate the MVG classifiers.

We will start by using 5 different applications with different values of π_T, C_{fn}, and C_{fp} and compare them in terms of effective prior.

- Application 1: $\pi_T = 0.5$, $C_{fn} = 1$, $C_{fp} = 1$ (effective prior = 0.5)
- Application 2: $\pi_T = 0.9$, $C_{fn} = 1$, $C_{fp} = 1$ (effective prior = 0.9)
- Application 3: $\pi_T = 0.1$, $C_{fn} = 1$, $C_{fp} = 1$ (effective prior = 0.1)
- Application 4: $\pi_T = 0.5$, $C_{fn} = 1$, $C_{fp} = 9$ (effective prior = 0.1)
- Application 5: $\pi_T = 0.5$, $C_{fn} = 9$, $C_{fp} = 1$ (effective prior = 0.9)

We observe that the effective prior is a good metric to summarize the costs of mis-classifications. In fact we can see that the cost of mis-classifications is reflected in the prior: stronger security (higher false positive cost) corresponds to an equivalent lower prior probability of a legit user.
That's why from now on we will use the effective prior (with mis-classification costs equal to 1) to evaluate the DCF of the classifiers.

Eff Prior	MVG	Naive MVG	Tied MVG
0.5	0.140 / **0.130** (7.0%)	0.144 / 0.131 (8.9%)	0.186 / 0.181 (2.6%)
0.9	0.400 / 0.342 (14.4%)	0.389 / 0.351 (9.8%)	0.463 / 0.442 (4.4%)
0.1	0.305 / 0.263 (13.8%)	0.302 / 0.257 (15.0%)	0.406 / 0.363 (10.6%)

Table 4.6: DCF/minDCF (percentual difference) for different models and effective priors (no PCA).

PCA	MVG	Naive MVG	Tied MVG
1	0.185 / 0.177 (4.4%)	0.185 / 0.177 (4.4%)	0.187 / 0.177 (5.4%)
2	0.176 / 0.173 (1.6%)	0.177 / 0.171 (3.4%)	0.185 / 0.179 (3.3%)
3	0.176 / 0.173 (1.4%)	0.180 / 0.175 (2.9%)	0.185 / 0.183 (1.1%)
4	0.161 / 0.154 (4.5%)	0.177 / 0.172 (3.0%)	0.185 / 0.182 (1.6%)
5	0.142 / 0.133 (6.2%)	0.175 / 0.174 (0.7%)	0.186 / 0.181 (2.6%)
6	0.140 / **0.130** (7.0%)	0.178 / 0.173 (3.0%)	0.186 / 0.181 (2.6%)

Table 4.7: DCF/minDCF (percentual difference) with prior 0.5 and PCA.

PCA	MVG	Naive MVG	Tied MVG
1	0.397 / 0.369 (7.1%)	0.397 / 0.369 (7.1%)	0.402 / 0.369 (8.3%)
2	0.388 / 0.353 (9.1%)	0.387 / 0.356 (7.9%)	0.396 / 0.363 (8.3%)
3	0.388 / 0.356 (8.1%)	0.395 / 0.365 (7.7%)	0.408 / 0.368 (9.8%)
4	0.353 / 0.301 (14.6%)	0.397 / 0.361 (9.0%)	0.403 / 0.361 (10.4%)
5	0.304 / 0.274 (10.0%)	0.393 / 0.354 (9.8%)	0.405 / 0.365 (10.0%)
6	0.305 / **0.263** (13.8%)	0.392 / 0.353 (9.8%)	0.406 / 0.363 (10.7%)

Table 4.8: DCF/minDCF (percentual difference) with prior 0.1 and PCA.

PCA	MVG	Naive MVG	Tied
1	0.478 / 0.434 (9.4%)	0.478 / 0.434 (9.4%)	0.481 / 0.434 (9.7%)
2	0.443 / 0.438 (1.1%)	0.442 / 0.432 (2.3%)	0.479 / 0.435 (9.1%)
3	0.468 / 0.439 (6.1%)	0.459 / 0.434 (5.4%)	0.457 / 0.434 (4.9%)
4	0.460 / 0.415 (9.7%)	0.463 / 0.431 (6.8%)	0.462 / 0.444 (3.8%)
5	0.398 / 0.351 (11.7%)	0.466 / 0.434 (6.9%)	0.463 / 0.445 (3.8%)
6	0.400 / **0.342** (14.4%)	0.451 / 0.436 (3.4%)	0.463 / 0.442 (4.4%)

Table 4.9: DCF/minDCF (percentual difference) with prior 0.9 and PCA.

Comparing the model in terms of minDCF we can see that the MVG classifier performs the best, in almost all the cases, even with different effective priors and PCA preprocessing.

To check if the if the models are well calibrated we can plot the Bayes error plots for the MVG classifier. In the following we will use the setup that gave the best results in terms of minDCF for an effective prior of 0.1 (**this will be from now on the target application we will consider**).

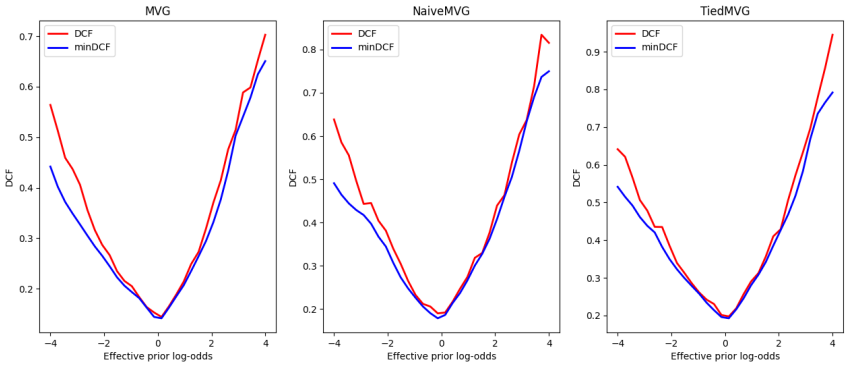

Figure 4.2: Bayes error plot for the MVG classifier.

From this figure we can see that the MVG classifiers are well calibrated in the middle part of the plot, while they show signs of little mis-calibration in the extremes of the plot.

4.3 Logistic Regression Classifier

The second model we are going to use is the binary Logistic Regression (LR) classifier.

In the following section we will present the results of the Logistic Regression classifier in terms of DCF and minDCF, using different setups.

We will start by using the target application (effective prior = 0.1) and no PCA preprocessing.

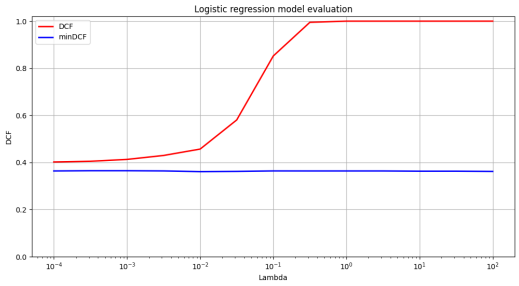

Figure 4.3: DCF and minDCF for the Logistic Regression classifier as a function of λ

We observe that the DCF increases with λ (the regularization degrades our results), while the minDCF remains quite stable.

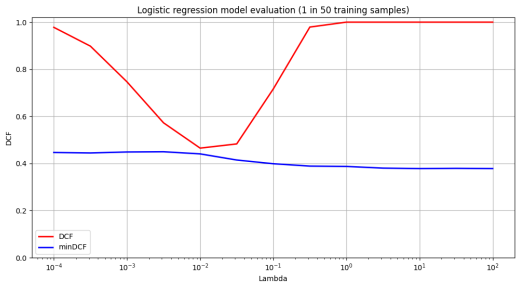

Figure 4.4: DCF and minDCF for the Logistic Regression classifier as a function of λ (1 in 50 samples taken)

Removing samples seems to be an effective way to improve the regularization. We can in fact see that now the minDCF decreases with λ, indicating that the regularization is now effective in improving the classifier.
Moreover we can see that for $\lambda = 0.01$ our classifier is able to achieve a DCF that is similar to the minDCF, which is a good result.

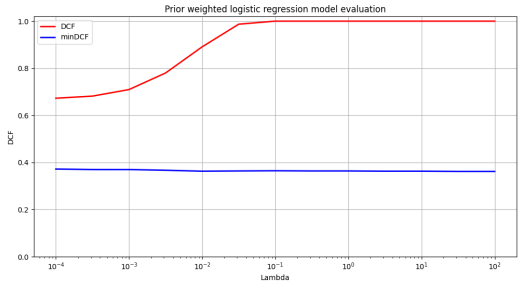

Figure 4.5: DCF and minDCF for the Prior Weighted Logistic Regression classifier as a function of λ (prior = 0.1)

We don't observe significant improvements in the minDCF using the prior weighted Logistic Regression classifier. There are though some improvements in the DCF, when using priors like 0.6, 0.7, and 0.8.

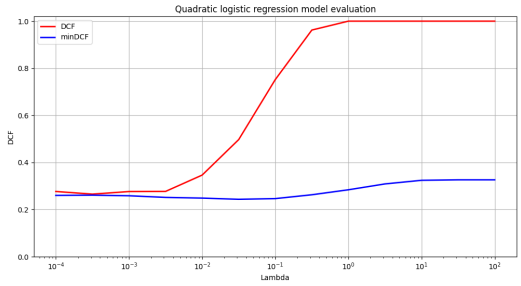

Figure 4.6: DCF and minDCF for the Quadratic Logistic Regression classifier as a function of λ

The Quadratic Logistic Regression classifier is the best model until now in terms of minDCF, with a lowest value of around 0.210 for $\lambda = 0.032$.
This can be explained by the fact that the Quadratic Logistic Regression classifier is able to model the data in a more complex way, and thus is able to achieve better results.
We can also see that a "medium" regularization is the best for this model.

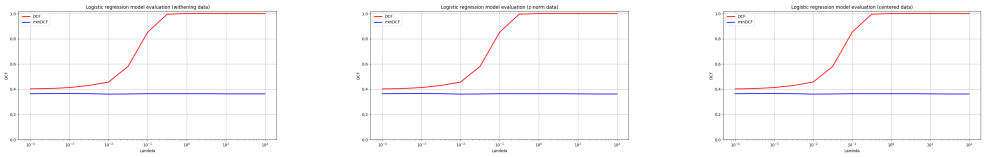

Figure 4.7: DCF and minDCF for the Logistic Regression classifier with different preprocessing steps.

In these plots we can see that using techniques like whitening, z-normalization, and centering on the standard LR model doesn't improve the performances. This is probably due to the fact that the original data is already well scaled and centered.

As a final step we will try to classify the samples using PCA as a preprocessing step.

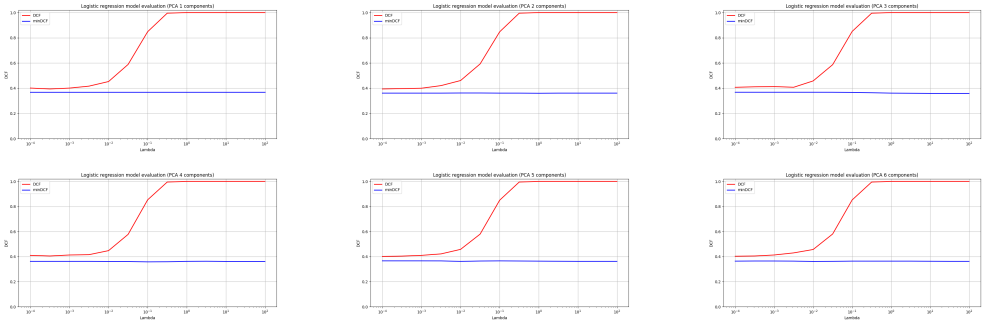

Figure 4.8: DCF and minDCF for the Logistic Regression classifier with PCA preprocessing.

We can see that also in this case PCA doesn't seem to be effective in improving the performances of the Logistic Regression classifier.

We can end our analysis of the Logistic Regression classifier by observing that all the different approach that we have used suffer from the same problem: heavy mis-calibration.

Moreover we can see that, up to now, the Quadratic Logistic Regression classifier is the best model in terms of minDCF, with a value of around

0.210, beating the MVG classifier with the best setup (0.263).
The gap in performances can be explained by the fact that the Quadratic Logistic Regression classifier is able to model the data that is not linearly separable in a better way than the MVG classifier. Observing the scatter plots of the features we can see that most of them are not linearly separable, thus the Quadratic Logistic Regression classifier is able to achieve better results.

4.4 Support Vector Machine Classifier

The third model we are going to use is the Support Vector Machine (SVM) classifier.
This is a powerful model that is able to classify samples both in a linear and non-linear way (with the use of the kernel trick).

We will start with the simple SVM classifier and then we will use the SVM with polynomial and RBF kernels.

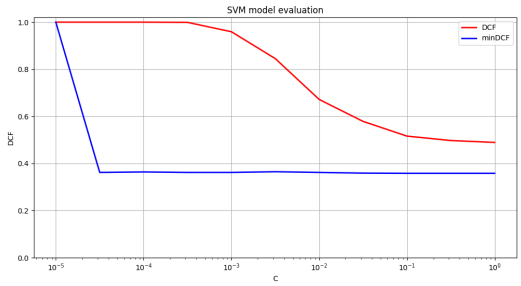

Figure 4.9: DCF and minDCF for the SVM classifier as a function of the regularization parameter.

As we can see the results obtained by this model are not good compared to the other models. This is another strong indication that the data is not linearly separable.
Moreover in this case lower regularization is better (high values of the parameter C), indicating that the model is not overfitting the data. Both the DCF and minDCF decrease with the regularization parameter.

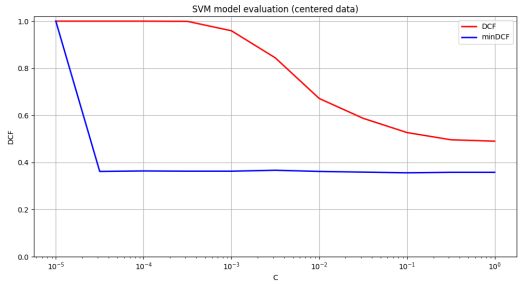

Figure 4.10: DCF and minDCF for the SVM classifier as a function of the regularization parameter (centered data).

Centering the data doesn't seem to improve the performances of the SVM classifier. The results are similar to the ones obtained with the original data, confirming that the data is already well centered.
The same can be said about the regularization parameter: the best results are obtained with low regularization.

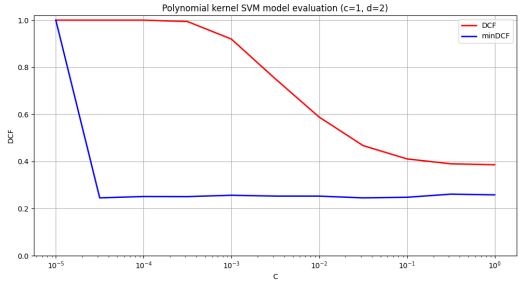

Figure 4.11: DCF and minDCF for the SVM with polynomial kernel classifier as a function of the regularization parameter.

The SVM with polynomial kernel is able to achieve better results than the simple SVM classifier. We can see a huge improvement in the minDCF, which is now around 0.210 for the best setup.
Here we used a polynomial kernel of degree 2, which is able to model the data in a better way than the linear kernel.
The low regularization trend is confirmed also in this case.

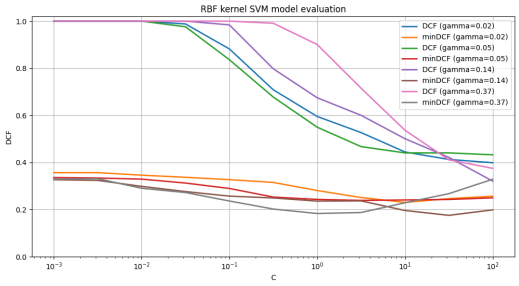

Figure 4.12: DCF and minDCF for the SVM classifier with RBF kernel as a function of the regularization parameter.

Using the RBF kernel we are able to achieve the best results until now. The minDCF is lower than 0.200 for the best setup, which is the best result obtained until now.
This can be explained by the fact that the RBF kernel is able to model the data in a more complex way, confirming again the non linear separability of the data.

To conclude the analysis of the SVM classifier we can say that the best model is the SVM with RBF kernel, with the best setup being $C = 32$ and $\gamma = 0.135$.
Moreover all the models suffer from mis-calibration, and they would greatly benefit from a calibration step.

4.5 Gaussian Mixture Model Classifier

The fourth and last model we are going to use is the Gaussian Mixture Model (GMM) classifier.

In the following we present the results of the GMM classifier in terms of DCF and minDCF, using different setups.

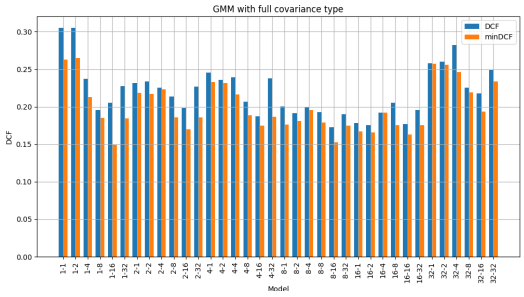

Figure 4.13: DCF and minDCF for the GMM classifier as a function of the number of components.

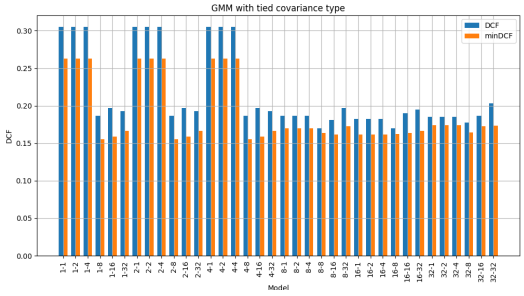

Figure 4.14: DCF and minDCF for the GMM (tied covariance matrix) classifier as a function of the number of components.

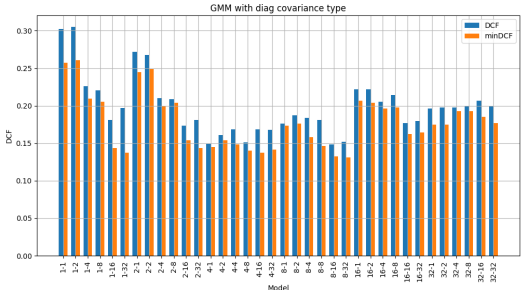

Figure 4.15: DCF and minDCF for the GMM (diagonal covariance matrix) classifier as a function of the number of components.

The GMM classifier is in general able to achieve good results. We could have

expected this since GMM is a more complex model than the other models we have used, and as we said before is a good candidate to model some of the features of our dataset.

The best results are obtained with the diagonal covariance matrix GMM with 8 components for class 0 and 32 components for class 1. The minDCF is around **0.130**, which is the best result obtained until this moment.
These results are in line with the analysis we have done. In particular we have seen that the data is not linearly separable, and that the features are not correlated (even tough in the diagonal covariance matrix case we are doing a different assumption wrt to the Naive assumption).

4.6 Model Comparison

We can now compare the models in terms of minDCF, to see which one is the best model for our dataset (if we selected the optimal threshold).
The three best models that we will consider are:

- Quadratic Logistic Regression classifier ($\lambda = 0.032$)
- SVM with RBF kernel ($C = 32$, $\gamma = 0.135$)
- GMM with diagonal covariance matrix (8 components for class 0, 32 components for class 1)

We will plot the Bayes error plots and analyze them.

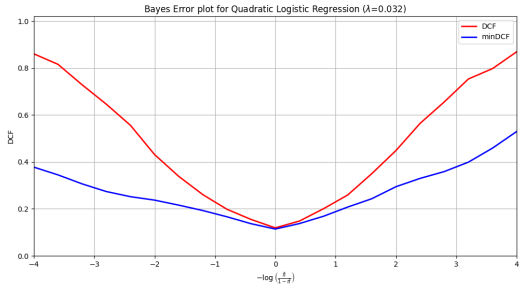

Figure 4.16: Bayes error plot for the Quadratic Logistic Regression classifier.

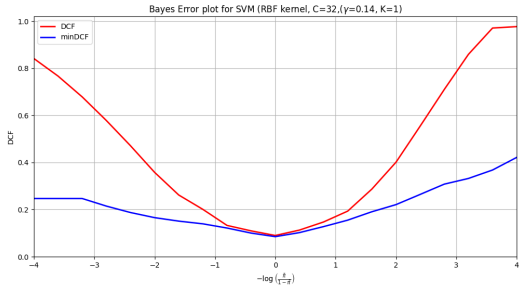

Figure 4.17: Bayes error plot for the SVM with RBF kernel classifier.

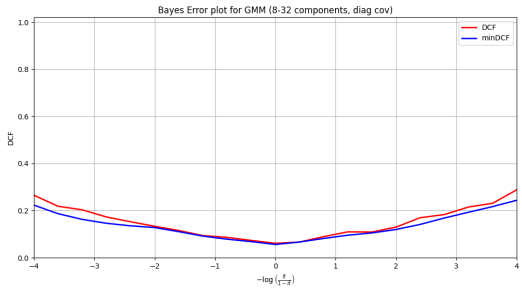

Figure 4.18: Bayes error plot for the GMM with diagonal covariance matrix classifier.

We can see that the relative ranking of the models is preserved even if we consider other priors: the GMM classifier is the best model, followed by the SVM with RBF kernel, and then the Quadratic Logistic Regression classifier.

We can also observe that the SVM model and the QLR model are not well calibrated, while the GMM model is well calibrated across the whole range of priors.
In particular these models (SVM and QLR) harm priors that are not close to 0.5. This is an important consideration since our application has a prior of 0.1.

4.7 Score Calibration

As we have seen in the previous sections, the models we have used are not well calibrated. This is a problem since the DCF is a metric that is sensitive to the calibration of the models.
This is due to the fact that until now we have used as threshold the theoretical threshold ($-log\frac{\pi}{1-\pi}$) that is not always optimal. That's why we see huge differences in the DCF and minDCF values, especially for our target application.

In this section we will try to calibrate the models using the Prior Weighted Logistic Regression as a calibration model.
To train the calibration we will use a 5-fold cross validation, using the previously computed scores as inputs. This will allow us to see which hyper parameter (training prior) is the best.

Quadratic Logistic Regression

We will start by calibrating the Quadratic Logistic Regression classifier.

In the following table we report the minDCF, DCF, and calDCF for different training priors, considering our target application (effective prior = 0.1).

Training Prior	minDCF	DCF	calDCF
0.1	0.244	0.496	0.268
0.2	0.244	0.496	0.265
0.3	0.244	0.496	0.276
0.4	0.244	0.496	0.251
0.5	0.244	0.496	0.281
0.6	0.244	0.496	0.254
0.7	0.244	0.496	**0.248**
0.8	0.244	0.496	0.258
0.9	0.244	0.496	0.256

Table 4.10: Quadratic Logistic Regression: minDCF, DCF, and calDCF for different training priors

We obtain the best results for training priors of 0.7, with a calDCF of 0.248.

In the following figure we have plotted the Bayes error plot for the Quadratic Logistic Regression classifier after the calibration step, with the calibration model trained with a prior of 0.7.

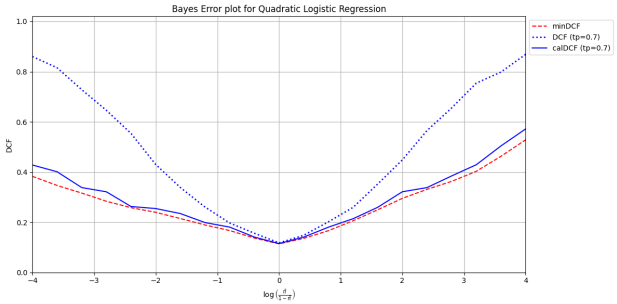

Figure 4.19: Calibration of the Quadratic Logistic Regression classifier (training prior = 0.7).

As we can see in this figure the calibration is able to drastically improve the performances of the model over the whole range of priors.

SVM with RBF Kernel

Training Prior	minDCF	DCF	calDCF
0.1	0.175	0.422	**0.179**
0.2	0.175	0.422	0.181
0.3	0.175	0.422	0.186
0.4	0.175	0.422	0.188
0.5	0.175	0.422	0.200
0.6	0.175	0.422	0.194
0.7	0.175	0.422	0.193
0.8	0.175	0.422	0.197
0.9	0.175	0.422	0.208

Table 4.11: RBF SVM: minDCF, DCF, and calDCF for different training priors

Here we obtain the best results for a training prior of 0.1, with a calDCF of 0.179.

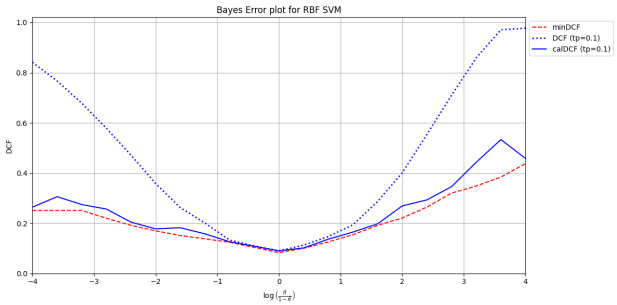

Figure 4.20: Calibration of the SVM with RBF kernel classifier (training prior = 0.1).

Following the same approach as before, we have plotted the Bayes error plot for the SVM with RBF kernel. We can see that the calibration is able to improve the performances of the model, especially in the extremes of the plot.

GMM

Finally we have calibrated the GMM classifier. The results are presented in the following table.

Training Prior	minDCF	DCF	calDCF
0.1	0.131	0.152	0.162
0.2	0.131	0.152	0.150
0.3	0.131	0.152	0.154
0.4	0.131	0.152	0.154
0.5	0.131	0.152	0.150
0.6	0.131	0.152	0.152
0.7	0.131	0.152	0.160
0.8	0.131	0.152	0.159
0.9	0.131	0.152	**0.148**

Table 4.12: GMM (8-32 components): minDCF, DCF, and calDCF for different training priors

Here we obtain the best results for a training prior of 0.9, with a calDCF of 0.148.

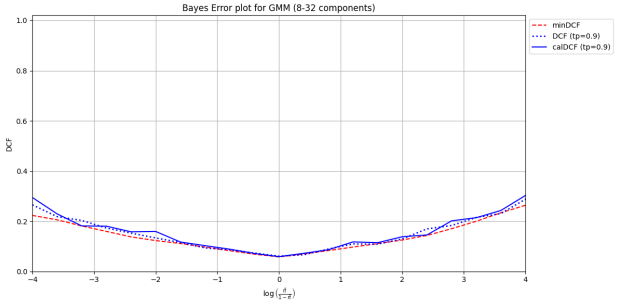

Figure 4.21: Calibration of the GMM with diagonal covariance matrix classifier (training prior = 0.9).

We can see that the original model was already well calibrated, and the calibration step didn't improve the performances of the model by much. For some priors the calibration step even worsened the performances of the model.

Even after the calibration step the GMM classifier is the best model, followed by the SVM with RBF kernel, and then the Quadratic Logistic Regression classifier.

4.8 Model Fusion

In this section we will try to fuse the models we have used in order to improve the performances of the classifiers.
We will use a technique called "score fusion" that is able to combine the scores of the models in order to obtain a different classification.

Here are the results of the fused models in terms of minDCF, DCF for our target application (effective prior = 0.1).

Model	minDCF	DCF
QLR + RBF SVM	0.216	0.459
QLR + GMM	0.217	0.324
RBF SVM + GMM	**0.185**	0.287
QLR + RBF SVM + GMM	0.214	0.356

Table 4.13: Fused models: minDCF and DCF for different combinations of models

We can see that the best model is the one that fuses the RBF SVM and the GMM classifier, with a minDCF of 0.185. We can observe that also this fused models suffer from mis-calibration (the DCF is quite higher than the minDCF for our target application).

Again, we applied score calibration with different priors for the training of the calibration model. In order to find the best calibration we have used a 5-fold cross validation on the validation set.
The results are presented in the following tables.

Training Prior	minDCF	calDCF
0.1	0.179	0.200
0.2	0.175	0.198
0.3	0.177	0.195
0.4	0.174	0.200
0.5	0.177	0.204
0.6	0.172	0.196
0.7	0.181	0.198
0.8	0.178	0.197
0.9	0.175	**0.191**

Table 4.14: QLR + RBF SVM: minDCF, DCF, and calDCF for different training priors

Training Prior	minDCF	calDCF
0.1	0.124	0.162
0.2	0.129	0.145
0.3	0.127	**0.144**
0.4	0.135	0.155
0.5	0.136	0.150
0.6	0.133	0.152
0.7	0.132	0.151
0.8	0.126	0.151
0.9	0.147	0.157

Table 4.15: QLR + GMM: minDCF, DCF, and calDCF for different training priors

Training Prior	DCF	calDCF
0.1	0.136	0.163
0.2	0.125	0.158
0.3	0.132	0.163
0.4	0.134	0.151
0.5	0.131	0.162
0.6	0.125	0.153
0.7	0.137	**0.150**
0.8	0.133	0.151
0.9	0.125	0.155

Table 4.16: RBF SVM + GMM: minDCF, DCF, and calDCF for different training priors

Training Prior	minDCF	calDCF
0.1	0.128	0.144
0.2	0.134	0.165
0.3	0.126	0.144
0.4	0.126	**0.142**
0.5	0.144	0.172
0.6	0.134	0.152
0.7	0.138	0.164
0.8	0.130	0.151
0.9	0.141	0.153

Table 4.17: QLR + RBF SVM + GMM: minDCF, DCF, and calDCF for different training priors

The best results are obtained by fusing all the models together. The best results are obtained for a training prior of 0.4, with a calDCF of 0.142.
Even if we have found an improvement, we have to consider that the GMM model alone is able to achieve similar results (calDCF of 0.148), and thus the fusion of the models is not strictly necessary.

We will however consider the fused model as the best model for our application (delivered model), and we will use it to classify the samples in the test set.

5. Experimental Results

In this chapter we will present the results of the models we have previously trained, and see how they perform on the test set.

We will consider the delivered system and the single models (QLR, RBF SVM, GMM) as well as the fused models.
All these models are trained using the training part of our original dataset (that we remember being 2/3 of the original dataset), and we will use them to classify the samples in the test set.

We will present the results in terms of minDCF, DCF, and calDCF, and we will compare them with the results obtained on the training set.

Delivered System

We will start by presenting the results of the delivered system on the test set.
Here is are the minDCF and DCF values obtained using the system for our target application, as well as the Bayes error plot for different priors.

minDCF	**DCF**
0.282	0.363

Table 5.1: Delivered System: minDCF and DCF for the test set (Application Prior = 0.1)

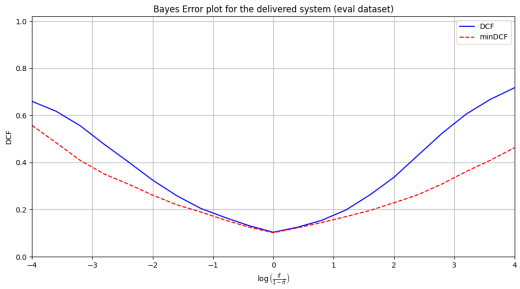

Figure 5.1: Bayes error plot for the delivered system on the test set.

We can see that the delivered system is not well calibrated, and the DCF is higher than the minDCF for our target application and for priors at the extremes of the plot.

Single and Fused Models

We will now present the results of the single models, as well as the fused ones, on the test set.

Model	**DCF**
QLR	0.494
RBF SVM	0.402
GMM	**0.195**
QLR + RBF SVM	0.448
QLR + GMM	0.344
RBF SVM + GMM	0.299
QLR + RBF SVM + GMM (Delivered System)	0.363

Table 5.2: Model Performance for Application Prior 0.1: DCF

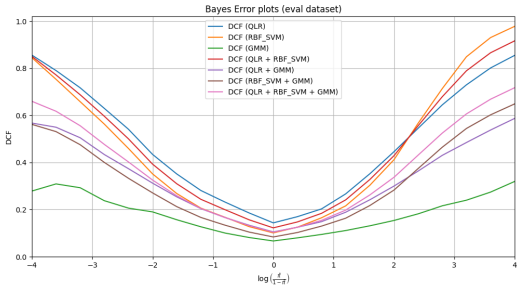

Figure 5.2: Bayes error plot for the single models and their fusion on the test set.

We can see that the GMM model is the best model in terms of DCF, with a value of 0.195. Our delivered system was not the best choice possible, and we could have obtained better results by using the GMM model alone.
After all, the fusion of the models was not strictly necessary, and the GMM model was also able to achieve similar results during the training phase.

Calibration

We have seen that the delivered model is not well calibrated. Given this, we will now try to calibrate the models using the Prior Weighted Logistic Regression.
In this phase we are going to use the validation part of the original dataset to train our calibration model, using the best training prior we have found during the training phase.
We will present the results obtained after the calibration step for the single and the fused models.

Model	minDCF	calDCF
QLR	0.351	0.370
RBF SVM	0.262	0.288
GMM	0.183	0.210
QLR + RBF SVM	0.258	0.282
QLR + GMM	0.183	0.211
RBF SVM + GMM	0.181	**0.208**
QLR + RBF SVM + GMM (Delivered System)	0.181	**0.208**

Table 5.3: Calibration of the single models on the test set (application prior = 0.1)

We can see that the calibration step is able to improve the performances of the models, especially for the QLR and SVM models. As in the training phase, the GMM model is already well calibrated and the calibration step seems to worsen the performances of the model.
Similarly to what we found during the training phase, the 3-fused model is the best model in terms of DCF, followed closely by the GMM.

Here instead are the Bayes error plots for the models after the calibration step.

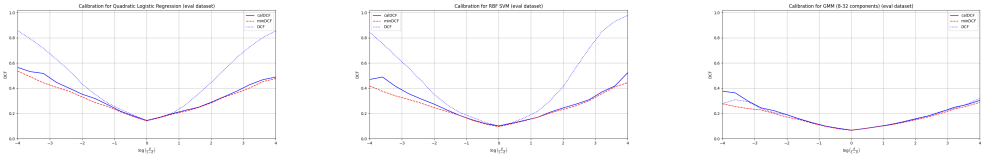

Figure 5.3: Bayes error plot for the single models after the calibration step on the test set.

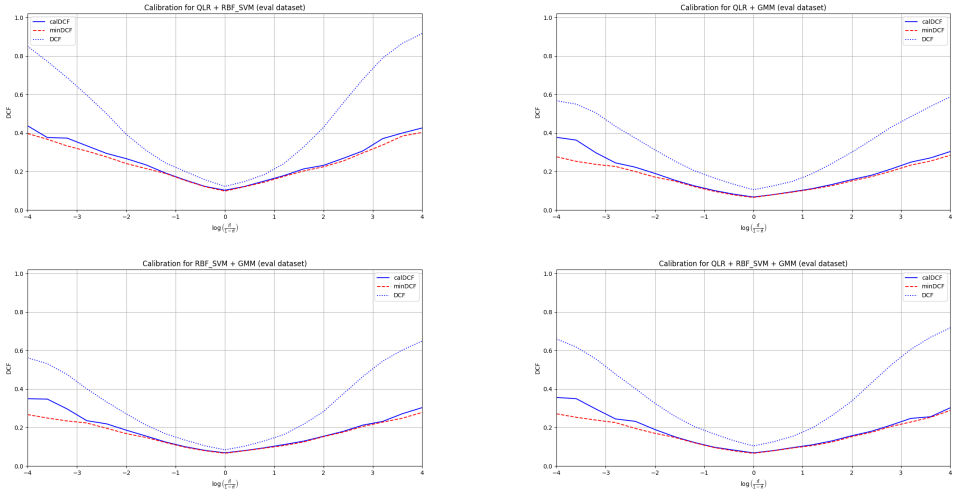

Figure 5.4: Bayes error plot for the fused models after the calibration step on the test set.

What we have said for the target application is confirmed also for other priors: calibration generally improves the performances of our models.

After the calibration step the 3-fused model is the best model in terms of DCF, with a value of 0.208, followed by the GMM model with a DCF of 0.210. This confirms what we have seen during the training phase.

Analysis of the training strategy

In this last section we are going to use the test set to analyze the training strategy we have used and see if it was effective in improving the performances of the models.
We are only going to consider the GMM model, but this analysis could be extended to the other models as well.

We will consider the following training strategies:

- Train the models using the training part of the original dataset
- Repeating the training step for GMM model with full, diagonal and tied covariance matrix, with a different number of components

- Use these models to classify the samples in the test set and calculate the resulting DCF and minDCF

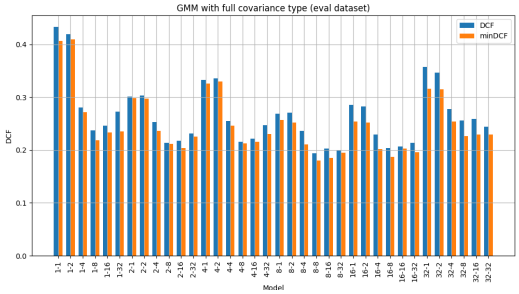

Figure 5.5: DCF and minDCF for the GMM classifier as a function of the number of components (eval dataset).

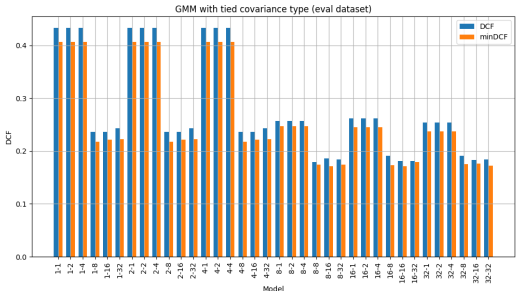

Figure 5.6: DCF and minDCF for the GMM (tied covariance matrix) classifier as a function of the number of components (eval dataset).

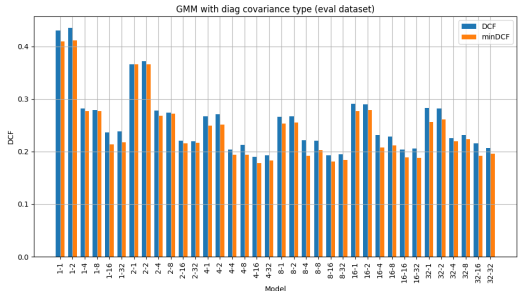

Figure 5.7: DCF and minDCF for the GMM (diagonal covariance matrix) classifier as a function of the number of components (eval dataset).

As we can see from the figures, the best results are obtained with the GMM model with diagonal covariance offers good performances, but we have to mention that the tied covariance matrix model performs surprisingly well compared to the training phase.
To conclude we can say that the training strategy we have used for the GMM model was effective in improving the performances.

6. Conclusions

In this project we have analyzed the performances of different models on a dataset of speech features. We have trained different models (Quadratic Logistic Regression, SVM with RBF kernel, Gaussian Mixture Model) and we have compared them in terms of DCF and minDCF.

The model we decided to deliver turned out to be good and the ideas we had about the dataset were confirmed by the results we obtained.
For what concerns the training strategy for the GMM model we can say that it was effective in improving the performances of the model.