MACHINE LEARNING AND PATTERN RECOGNITION REPORT

Fingerprint Spoofing Detection

Antonio Iorio
Who?
Where?
When?

Contents

Introduction The project task consists of a binary classification problem. The goal is to perform fingerprint spoofing detection, i.e. to identify genuine vs counterfeit fingerprint images. The dataset consists of labeled samples corresponding to the genuine (True, label 1) class and the fake (False, label 0) class. The samples are computed by a feature extractor that summarizes high-level characteristics of a fingerprint image. The data is 6-dimensional.

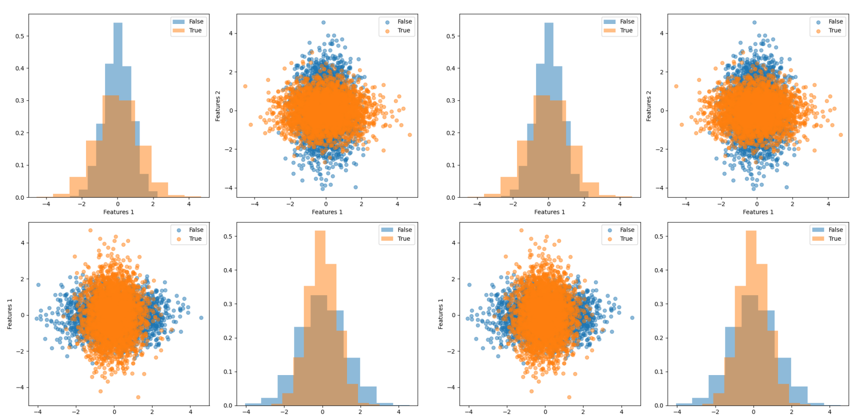
1 Dataset Analysis

In our analysis process, one first begins to represent what are the data related to the various features and how among the various features the data are distributed by making a visual representation in pairs of features.

1. Starting with the analysis of the first two features and creating a histogram and a scatter Figure 1, one can see:

    - Both features overlap
    - Follow a Gaussian distribution
    - Feature 1 has a peak at [-0.213, 0.276] and it is worth 0.541 for the false class, instead for Feature 2 the peak at [-0.402, 0.165] and it is worth 0.516 for the true class.

    Seeing Figure 1 again, it would appear that Figure 1a and Figure 1b appear to be visually the same but in b they are represented centred which as can be seen is quite similar because Feature 1 has $\mu = 0.00170711$ and $\sigma^2 = 1.00134304$, instead Feature 2 has $\mu = 0.00503903$ and $\sigma^2 = 0.9983527$



(a) Without mean (b) With mean

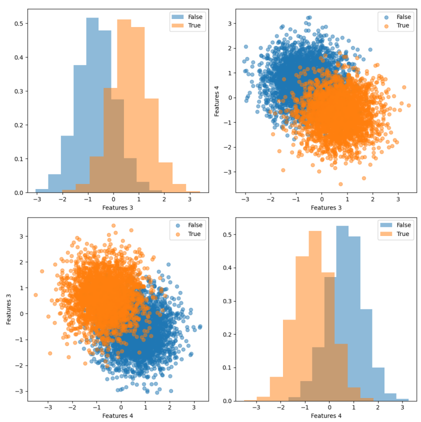
Figure 1: Feature 1 vs Feature 2 - Without centering the data relative to the average (a) and with centering the data relative to the average (b)

2. For features 3 and 4, observed in Figure 2, on the other hand, they have:

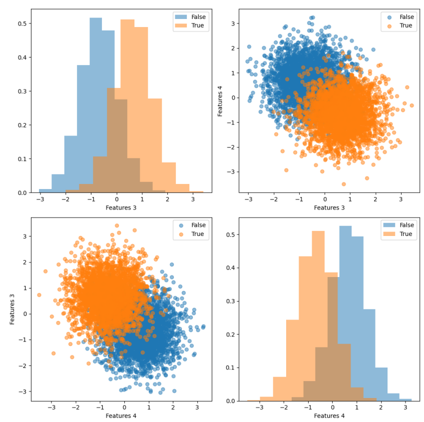
    - do not overlap like the previous two
    - Follow a Gaussian distribution but the true and false labels are centred at different points

- Feature 3 has a peak at [-1.063, -0.568] and it is worth 0.517 for the false class, instead for Feature 4 the peak at [0.290, 0.783] and it is worth 0.525 for the false class.

Seeing Figure 2a and Figure 2b, data are already similar because, the mean calculated with reference to the two classes is close to 0, in fact: Feature 3 has $\mu = -0.00560753$ and $\sigma^2 = 1.0024818$, instead Feature 4 has $\mu = 0.00109537$ and $\sigma^2 = 0.99029389$



(a) Without mean



(b) With mean

Figure 2: Feature 3 vs Feature 4 - Without centering the data relative to the average (a) and with centering the data relative to the average (b)

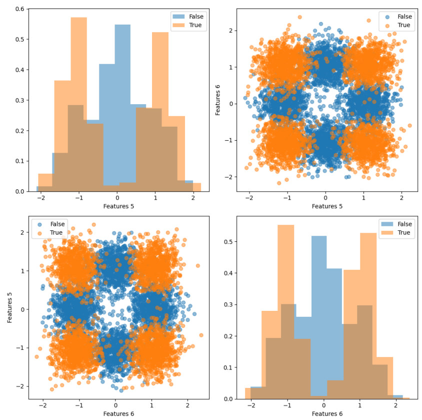
3. For features 5 and 6, observed in Figure 3, one can see:

   - Do not totally overlap
   - For both features, the true labels don't follow a Gaussian distribution as opposed to the false ones, which could be more approximate
   - Feature 5 has a peak at [-1.211, -0.783] and it is worth 0.572 for the true class, instead for Feature 6 has a peak at [-1.273, -0.817] and it is worth 0.553 for the true class.

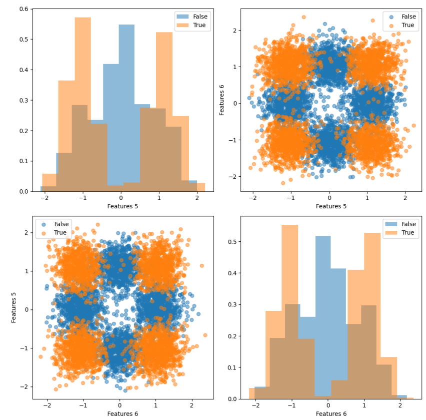
   Also in this other case, Figure 3a and Figure 3b are similar because again the average is close to 0. Feature 5 has $\mu = -0.00700025$ and Feature 6 has $\mu = 0.00910515$

2 Dimensionality Reduction

Before proceeding with classification, two techniques of dimensionality reduction PCA and LDA can be analysed. The goal is to find a subspace of the feature space that preserves most of the useful information, that is, mapping from the *n–dimensional* feature space to *m–dimensional* space, with $m \ll n$



(a) Without mean



(b) With mean

Figure 3: Feature 5 vs Feature 6 - Without centering the data relative to the average (a) and with centering the data relative to the average (b)

2.1 PCA

is an unsupervised technique. Where starting from a dataset $X = \{x_1, \dots, x_k\}$ and calculated average. It starts with the empirical covariance matrix:

$$C = \frac{1}{K} \sum (x_i - \bar{x})(x_i - \bar{x})^T \tag{1}$$

We compute the eigen-decomposition of $C = U \Sigma U^T$ and project the data in the subspace spanned by the m columns of U corresponding to the m largest eigenvalues.

$$y_i = P^T (x_i - \bar{x}) \tag{2}$$

where P is the matrix of the m columns of U associated to the m highest eigenvalues of C. A cross-validation approach can be used to figure out the optimal value of m to be selected. To evaluate each eigenvalue, one would have to calculate the variance corresponding to the axis. The percentage can be calculated as the rate between the sum of the m eigenvalues and the sum of all of them. In Figure 4 we can see how it changes in the project. A good m, corresponds to that value which allows a percentage greater than 95%, so in our case we would need all 6 features.

2.2 LDA

is a supervised technique. To find a direction that has the best separation between classes, we measure spread between classes in terms of class covariance. The objective is to maximize the $between - class$ variability over $within - class$ variability ratio for the transformed samples:

$$\max_w \frac{w^T S_B w}{w^T S_W w} \tag{3}$$

where:

$$S_B \triangleq \frac{1}{N} \sum_{c=1}^{K} n_c (\mu_c - \mu)(\mu_c - \mu)^T \tag{4}$$

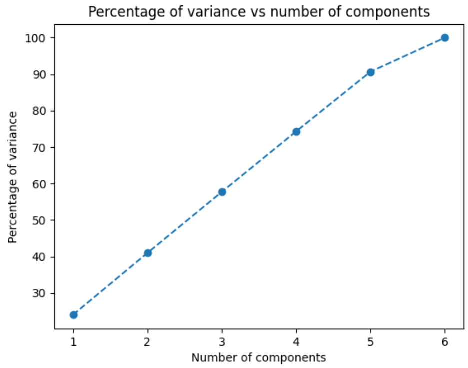


Figure 4: Cross validation for PCA impact evaluation

$$S_W \triangleq \frac{1}{N}\sum_{c=1}^{K}\sum_{i=1}^{n_c}(x_{c,i}-\mu_c)(x_{c,i}-\mu_c)^T \tag{5}$$

μ is dataset mean μ_c is class mean

2.3 Our project

PCA and LDA are applied to the dataset, in particular, m = 6 is used, and in Figure 5 we can observe what are the outcomes for the indicated directions.

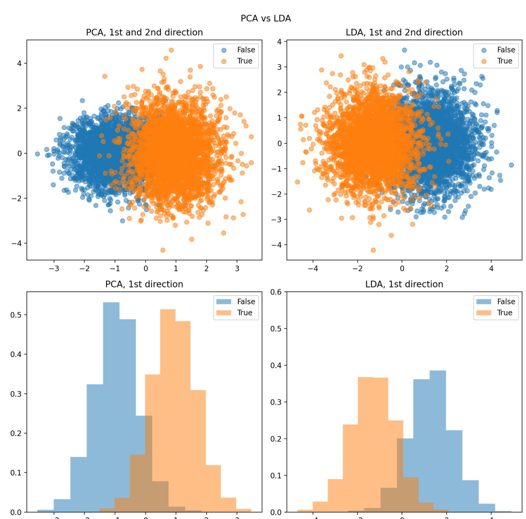


Figure 5: Comparing resulta between PCA and LDA

At a later stage, they were used to carry out a classification. The available dataset was divided into two sub-portions one for training and the other for validation. In Table 1 we can see the error of the classification, errors made in the classification when varying m (only for some values) and the threshold were reported

Method	Num Samples	Error	Error Rate (%)
LDA - First threshold	2000	186	9.30
LDA - Second threshold	2000	186	9.30
PCA (m=5) + LDA - First threshold	2000	186	9.30
PCA (m=5) + LDA - Second threshold	2000	185	9.25
PCA (m=6) + LDA - First threshold	2000	186	9.30
PCA (m=6) + LDA - Second threshold	2000	184	9.20

Table 1: Table showing the results of the LDA and PCA + LDA method for classification.

3 Multivariate Gaussian Density

Multivariate Gaussian Density is an extension of the Gaussian Density to multiple dimensions. It is used to describe the distribution of a vector of random variables in a $multi-dimensional$ space, and it could be defined as:

$$N(x \mid \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{M}{2}} \, |\Sigma|^{\frac{1}{2}}} \exp^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)} \tag{6}$$

where M is the size of the feature vector x, and $|\Sigma|$ is the determinant of Σ. Since the computation of the exponential could cause problems, the logarithm is applied, so from Equation 6 we get Equation 7:

$$\log N(x \mid \mu, \Sigma) = -\frac{M}{2}\log(2\pi) - \frac{1}{2}\log|\Sigma| - \frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu) \tag{7}$$

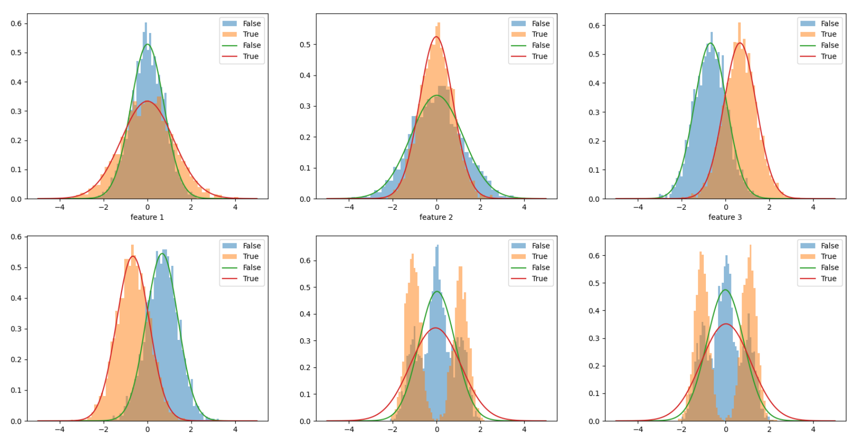


Figure 6: Gaussian Density

Thus, applying Gaussian probability density to the 6 features in our dataset, the following results in Figure 6, are obteinded.

It can observe that **features 1 - 2 - 3 - 4** fit the Gaussian distribution, the histogram of the data is well approximated by the Gaussian curve. For **features 5 - 6** this does not happen and therefore could not be a good model.

4 Model evaluation for classification

To assess whether one model classifies correctly or it is better than another, it is important to introduce what may be a unit of measurement. In particular, the evaluation of a model is done using the **DCF (Detection Cost Function)** or also called **empirical Bayes Risk**. But before delving into this method of valuation, we must consider what is called the working point of a binary classification application. This point is characterised by a triplet of values:

$$(\pi_T, C_{fn}, C_{fp}) \tag{8}$$

where π_T is the prior probability of the target class, C_{fn} is the cost of a false negative and C_{fp} is the cost of a false positive.

But from π_T, one can have attention on a particular triplet that is defined by:

$$(\tilde{\pi}, C_{fn}, C_{fp}) = (\tilde{\pi}, 1, 1) \tag{9}$$

where $\tilde{\pi}$ in Equation 9 is called **effective prior** probability of the target class, defined as:

$$\tilde{\pi} = \frac{\pi_T C_{fn}}{\pi_T C_{fn} + (1 - \pi_T) C_{fp}} \tag{10}$$

So we can define DCF_u as:

$$B_{emp} = DCF_u = \sum_{c=1}^{K} \frac{\pi_c}{N_c} \sum_{i|c_i=c} C(a(x_i, R) \mid c) \tag{11}$$

From Equation 11, we can obtain:

$$DCF_u(\pi_T, C_{fn}, C_{fp}) = \pi_T C_{fn} P_{fn} + (1 - \pi_T) C_{fp} P_{fp} \tag{12}$$

where:

$$P_{fn} = \frac{FN}{FN + TP} \ , \quad P_{fp} = \frac{FP}{FP + TN} \tag{13}$$

From the Equation 11, it is possible to find **minDCF** and **actDCF**. When calculating minDCF, the threshold used is the one that minimize DCF, and there are various methods for finding it. Whereas for actDCF, the threshold used is:

$$t' = -\log \frac{\tilde{\pi}}{1 - \tilde{\pi}} \tag{14}$$

5 Classification Models Analysis

To perform the classification, the dataset must first be divided into two sub-portions, the training and validation sub-portions.

5.1 Gaussian models

Since, it is dealing with a binary classification task, it will assign a probabilistic score to each sample in terms of the class-posterior log-ratio:

$$\log r(x_t) = \log \frac{P(C = h_1 \mid x_t)}{P(C = h_0 \mid x_t)} \tag{15}$$

Analysing Equation 15 in more detail, it becomes:

$$\log r(x_t) = \log \frac{f_{X|C}(x_t \mid h_1)}{f_{X|C}(x_t \mid h_0)} + \log \frac{P(C = h_1)}{P(C = h_0)} \tag{16}$$

The first addend of the equation is called the *llr* or *log-likelihood ratio* and an optimal decision is given by Equation 17.

$$\log r(x_t) \gtrless 0 \tag{17}$$

Considering $P(C = h_1) = \pi$ and $P(C = h_0) = 1 - \pi$, from Equation 16 and Equation 17, it is possible to write that the class assignment is based on Equation 16 and Equation 17, to obtain Equation 18.

$$llr(x_t) = \log \frac{f_{X|C}(x_t \mid h_1)}{f_{X|C}(x_t \mid h_0)} \gtrless -\log \frac{\pi}{1 - \pi} \tag{18}$$

The optimal class decision is based on a comparison between the and a threshold , if the llr is greater than th the sample is assigned to class h_1, otherwise to class h_0. It is necessary to find the parameters θ, μ_c, Σ_c; this can be done by maximising the log-likelihood. Parameter estimation is part of the training phase and this therefore performed on the training part of the dataset, then an estimation of the error rate can be performed on the validation part.

5.1.1 Multivariate Gaussian Classifier

The first classifier is MVG and it is given by the empirical mean and covariance matrix for each class,

$$\mu_c^* = \frac{1}{N_c} \sum_{i|c_i=c} x_i \ , \quad \Sigma_c^* = \frac{1}{N_c} \sum_{i|c_i=c} (x_i - \mu_c^*)(x_i - \mu_c^*)^T \tag{19}$$

5.1.2 Naive Bayes Gaussian Classifier

This model makes an important assumption that simplifies the number of parameters to be estimated, it assumes that the features are independent given their class. This causes the covariance matrix to be a diagonal matrix, consequently, matching MVG with a diagonal covariance matrix. However, the assumption of independence may be too restrictive and lead to inferior performance if the features are indeed correlated.

$$\mu_{c,[j]}^* = \frac{1}{N_c} \sum_{i|c_i=c} x_{i,[j]} \ , \quad \sigma_{c,[j]}^2 = \frac{1}{N_c} \sum_{i|c_i=c} (x_{i,[j]} - \mu_{c,[j]}^*)^2 \tag{20}$$

5.1.3 Tied Covariance Gaussian Classifier

The assumption of the latter model consists of its own average for each class, but an equal covariance matrix for all classes.

$$\mu_c^* = \frac{1}{N_c} \sum_{i|c_i=c} x_i \ , \quad \Sigma^* = \frac{1}{N} \sum_{c} \sum_{i|c_i=c} (x_i - \mu_c)(x_i - \mu_c)^T \tag{21}$$

The characteristic of this model is that it is strongly correlated to LDA.

5.1.4 Gaussian Models Comparison

A threshold of 0 was used to perform our results, which means that $P(C = 1) = P(C = 0) = 1/2$. This model was applied and the outcomes can be seen in the Table 2.

Features	**Model**	**Error Rate** (%)
	no PCA	
1 to 6	MVG	7.00
1 to 6	Naive Bayes	7.20
1 to 6	Tied Covariance	9.30
1 to 4	MVG	7.95
1 to 4	Naive Bayes	7.65
1 to 4	Tied Covariance	9.50
1 - 2	MVG	36.50
1 - 2	Naive Bayes	36.30
1 - 2	Tied Covariance	49.45
3 - 4	MVG	9.45
3 - 4	Naive Bayes	9.45
3 - 4	Tied Covariance	9.40
	PCA m = 5	
1 to 6	MVG	7.10
1 to 6	Naive Bayes	8.75
1 to 6	Tied Covariance	9.30
	PCA m = 6	
1 to 6	MVG	7.00
1 to 6	Naive Bayes	8.90
1 to 6	Tied Covariance	9.30

Table 2: Table showing the results of the Error Rate for different Models and Features.

Comparing the results with the Table 1, we can see that for some configurations there were improvements in terms of error rate. This means that Gaussian models are better able to classify the data. If we go into the details of how the error rate changes as a function of the observed features, we can see that:

- **1 to 6**: in the case we consider all 6 features, the error rate is quite low and its range goes from 7.00% to 9.30%. This means that they all provide useful information.
- **1 to 4**: if we consider features from 1 to 4, we can see that the error rate increases

slightly. This allow us to say that features 5 and 6 have useful but not fundamental information to change the outcome.

- **1 - 2**: features 1 and 2 have a rather high error rate, meaning that they don't contain relevant information.

- **3 - 4**: on the other hand, the latter two features considered have a rather low error rate, a value close to the case where all features are considered. This means that the information contained in these two features is relevant for making the classification.

Starting with the previous performance, it may be interesting to analyse how performance changes as three main parameters vary:

- $\tilde{\pi}$: represents prior probability of the positive class

- C_{fn}: misclassification cost of a sample predicted as negative but it is positive

- C_{fp}: misclassification cost of a sample predicted as positive but it is negative

	MVG	**Naive Bayes**	**Tied Covariance**
	Application $(\tilde{\pi}, C_{fn}, C_{fp}) = (0.5, 1, 1)$		
actDCF	0.1399	0.1439	0.1860
minDCF	0.1302	0.1311	0.1812
	Application $(\tilde{\pi}, C_{fn}, C_{fp}) = (0.9, 1, 1)$		
actDCF	0.4001	0.3893	0.4626
minDCF	0.3423	0.3509	0.4421
	Application $(\tilde{\pi}, C_{fn}, C_{fp}) = (0.1, 1, 1)$		
actDCF	0.3051	0.3022	0.4061
minDCF	0.2629	0.2569	0.3628
	Application $(\tilde{\pi}, C_{fn}, C_{fp}) = (0.5, 1, 9)$		
actDCF	0.3051	0.3022	0.4061
minDCF	0.2629	0.2569	0.3628
	Application $(\tilde{\pi}, C_{fn}, C_{fp}) = (0.5, 9, 1)$		
actDCF	0.4001	0.3893	0.4626
minDCF	0.3423	0.3509	0.4421

Table 3: Table showing minDCF and actDCF for different models and applications.

Analysing the results of the Table 3, it is possible to observe:

- Observing how the $\tilde{\pi}$ varies, it can be seen that the best outcome is obtained when it takes the value 0.5. On the other hand, when it takes value 0.1 and 0.9, the outcome gets worse because it penalises false negatives and false positives respectively

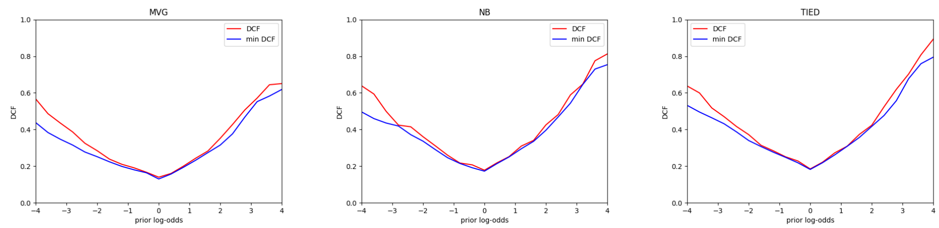
- Observing how the values of C_{fn} and C_{fp} change when they assume value 9. The outcomes worsen, in particular there is a greater impact on the model when the cost of false negatives increases.

Starting from the result obtained in Table 3, it is possible to consider the application of PCA as pre-processing technique focusing on the cases of $\tilde{\pi}$ equal to 0.1, 0.5 and 0.9 and $C_{fn} = C_{fp} = 1$, obtaining the outcomes shown in Table 4

	MVG	Naive Bayes	Tied Covariance
Application $(\tilde{\pi}, C_{fn}, C_{fp}) = (0.5, 1, 1)$			
no PCA			
actDCF	0.1399	0.1439	0.1860
minDCF	0.1302	0.1311	0.1812
PCA $m = 5$			
actDCF	0.1419	0.1749	0.1860
minDCF	0.1331	0.1737	0.1812
$m = 6$			
actDCF	0.1399	0.1780	0.1860
minDCF	0.1302	0.1727	0.1812
Application $(\tilde{\pi}, C_{fn}, C_{fp}) = (0.9, 1, 1)$			
no PCA			
actDCF	0.4001	0.3893	0.4626
minDCF	0.3423	0.3509	0.4421
PCA $m = 5$			
actDCF	0.3980	0.4660	0.4626
minDCF	0.3512	0.4340	0.4451
$m = 6$			
actDCF	0.4001	0.4512	0.4626
minDCF	0.3423	0.4359	0.4421
Application $(\tilde{\pi}, C_{fn}, C_{fp}) = (0.1, 1, 1)$			
no PCA			
actDCF	0.3051	0.3022	0.4061
minDCF	0.2629	0.2569	0.3628
PCA $m = 5$			
actDCF	0.3042	0.3930	0.4051
minDCF	0.2738	0.3545	0.3648
$m = 6$			
actDCF	0.3051	0.3920	0.4061
minDCF	0.2629	0.3535	0.3628

Table 4: Show minDCF and actDCF for different models and applications before and after applying PCA.

From the results obtained in the Table 4, it can be seen that the application of PCA was not very helpful because in no case did the outcomes improve, instead they remained



(a) MVG (b) Naive Bayes (c) Tied Covariance

Figure 7: Error Bayes plots

the same or even worsened. Analysing overall, it can be said that the model that tends to perform worse is the Tied Covariance, whereas MVG and Naive Bayes are rather similar. In particular for MVG and Naive Bayes, the best result is obtained for the $0.5, 1, 1$ configuration whether applying PCA or not. In addition, it can be said that there is a good calibration for this configuration because applying or not applying PCA, minDCF and actDCF doesn't change what is not the case for the other configurations.

In Figure 7 the Bayes error was calculated for a prior log odds in the range $(-4, +4)$, for the three models with a configuration having a $\tilde{\pi} = 0.1$ and applying the PCA as pre-processing.

5.2 Logistic Regression Classifier

Logistic Regression is a discriminative classification model, directly evaluating the posterior probability $C \mid X$. In particular by determining that hyperplane which maximises the posterior probability. Starting from the results obtained from the Tied Gaussian that provides log-likelihood ratios that are linear functions of our data, where log-posterior probability ratio is:

$$\log \frac{P(C = h_1 \mid X)}{P(C = h_0 \mid X)} = \log \frac{f_{X|C}(x \mid h_1)}{f_{X|C}(x \mid h_0)} + \log \frac{\pi}{1 - \pi} = \omega^T x + b \tag{22}$$

where prior information has been absorbed in the bias term b of the Equation 22. So from this point we can define the score function as:

$$s(x) = \omega^T x + b = 0 \tag{23}$$

where it is positive for samples of class h_1 and negative for samples of class h_0. Given ω and b we can compute the posterior class probability as:

$$P(C = h_1 \mid x, \omega, b) = \sigma(\omega^T x + b) = \sigma(s(x)) \tag{24}$$

where σ is sigmoid function defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{25}$$

This approach assumes that the decision rules will be hyperplanes orthogonal to vector w.

5.2.1 Binary Logistic Regression

Binary Logistic Regression Not Prior-Weighted

The objective is to minimise the loss function $J(\omega, b)$, but to this is introduced what is a penalty term, so the new function becomes:

$$J(\omega, b) = \frac{\lambda}{2} \|w\|^2 + \frac{1}{n} \sum_{i=1}^{n} \log(1 + e^{-z_i(\omega^T x_i + b)}), \quad z_i = \begin{cases} 1 & \text{if } c_i = 1 \\ -1 & \text{if } c_i = 0 \end{cases} \tag{26}$$

where λ of Equation 26 is the regularization term, this term has been introduced to make problem solvable in case of linearly separable classes.

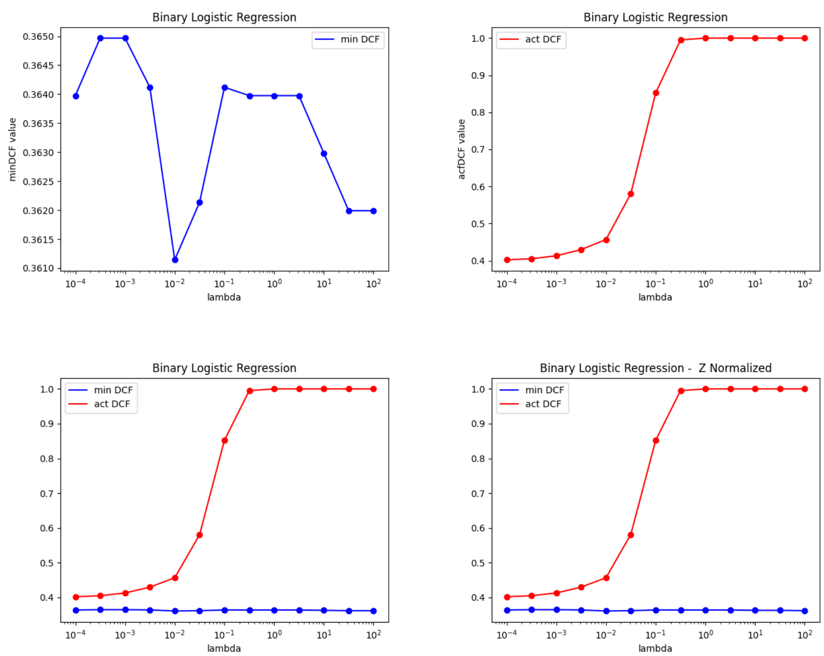


Figure 8: Binary Logistic Regression not Prior-Weighted

In this model $\pi_T = 0.1$ is used. In Table 5, it can be seen how the values of minDCF and actDCF vary when λ changes, Z-normalization is applied or not and if the whole training set or a portion was used. It can be deduced from the values obtained that the application of z-normalisation brings no advantage. On the other hand, by using only 50 samples, it can see that using a limited number of samples can significantly influence the model and could lead to misleading results that are not representative of the entire sample. Consequently, as many samples as possible should be used for training to obtain a more accurate model. Figure 8 and Figure 9 give a graphic representation of how minDCF and actDCF vary with λ

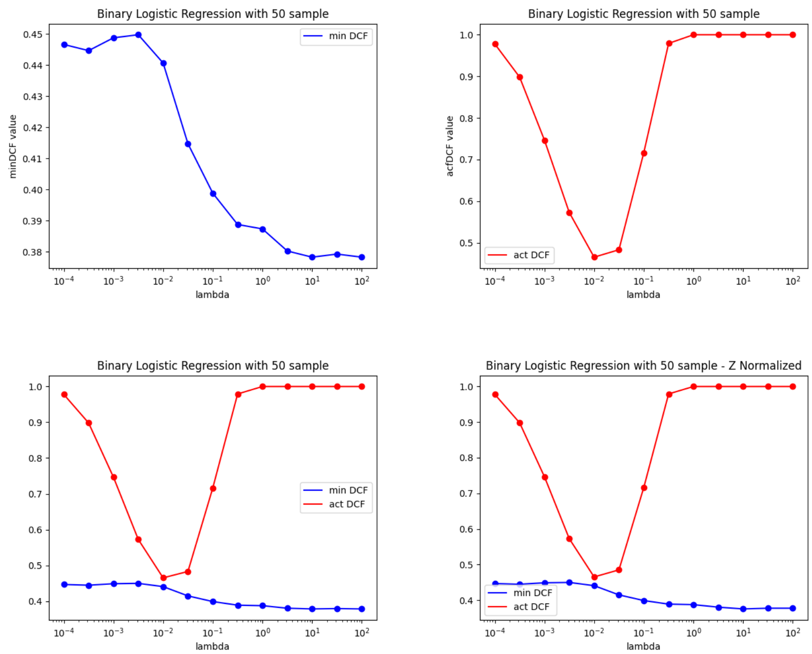


Figure 9: Binary Logistic Regression not Prior-Weighted with 50 Samples

Binary Logistic Regression Not Prior-Weighted				
λ	**minDCF**		**actDCF**	
	no z-norm	**z-norm**	**no z-norm**	**z-norm**
10^{-4}	0.3640	0.3640	0.4021	0.4021
10^{-3}	0.3650	0.3650	0.4130	0.4130
10^{-2}	0.3611	0.3611	0.4568	0.4568
10^{-1}	0.3641	0.3641	0.8522	0.8522

Binary Logistic Regression Not Prior-Weighted (50 Samples)				
λ	**minDCF**		**actDCF**	
	no z-norm	**z-norm**	**no z-norm**	**z-norm**
10^{-4}	0.4466	0.4466	0.9780	0.9780
10^{-3}	0.4487	0.4487	0.7466	0.7466
10^{-2}	0.4407	0.4407	0.4652	0.4652
10^{-1}	0.3988	0.3988	0.7164	0.7164

Table 5: Show minDCF and actDCF for Binary Logistic Regression Not Prior-Weighted model

Binary Logistic Regression Prior-Weighted

Another possible Logistic Regression approach is that Prior-Weighted; it allows to simulate different priors for class 1. Therefore, the objective function becomes:

$$J(\omega, b) = \frac{\lambda}{2}\|w\|^2 + \sum_{i=1}^{n} \xi_i \log(1 + e^{-z_i(\omega^T x_i + b)}), \quad \xi_i = \begin{cases} \frac{\pi_t}{n_T} & \text{if } z_i = +1 (c_i = 1) \\ \frac{1-\pi_T}{n_F} & \text{if } z_i = -1 (c_i = 0) \end{cases} \tag{27}$$

Now it is possible analyze the results obtained from the Prior-Weighted model with $\pi_T = 0.1$.

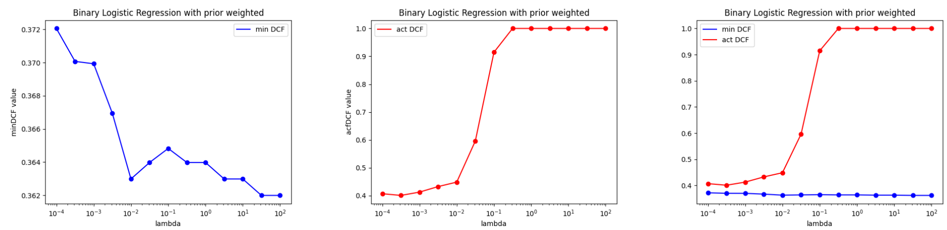


Figure 10: Binary Logistic Regression Prior-Weighted

Binary Logistic Regression Prior-Weighted		
$\pi_T = 0.1$		
λ	**minDCF**	**actDCF**
10^{-4}	0.3721	0.4071
10^{-3}	0.3699	0.4129
10^{-2}	0.3630	0.4487
10^{-1}	0.3648	0.9147

Table 6: Show minDCF and actDCF for Binary Logistic Regression Prior-Weighted

The role of the prior is to weight the samples during model training. In particular, samples in the higher priority class receive a higher weight than those in the lower priority class. This can be useful if the dataset is unbalanced, the choice of the prior must be made at the beginning and this choice can affect the model a lot, in fact we may even have a worsening of the model.

Comparing the results obtained in Table 5 and Table 6, there isn't noticeable change in the outcomes on minDCF and actDCF this means that our dataset is not unbalanced. The interesting value to observe is the value of actDCF when $\lambda = 10^-1$.

Binary Logistic Regression with Pre-processing (PCA)

In this case, PCA can be applied as pre-processing, and by looking at the values in Table 7, it can be seen that it does not result much improvement of the system.

5.2.2 Quadratic Logistic Regression

In this step we can analyze training on a Quadratic Logistic Regression model by performing features expansion, so it's possible write log-likelihood ratio as:

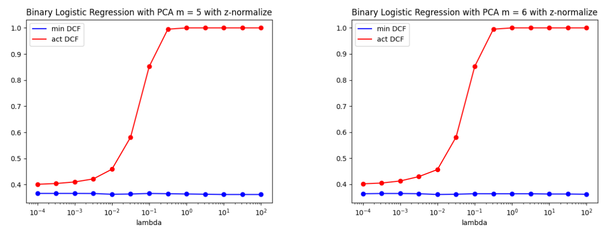


Figure 11: Binary Logistic Regression applying PCA and z-normalization

Binary Logistic Regression with PCA				
λ	**minDCF**		**actDCF**	
	no z-norm	**z-norm**	**no z-norm**	**z-norm**
		$m = 5$		
10^{-4}	0.3661	0.3661	0.4011	0.4011
10^{-3}	0.3661	0.3661	0.4100	0.4100
10^{-2}	0.3618	0.3628	0.4578	0.4588
10^{-1}	0.3660	0.3660	0.8502	0.8522
		$m = 6$		
10^{-4}	0.3640	0.3640	0.4021	0.4021
10^{-3}	0.3650	0.3650	0.4130	0.4130
10^{-2}	0.3611	0.3611	0.4568	0.4568
10^{-1}	0.3641	0.3641	0.8522	0.8522

Table 7: Show minDCF and actDCF for Binary Logistic Regression applying PCA and with adn withoud z-normalization

$$\log \frac{P(C = h_1|x)}{P(C = h_0|x)} = x^T A x + b^T x + c = s(\mathbf{x}, \mathbf{A}, \mathbf{b}, \mathbf{c}) \tag{28}$$

The Equation 28 is quadratic in x but it's linear in A and b. It can be rewritten to obtain a decision function that is linear for the expanded features space but quadratic in original features space.
So we can write features expansion as:

$$\Phi(x) = \begin{bmatrix} vec(xx^T) \\ x \end{bmatrix}, \quad w = \begin{bmatrix} vec(A) \\ b \end{bmatrix} \tag{29}$$

where $vec(X)$ in Equation 29 is the operator that stacks the columns of X into a single column vector. In this way we can write the posterior log-likelihood as:

$$s(x, w, c) = s^T \phi(x) + c \tag{30}$$

Looking the value in Table 8, we can see that this method gives better results than the methods seen before. This is possible because the quadratic method allows us to extract characteristics that are not possible by a linear method.

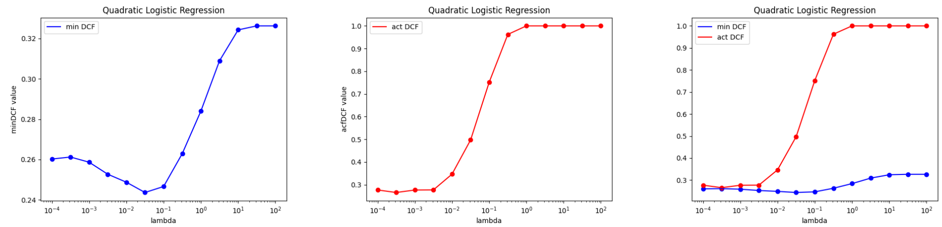


Figure 12: Quadratic Logistic Regression minDCF and actDCF

Quadratic Logistic Regression		
λ	**minDCF**	**actDCF**
10^{-4}	0.2602	0.2768
10^{-3}	0.2587	0.2765
10^{-2}	0.2487	0.3464
10^{-1}	0.2466	0.7520
	Best Result	
3.162^{-2}	0.2436	0.4972

Table 8: Show minDCF and actDCF for Quadratic Logistic Regression and best result in logistic regression

Summarize

From the results obtained, one can observe:

- The application of z-normalization does not bring any improvements, which means that being a linear transformation didn' t change the ability of logistic regression to separate classes, if labels and characteristics are linearly separable.
- Looking at the graphs depicted in Figure 8, Figure 9, Figure 10, Figure 11 and Figure 12, it can be seen that λ significantly affects the performance of the model in term of minDCF and actDCF. In particular, it can be observed that for values above 10^{-1} there is significant degradation, which is why the tables have shown values up to this λ value. This is because, remember that a larger value of λ can lead to overgeneralise resulting, so underfitting. Whereas too small λ can lead to low generalisation and thus to overfitting.
- It can be seen that there is a difference between minDCF and actDCF, which means that the models don't have a good calibration. In our case, Quadratic Logistic Regression seems to give the best results, so we can deduce that the model must be able to capture non-linear relationships between the variables and thus find non-linear separation rules in the feature space.

5.3 Support Vector Machine Classifier

5.3.1 Linear Support Vector Machines

Support Vector Machines are linear classifiers that look for maximum margin separation hyperplanes. The primal formulation of the soft-margin SVM problem consists in minimizing the function:

$$\mathbf{J}(w,b) = \frac{1}{2}||w||^2 + C\sum_{i=1}^{N} max(0, 1 - z_i(w^T x_i + b)) \tag{31}$$

where N is the number of training samples, C is the regularization parameter, and z_i is the margin of the i-th sample.
The dual formulation of the problem is:

$$\mathbf{J}(\alpha) = -\frac{1}{2}\alpha^T\ \mathbf{H}\ \alpha + \alpha^T\mathbf{1} \qquad 1 \leq \alpha_i \leq C,\ \ \forall i \in \{1, ..., N\},\ \ \sum_{i=1}^{n}\alpha_i z_i = 0 \tag{32}$$

where H is $H_{ij} = z_i z_j x_i^T x_j$ and the dual solution is the maximizer of $J^D(\alpha)$.
Primal and dual solutions are releted through:

$$w^* = \sum_{i=1}^{N}\alpha_i^* z_i x_i \tag{33}$$

In addition it's possible to rewrite dual problem as minimization of:

$$\hat{\mathbf{L}}(\alpha) = -\mathbf{J}(\alpha) = \frac{1}{2}\alpha^T\ \mathbf{H}\ \alpha - \alpha^T\mathbf{1} \tag{34}$$

and it can be minimize by L-BFGS-B algorithm. After that we have calculated the optimal α we can compute w^*.