Machine learning and pattern recognition project

Massimo Francios - s328914

17 July 2024

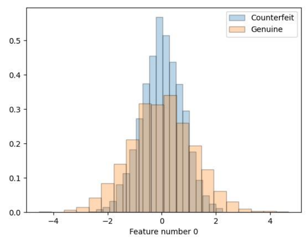
1 Introduction and project goals

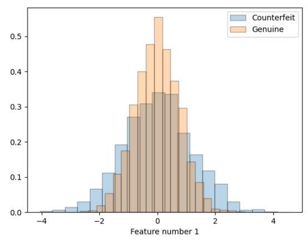
The main objective of the project involves creating a classifier capable of distinguishing between an Genuine fingerprint or a Counterfeit fingerprint. Our goal is to perform classification using different techniques, analyze outcomes and justify our chosen methodologies. We will examine Gaussian classifiers, Logistic Regression, Support Vector Machine and Gaussian Mixture Models. After that, the last part will use an evaluation set to see the performance of the choices we have made.

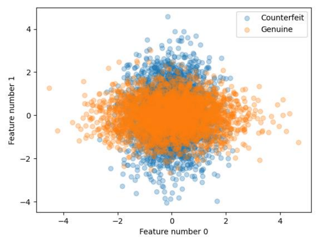
2 Dataset features

The first thing we want to do is to perform an initial analysis on how data looks like. The dataset samples are 6-dimensional, so we need to analyze 6 different features. In order to do so, we consider two features at a time and we visualize histograms and scatter plots.

2.1 Feature 0 and 1

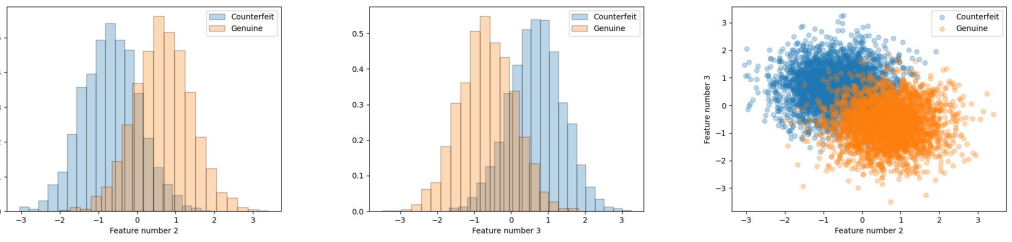






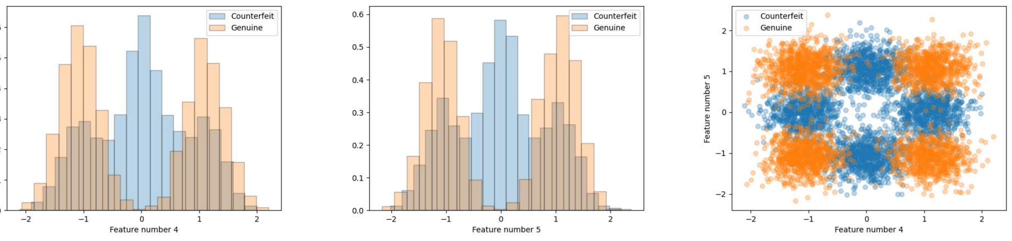
We can see from histograms that, in both features, classes overlap mainly in range [-2,2], they have similar mean but different variance: in particular for feature 0 we notice a higher variance for Genuine class, and a lower one for Counterfeit class. For feature 1 the behavior is the opposite. We can observe one single mode.

2.2 Feature 2 and 3



For this pair, we can observe two different modes. Classes overlap in range [-1,1]: Genuine class shows a higher mean for feature 2 and a lower mean for feature 3. Vice-versa for the Counterfeit class. Variances are very similar.

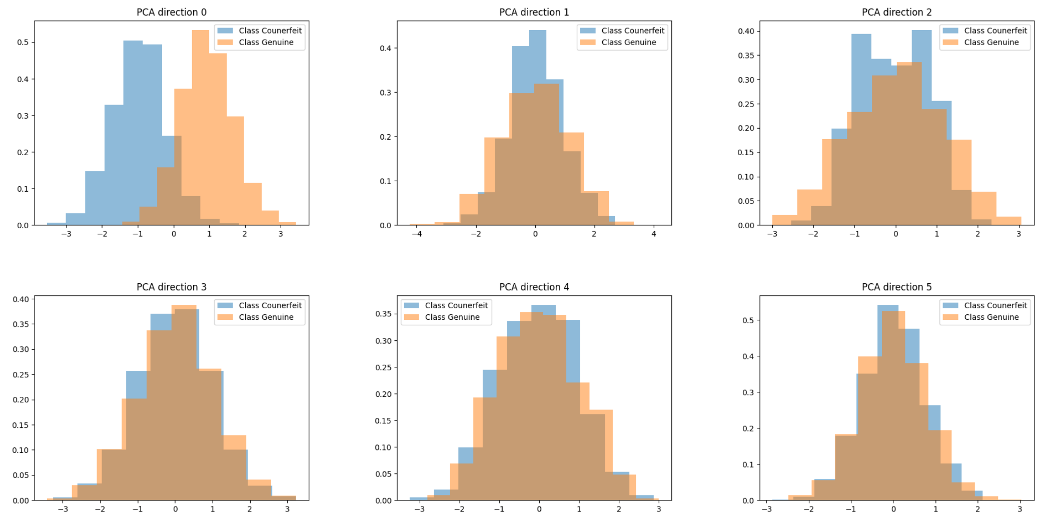
2.3 Feature 4 and 5



We can observe three modes for the last two features (4 and 5). Classes overlap in intervals [-2, -0.5] and [0.5, 2]. From the scatter plot we can identify 4 distinct clusters for each class.

3 PCA

We'll start analyzing how PCA works with our dataset. In particular we plot the histograms of the six features after applying PCA, starting with the direction with highest variance. We can see that the PCA direction with highest variance allows us to distinguish two clusters, while the other components show significant class overlap.



4 LDA

In this section we will examine the effects of LDA on our dataset. We are looking for the LDA discriminat direction that maximizes classes separations. The histogram shows that LDA is finding a direction that separates classes and shows little class overlap. Compared to the histograms of the 6 features, LDA is able to provide better separation

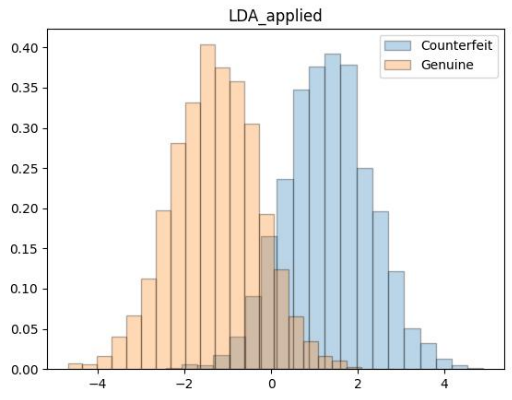


Figure 1: LDA applied on dataset

We can try to apply LDA as a classifier. In order to do that, we need to project samples in the

LDA discrimant direction, then we compute a threshold to make prediction:

$$th = \frac{\mu_0 + \mu_1}{2}$$

Then we optimize this threshold in order to get the best possible error rate for our validation set. At the end, we also try pre-process data using PCA. The following table shows the error rates of LDA as a classifier:

No PCA	PCA $m = 5$	PCA $m = 4$	PCA $m = 3$	PCA $m = 2$
9.3%	9.3%	9.25%	9.25%	8.95%

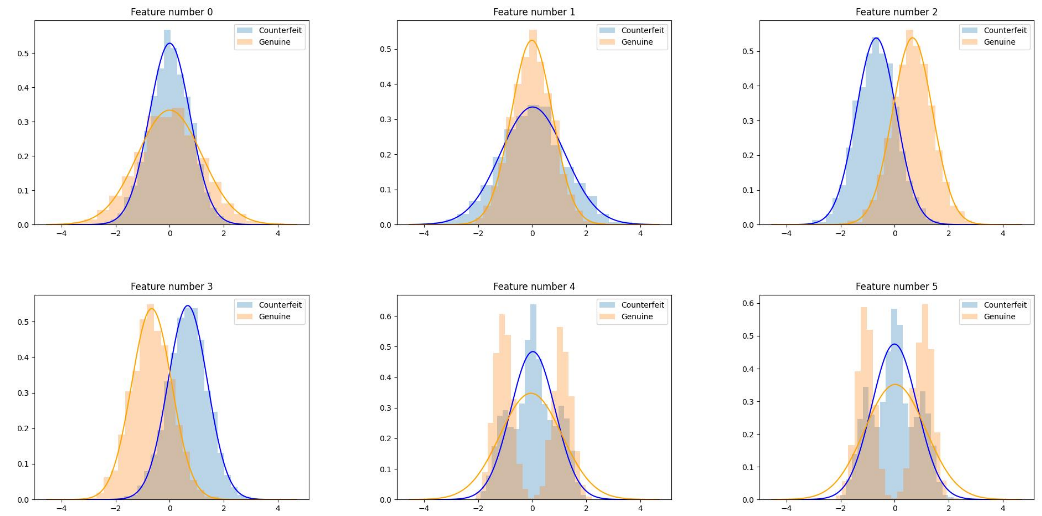
Table 5: Error rates for LDA applied as a classifier

We can see from the result that PCA slightly helps reducing error rate, with a best hyperparameter $m = 2$ and 8.95% of error rate. We can say that for our data, PCA is not very helpul when combined with LDA.

5 Gaussian models

5.1 Density estimation

In this section we want to analyze how generative Gaussian models performs on our dataset. Before examine all the possible models we try to model the distribution of different features using uni-variate Gaussian. We can estimate the parameters of a Gaussian distribution for each feature using ML estimate. Then we plot the estimated density on top of the normalized histograms of data.



We can see from the histograms above that our estimations provide a good fit for feature 0,1,2,3 for the genuine class; for the last two features our estimates are very inaccurate. For class Counterfeit the behavior is similar, but for the last two features we have a slightly better fit.

5.2 Models trained on all features

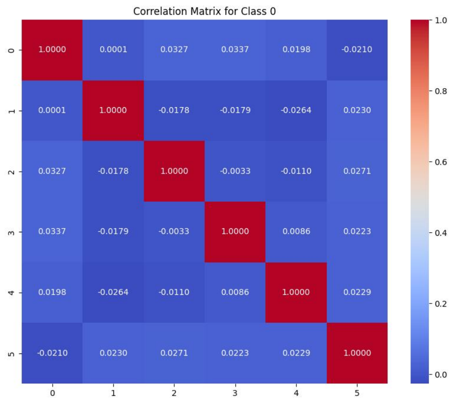
After verifying MVG assumption, we proceed to train data on all features. We start training the standard MVG classifier, then the Naive Bayes, and also the Tied MVG model. We compare all models in terms of error rates, including LDA classifier. Results are shown in the following table:

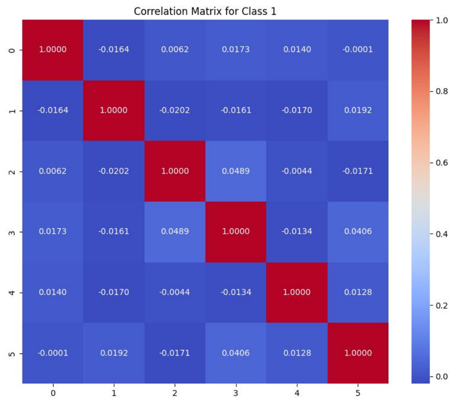
Standard MVG	Naive Bayes	Tied Covariance	LDA
7.0%	7.2%	9.3%	9.3

Table 7: Error rates for MVG model comparison with LDA

The best performing is the standard MVG model, followed by the Naive and lastly by LDA and Tied Covariance. This last two model are very correlated: the Tied covariance suggests that the optimal threshold for classifying samples depends on the prior.

Now we try to give an explanation for the Naive Bayes performing relatively well: we analyze the Pearson correlation between features. We know that Naive Bayes assumes correlations between features equal to zero. The following heatmaps explain why Naive Bayes has similar performance to the unconstrained model: we can clearly see that features are weekly correlated.





In both classes, correlation between features is very close to zero for all pairs.

5.3 Models trained on features 0 to 4

After visualizing the results of Gaussian distribution fitting, we noticed that the last two features didn't provide a good fit. We try to use MVG models excluding these two features to see how they affect performance. From the following table we can see that their absence makes error rates higher, suggesting that even if their fit were not good, they can provide useful information.

Standard MVG	Naive Bayes	Tied covariance
7.95%	7.65%	9.5%

Table 9: Error rates of MVG models excluding last two features

5.4 Models trained on first and second pair of features

From our initial analysis we remember that the first two features (0,1) show similar mean but different variances. while for the second pair (2,3) the behavior is the opposite. We train MVG model (Standard and Tied only) to see how these pairs perform.

Feature pair	Standard MVG	Tied covariance
First two features	36.5%	49.5%
Third and fourth features	9.5%	9.4%

Table 10: Error rates for MVG (pair features)

First two features seems to be poorly effective for MVG and Tied models: they have different variances so the tied model assumption is not satisfied. Also they have similar means and this does not help standard MVG. Third and fourth features instead provide good results (they are similar in terms of error rate) and this can be explained because they share similar variances (tied assumption is satisfied) and also different means help standard MVG model.

5.4.1 PCA pre-processing for MVG

Finally we analyze the effect of PCA for MVG models. We try all possible PCA values and visualize the results in the following table:

PCA value	Standard MVG	Naive Bayes	Tied covariance
No pca	7.0%	8.9%	9.3%
5	7.1%	8.75%	9.3%
4	8.05%	8.85%	9.25%
3	8.8%	9.0%	9.25%
2	8.8%	8.85%	9.25%

Table 11: MVG models error rate with PCA

We can see that PCA does not provide significant benefits to MVG models. We can say that the best performing model among all of them is the **Standard MVG model with no PCA**.

6 Performance of MVG models

In this section we evaluate the performance of the previous MVG models for different working points. We start considering these 5 configuration composed of π, C_{fn}, C_{fp}:

- (0.5, 1, 1)
- (0.9, 1, 1)
- (0.1, 1, 1)
- (0.5, 1, 9)
- (0.5, 9, 1)

If we analyze these working point in terms of effective prior we can notice that configuration 4 is the same as 3 (a higher C_fp cost makes the model to predict more false label, making the prior lower. Vice-versa, configuration 5 is the same as configuration 2 (a higher C_fn cost makes the model predict more true label). We can focus on just 3 working point in terms of effective prior: $\tilde{\pi}_1 = 0.1$, $\tilde{\pi}_2 = 0.5$ and $\tilde{\pi} = 0.9$.

6.1 Configuration (0.5, 1.0, 1.0)

PCA value	Standard MVG minDCF vs ActDCF	Naive Bayes minDCF vs ActDC	Tied Covariance minDCF vs ActDCF
no pca	0.1302; 0.1399	0.1727; 0.1780	0.1812; 0.1860
5	0.1331; 0.1419	0.1737; 0.1750	0.1812; 0.1860
4	0.1537; 0.1609	0.1717; 0.1770	0.1821; 0.1850
3	0.1734; 0.1759	0.1746; 0.1799	0.1830; 0.1850
2	0.1731; 0.1760	0.1710; 0.1770	0.1789; 0.1850
1	0.1769; 0.1850	0.1769; 0.1850	0.1769; 0.1870

Table 12: MinDCF and Actual DCF values for different MVG classifiers and PCA values

In terms of minimum DCF the best model is again standard MVG with no pre-processing, followed by naive and then tied model. Overall, scores seems to be well calibrated.

6.2 Configuration (0.1, 1.0, 1.0)

PCA value	Standard MVG minDCF vs ActDCF	Naive Bayes minDCF vs ActDC	Tied Covariance minDCF vs ActDCF
6	0.2629; 0.3051	0.3535; 0.3920	0.3628; 0.4061
5	0.2738; 0.3041	0.3545; 0.3930	0.3648; 0.4051
4	0.3012; 0.3529	0.3614; 0.3970	0.3610; 0.4031
3	0.3563; 0.3879	0.3645; 0.3950	0.3681; 0.4082
2	0.3526; 0.3879	0.3562; 0.3869	0.3630; 0.3960
1	0.3686; 0.3970	0.3686; 0.3970	0.3686; 0.4021

Table 13: MinDCF and Actual DCF values for different MVG classifiers and PCA values

For this application we see that the standard MVG in again the best model in terms of minimum DCF. Calibration seems to be slightly worse than the previous application.

6.3 Configuration (0.9, 1.0, 1.0)

PCA value	Standard MVG minDCF vs ActDCF	Naive Bayes minDCF vs ActDC	Tied Covariance minDCF vs ActDCF
6	0.3423; 0.4001	0.4359; 0.4512	0.4421; 0.4626
5	0.3512; 0.3980	0.4340; 0.4660	0.4451; 0.4626
4	0.4150; 0.4598	0.4313; 0.4630	0.4441; 0.4615
3	0.4392; 0.4680	0.4343; 0.4591	0.4342; 0.4565
2	0.4384; 0.4434	0.4323; 0.4424	0.4352; 0.4785
1	0.4342; 0.4775	0.4342; 0.4775	0.4342; 0.4806

Table 14: MinDCF and Actual DCF values for different MVG classifiers and PCA values

Again for this application the standard MVG performs better. Calibration is worse than the first application.

6.4 Bayes error plots

Bayes error plots can help visualizing performance for a wide range of applications. We visualize plots where we compare minimum DCF vs actual DCF of different MVG models for our target application $\pi_T = 0.1$.

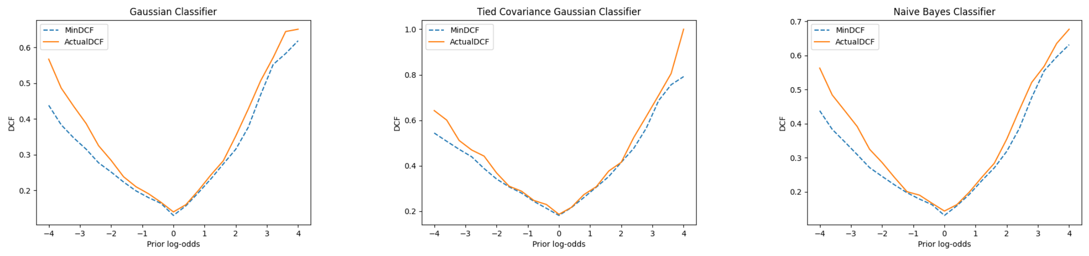


Table 15: MVG vs Tied covariance vs Naive Bayes

We can see that calibration is good for prior log odds near 0, while far from 0 calibration is worse. We can also visualize minimum DCF of all models:

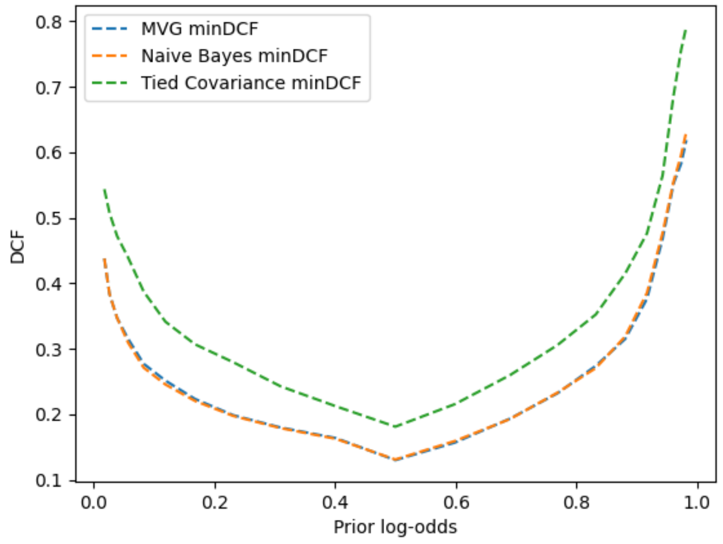


Figure 2: Minimum DCF for MVG, Naive Bayes and Tied MVG

We can clearly see that standard MVG and Naive Bayes perform better than Tied covariance all application points we examined.

7 Logistic regression

In this section, we analyze the binary logistic regression on our project data. From now we focus on an application that has a prior of $\pi_T = 0.1$. We also use cross-validation for testing different values for the regularization coefficient λ. We start analyzing data without any pre-processing

7.1 Standard logistic regression

Starting from the standard model, we use different values of λ from $(10^{-4}, 10^{2})$; then we plot actual DCF and minimum DCF for different values of λ

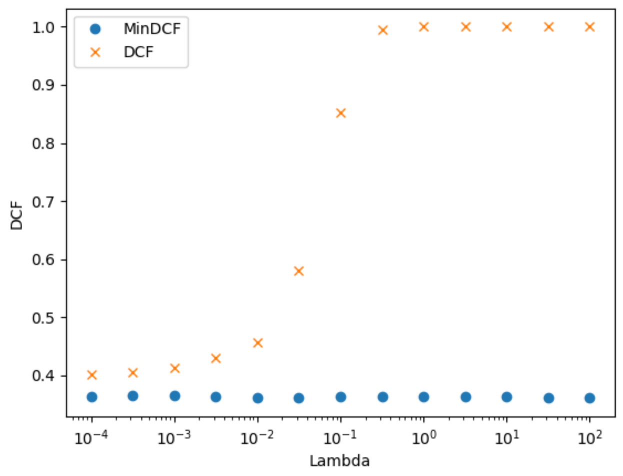


Figure 3: Standard Logistic regression: DCF vs λ

We can see that for greater value of the regularization coefficient, the calibration of the scores become lower and also the effectiveness for the application. In this case higher value of λ make scores lose their probabilistic interpretation.

7.2 Standard LR on a reduced dataset

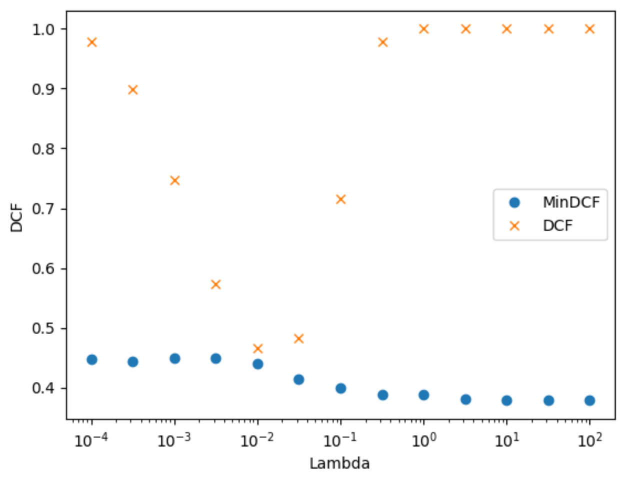


Figure 4: Standard LR on reduced dataset: DCF vs λ

Here, we trained the logistic regression with few samples: small values of λ imply risk of overfitting data. As the value increase we mitigate the problem. Then the behavior is similar to the standard model: increasing λ makes the scores lose their probabilistic interpretation.

7.3 Prior-weighted Logistic regression

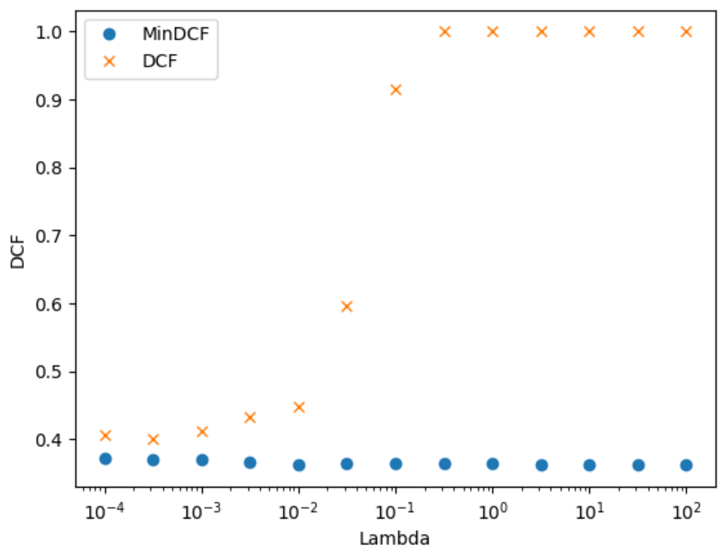


Figure 5: Prior-weighted LR: DCF vs λ

For this specific dataset, using a prior-weighted version of LR does not provide significal benefits for our application.

7.4 LR with centered data

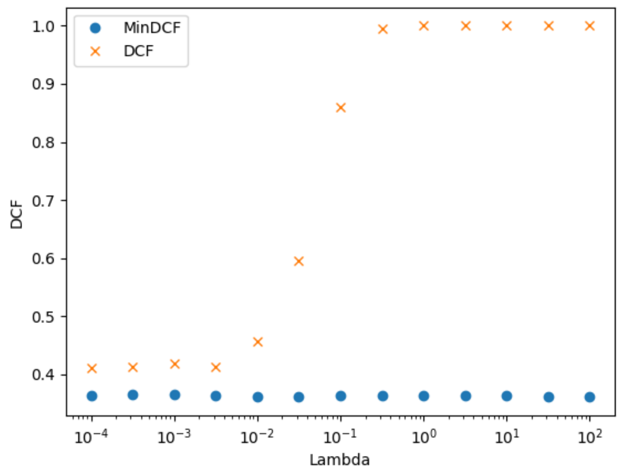
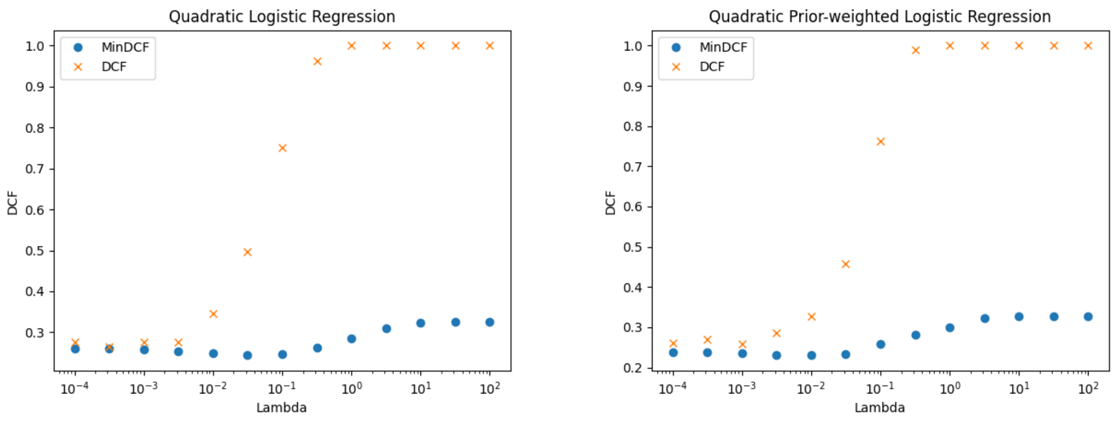


Figure 6: Centered data LR: DCF vs λ

Again, centering data seems to provide little to no benefits for our application.

7.5 Quadratic Logistic regression

In this section we focus on the Quadratic Logistic Regression model. In order to evaluate this model we apply a transformation to expand our samples. For this analysis we focus again on the entire dataset. We train a standard and a prior-weighted model, then we plot again actual and minimum dcf for different values of λ:



In both cases, we notice smaller values of actual DCF compared with linear LR: this means that a quadratic decision surface performs better than a linear one. Looking at values of λ we notice that regularization has become more effective for this expanded space, because it helps reduce values of minimum and actual DCF.

7.6 Model comparison up to now

The best model among all possible Logistic regression variation is the quadratic prior-weighted logistic regression model with the following parameters (in terms of minimum DCF):

$$\pi_T = 0.1 \text{ minDCF} = 0.23 \ \lambda = 3.16e^{-3}$$

For the gaussian classifiers the best model is standard MVG with no PCA:

$$\pi_T = 0.1 \text{ minDCF} = 0.2629$$

Both models have quadratic separation surface. In particula Naive Bayes classifier assumes independence between features which is an acceptable assumption for our data.

8 Support Vector Machine

For this new type of classifier we applied SVM on our entire dataset, focusing again on an application prior $\pi_T = 0.1$ and on different values of the regularization coefficient C. In particular we examine values of C in range $(10^{-5}, 1)$

8.1 Linear SVM

Starting with a standard model, we train our linear SVM with $K = 1$:

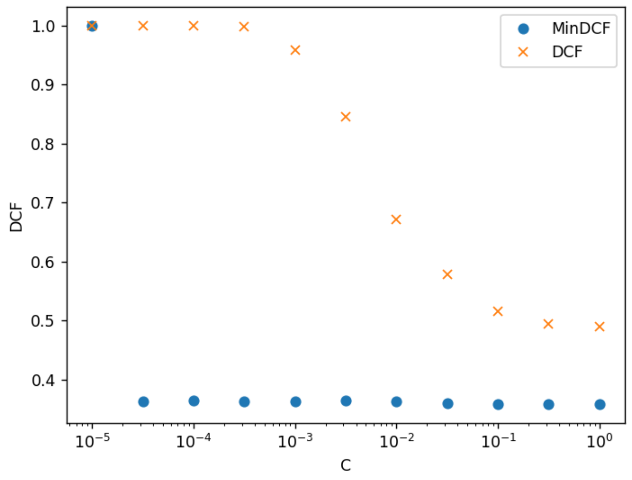


Figure 7: Linear SVM with no pre-processing: DCF vs C

We instantly notice that scores aren't well calibrated for our application. As the regularization coefficient C increases we notice that score calibration and effectiveness become better. Comparing minimum DCF with other linear model, this one performs slightly better. We notice that, except for the first value of C, minimum DCF is almost the same, while actual DCF.

8.2 Linear SVM with centered data

Similar to LR, we can try to center data:

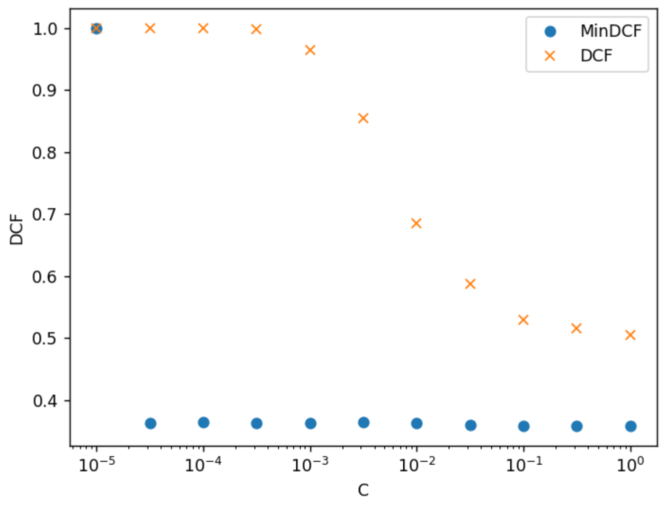


Figure 8: Linear SVM centered data: DCF vs C

Again, the centered data model seems to provide no benefits compared to the model trained on non pre-processed data.

8.3 Polynomial Kernel SVM

Now we consider polynomial kernel. For simplicity we consider only kernel with $d = 2$, $c = 1$ and we focus on original training data, with no pre-processing or centering. Looking at the plot, we see that

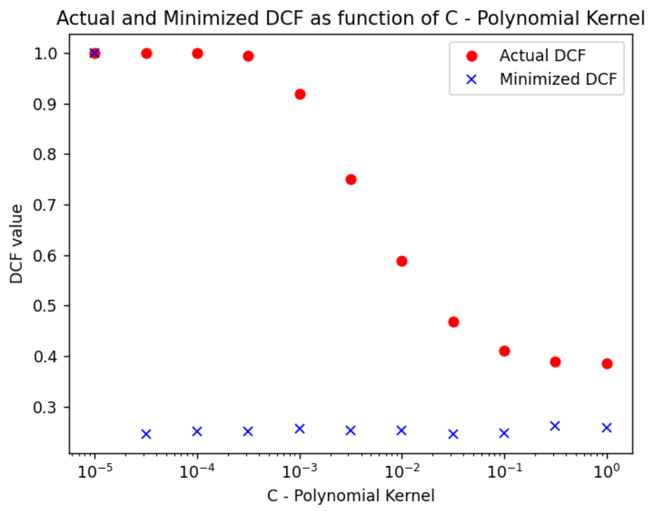


Figure 9: Polynomial Kernel SVM: DCF vs C

the behavior of the DCFs is the same as linear SVM. We notice that minimum and actual DCF values are lower compared with linear version of the model: this is consistent with the behavior we observed with Quadratic LR and Gaussian classifier (both have quadratic separation surface).

8.4 RBF kernel SVM

For this type of kernel we need to optimize both C and γ: we adopt a grid search approach, trying all possible combinations of the two values. γ values are selected from $[e^{-4}, e^{-3}, e^{-2}, e^{-1}]$.

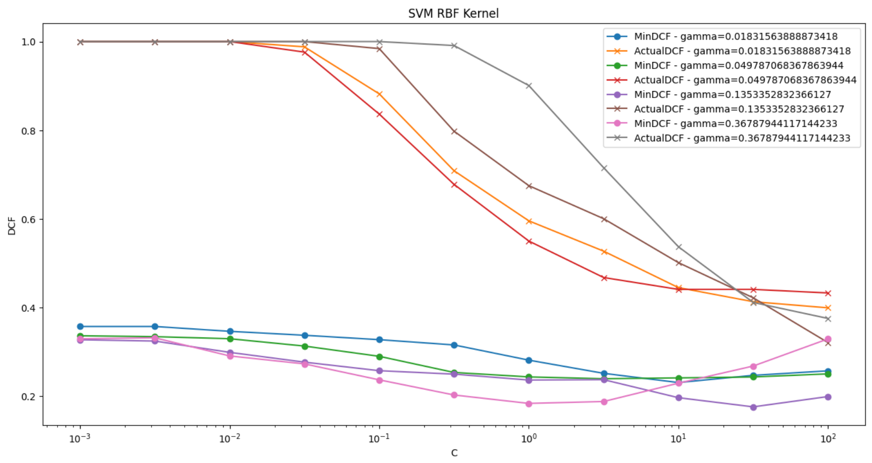


Figure 10: RBF kernel SVM: DCF vs C

In this case results are better when both γ and C have largest values. We can also see that scores are not well calibrated for this application. Up to now, among all SVM models this is the one that performs better with the following parameters:

$$\pi_T = 0.1 \quad \text{minDCF} = 0.1755 \quad C = 3.16e^{1} \quad \gamma = e^{-2}$$

9 Gaussian Mixture models

Our last approach is using Gaussian Mixture Models. As always we focus on application prior $\pi_T = 0.1$. We analyze both full and diagonal model. The following tables show minimum and actual DCF values for different combinations of GMM component numbers.

C0	C1	FULL minDCF	FULL ActualDCF	DIAG minDCF	DIAG ActualDCF
1	1	0.2629	0.3051	0.2570	0.3022
1	2	0.2649	0.3051	0.2605	0.3051
1	4	0.2139	0.2368	0.2098	0.2258
1	8	0.1850	0.1957	0.2045	0.2207
1	16	0.1495	0.2055	0.1433	0.1807
1	32	0.1847	0.2273	0.1373	0.1967
2	1	0.2181	0.2317	0.2449	0.2716
2	2	0.2160	0.2337	0.2489	0.2674
2	4	0.2234	0.2255	0.1990	0.2099
2	8	0.1860	0.2133	0.2039	0.2089
2	16	0.1701	0.1980	0.1536	0.1731
2	32	0.1857	0.2269	0.1436	0.1810
4	1	0.2330	0.2458	0.1451	0.1501
4	2	0.2317	0.2357	0.1542	0.1611
4	4	0.2161	0.2395	0.1481	0.1687
4	8	0.1887	0.2064	0.1403	0.1515
4	16	0.1745	0.1871	0.1372	0.1687
4	32	0.1867	0.2380	0.1412	0.1677
8	1	0.1759	0.2004	0.1731	0.1763
8	2	0.1808	0.1913	0.1762	0.1874
8	4	0.1959	0.1989	0.1585	0.1838
8	8	0.1786	0.1928	0.1463	0.1809
8	16	0.1526	0.1725	0.1324	0.1487
8	32	0.1755	0.1903	0.1312	0.1517
16	1	0.1670	0.1783	0.2069	0.2217
16	2	0.1657	0.1753	0.2039	0.2217
16	4	0.1918	0.1918	0.1963	0.2050
16	8	0.1755	0.2050	0.1979	0.2140
16	16	0.1631	0.1766	0.1622	0.1769
16	32	0.1752	0.1955	0.1644	0.1799
32	1	0.2570	0.2580	0.1748	0.1964
32	2	0.2560	0.2600	0.1748	0.1974
32	4	0.2460	0.2825	0.1929	0.1979
32	8	0.2191	0.2251	0.1929	0.1989
32	16	0.1938	0.2180	0.1850	0.2069
32	32	0.2337	0.2499	0.1766	0.1989

Table 17: Performance Metrics for FULL and DIAGONAL Models

The best combination for the Full model is: 1 components for class 0 and 16 components for class 1. For the diagonal model we have best performance with 8 components for class 0 and 32 components for class 1. However, a higher number of components doesn't necessarily mean better performance. We can also notice that diagonal GMM is best model up to now, in terms of minimum DCF. It performs even better than MVG, indeed the last two features of dataset did not fit the MVG assumptions.

10 Score calibration and Fusion

10.1 Best performing models

After training all models, we now focus on calibrating the obtained scores. We first select the overall three best performing models in terms of minimum DCF for our application:

- The first one is **Quadratic Prior-weighted Logistic Regression** ($\lambda = 3.16e^{-3}$)
- The second model is **RBF Kernel SVM** ($\gamma = e^{-1}$ $C = 3.16e^{1}$)
- The last and most promising one is the **Diagonal covariance GMM** model with 8 components for class 0 and 32 for class 1

The following tables show minimum and actual DCF values for these best models:

Quadratic LR	RBF Kernel SVM	Diagonal covariance GMM
0.2300	0.1755	0.1312

Table 18: Minimum DCF the for best three models

Quadratic LR	RBF Kernel SVM	Diagonal covariance GMM
0.2852	0.4216	0.1517

Table 19: Actual DCF the for best three models

The most promising model is Diagonal covariance GMM with (8,32) components. For our applications GMM scores seems also to be well calibrated. We can try to extend our analysis to different application points:

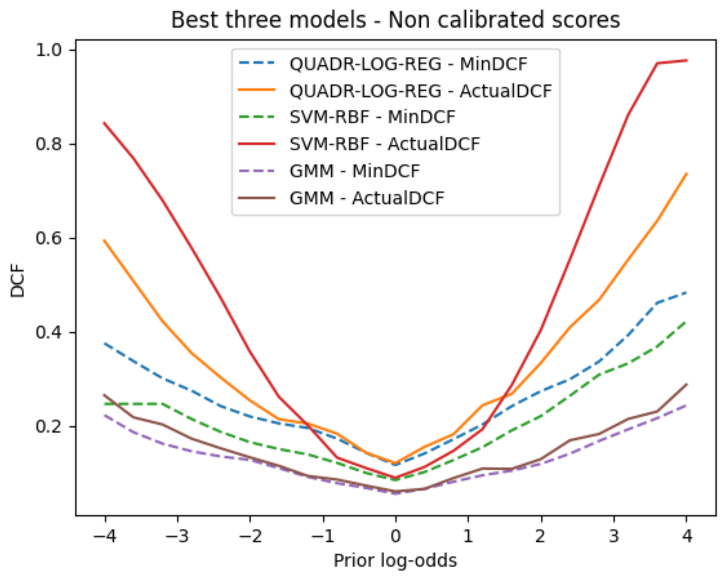


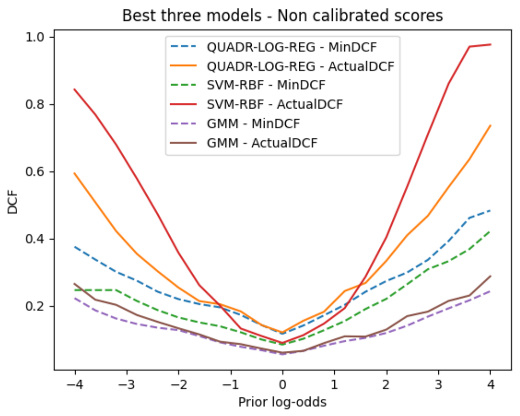
Figure 11: Minimum and actual DCF values for the three best models

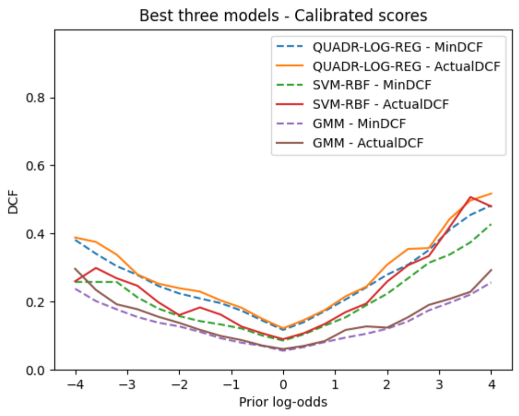
Analyzing different application points we notice that the performance ranking is the same for our selected application: the best model is GMM, then we have SVM and last QLR (in terms of minimum

DCF) for the majority of operation points. In terms of Actual DCF we get a different ranking: SVM shows poor calibration for the majority of the working point, QLR also shows poor calibration, but in working points far from 0 it becomes the second best model. GMM is overall the best performing both in terms of minimum and actual DCF, showing also well calibration.

10.2 Calibration

In this section we use a Logistic regression calibration model, trained with different priors, in order to achieve well calibrated scores for the selected models. We also use a KFOLD approach with $K = 5$. Starting from the QLR model, we train the calibration model using prior equal to 0.05 (which leads to the best result). For the SVM model we select a calibration model prior equalt to 0.05, while for GMM we select prior equal to 0.1. The following tables show how calibration improved actual DCF for our application. We also analyze the calibration for a wide range of application points.





We can notice how calibration was very effective for Quadratic Logistic Regression and RBF Kernel SVM. In both cases the calibration process improved actual DCF for the majority of working point. Focusing on GMM we can see that calibration provided no benefit to the system, but this is consistent with our expectation because GMM already has well calibrated scores.

Minimum DCF	Actual DCF no calibration	Actual DCF calibrated
0.1312	0.1517	0.1518

Table 21: Diagonal covariance GMM calibration

Minimum DCF	Actual DCF no calibration	Actual DCF calibrated
0.1755	0.4216	0.1864

Table 22: RBF Kernel SVM calibration

Minimum DCF	Actual DCF no calibration	Actual DCF calibrated
0.230	0.2852	0.2360

Table 23: Quadratic LR calibration

10.3 Fusion

Now, we create a model fusing scores from the three different classifiers: the logistic regression calibration model is trained with prior equal to 0.05. In particular, for our application point we get the following values:

Minimum DCF	Actual DCF calibrated
0.1332	0.1558

Table 24: DCF values for fusion model

We can visualize the performance of the fusion model in terms of actual and minimum DCF:

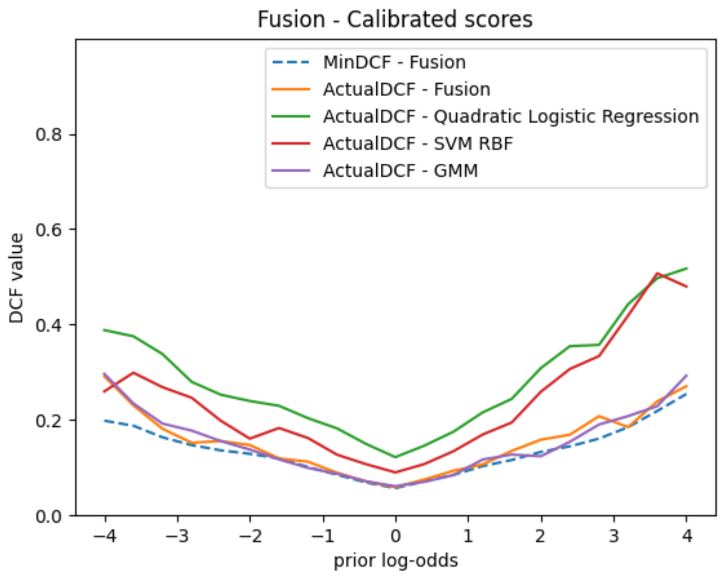


Figure 12: Minimum and actual DCF values for fusion model

We can see that fusion model achieves performance similar to diagonal covariance GMM model, it also has well calibrated scores. However for our application with prior equal to 0.1, actual DCF of fusion model is higher compared to GMM, so we might want to select (also for reducing complexity) the Diagonal covariance GMM model as our final delivery system.

11 Evaluation

In this phase we use the test dataset to measure the performance of our model after all choices we have made so far.

We start analyzing how our final system performs on the evaluation set data. The following table contains DCF values, then we can compute a Bayes error plot to see how the models works with different application points.

Minimum DCF	Actual DCF not calibrated	Actual DCF calibrated
0.1838	0.1953	0.1894

Table 25: DCF values for final model

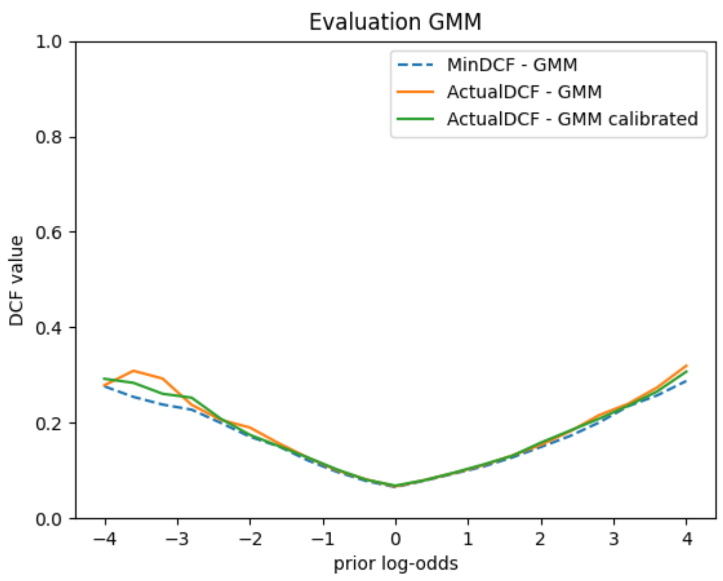


Figure 13: Bayes error plot for GMM final model

Looking at the plot we find again that calibration does not provide significant improvement to the system, but for the evaluation set and our application it provides a better actual DCF value.

Now, we consider all discarded models and compare their actual DCF values (calibrated) for a variety of application points.

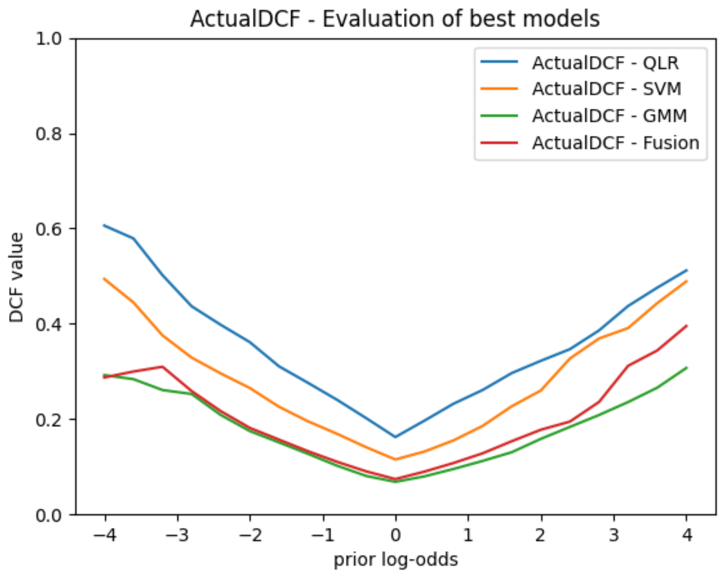
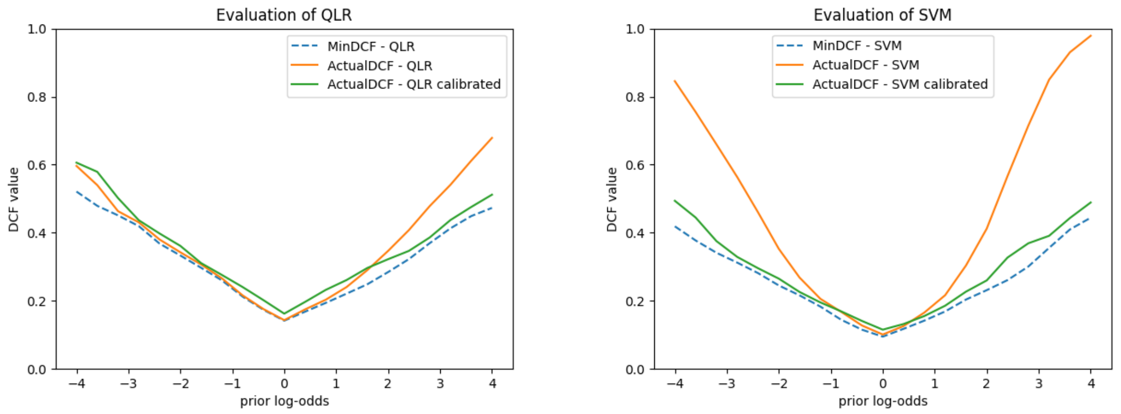


Figure 14: Actual DCF for discarded models vs final

We can see that our choices were good, since our final system performs better than all other models in almost all application points. Fusion model performs better for a very small set of application points. We can also see performances of QLR ad SVM on the evalution set:



We can see that in both cases calibration reduced actual DCF values. Finally, we retrain GMM and compute minimum DCF value on the evaluation set, to compare it with the one obtained with the chosen final system. Trying all possible combinations of components we get the Diagonal GMM model (class 0 : 4 components; class 1: 16 components) achieves a minimum DCF value of 0.1782 for our target application, and would havethus allowed us to obtain better performance for the task.