Laboratory 13

In this laboratory we will focus on calibration and fusion of scores of binary classifiers, we will analyze the K-fold cross-validation approach and we will introduce the evaluation set used for our project.

Training, validation, evaluation

Model training, calibration training and (calibration) validation

Up to now we have employed a training set to train and select the best model for a given task. For this, we have split the training set in two separate partitions: model training and validation. The former partition was used to estimate model parameters, whereas the second partition was used to evaluate different models and perform hyper-parameter tuning or model selection (cross-validation). In real scenarios this procedure is employed to select the model that we want to actually use for our task.

As we have seen, some classifiers may provide scores that are not well-calibrated. In these scenarios we need to either estimate an application-specific threshold or a calibration transformation that maps scores to well-calibrated log-likelihood ratios (LLRs).

Calibration transformations are typically estimated from a set of calibration scores. We can thus treat our former validation set as a dataset to train and evaluate calibration. A possible approach consists then in splitting the former validation set in two parts, calibration training and calibration validation sets, using the first to estimate the calibration function and the second to evaluate its goodness (e.g. to select the training object hyper-parameters, to compare different calibration approaches or simply to assess whether our calibration model seems effective). Due to the small amount of validation scores, however, in some scenarios it may be more effective to employ a K-fold approach.

Single fold: We split the dataset in two partitions. The first is used to estimate a calibration model $\mathcal{M}$, the second to evaluate its effectiveness (and thus to perform model selection in case of competing models). The calibration model that we will then use for real data is $\mathcal{M}$

K-fold: We split the set in K partitions. We iteratively compute a model $\mathcal{M}_i$ using data pooled from K-1 partitions, and compute calibrated scores for the remaining fold. The performance metric of the method is computed over the set of pooled scores obtained by all the $\mathcal{M}_i$ models (and is used to perform model selection in case of competing models). A final model $\mathcal{M}$ is then trained over the whole dataset, and employed for application data.

Training set and evaluation set

In real scenarios training data is employed to select the model that we want to actually use for our task.

In many use cases we may want to further analyze whether our approach was indeed effective for some task. For this, we can employ a further *evaluation* dataset, i.e. a dataset that was no exposed during model training, and that may thus provide unbiased results for our target application. In some cases, such an evaluation set may consist of data collected during the lifetime of our recognizer. In other cases, the evaluation set may consist of data that we withheld from the training stage. In both cases, the evaluation set can be used to assess the effectiveness of our methodology, and to compare it with alternative approaches.

NOTE: in theory, we should employ the evaluation set only to assess the goodness of our approach. As soon as we employ the dataset to make decisions on any aspect of our recognizer, we are transforming the evaluation set in an additional training set. In practice, it may be difficult to employ a new evaluation set whenever we need an unbiased result, so it may happen that the same evaluation set is used to select an approach among different alternatives (a form of model selection). While limited use of the evaluation set for model selection may not significantly bias the results, we need to pay attention to limit as much as possible such usage of evaluation datasets.

Score calibration

Score calibration allows recovering a LLR interpretation of the scores of a classifier as to allow us to perform optimal Bayes decisions, ideally for a wide range of possible applications. This is achieved through the estimation of a calibration function $f_{cal}(s)$ that maps a raw score s to a calibrated score $s_{cal} = f_{cal}(s)$.

Function f_{cal} can be estimated in several ways. In this laboratory, we will focus on a parametric affine model for f_{cal}:

$$f_{cal}(s) = \alpha s + \gamma \tag{1}$$

where α and γ are parameters that we need to estimate on a calibration training set. A possible approach for the estimation of α and γ consists in training a (prior-weighted, non regularized) logistic regression model over the original, raw scores:

$$J(\alpha, b) = \sum_{i=1}^{n} \xi_i \log\left(1 + e^{-z_i(\alpha s + \beta)}\right) \ , \quad \xi_i = \begin{cases} \frac{\pi_T}{n_T} & \text{if } z_i = +1 \quad (c_i = 1) \\ \frac{1-\pi_T}{n_F} & \text{if } z_i = -1 \quad (c_i = 0) \end{cases}$$

The model score $\alpha s + \beta$ can be interpreted as a log-ratio of class posterior probabilities for the prior π_T, thus a LLR-like score can be obtained by removing the corresponding prior log-odds:

$$f_{cal}(s) = \alpha s + \beta - \log \frac{\pi_T}{1 - \pi_T}$$

i.e.,

$$f_{cal}(s) = \alpha s + \gamma \ , \quad \gamma = \beta - \log \frac{\pi_T}{1 - \pi_T}$$

Although the logistic regression model is trained for a specific application typically it will prove effective also for different applications, as we will see in our example.

We can generalize the approach to the fusion of the scores of multiple systems. In this case, we want to estimate an affine function

$$f_{fusion}(s_1, s_2, \dots s_m) = \alpha_1 s_1 + \alpha_2 s_2 + \dots + \alpha_m s_m + \gamma = \boldsymbol{\alpha}^T \boldsymbol{s} + \gamma \ , \quad \boldsymbol{s} = \begin{bmatrix} s_1 \\ \vdots \\ s_m \end{bmatrix} \ , \quad \boldsymbol{\alpha} = \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_m \end{bmatrix}$$

where $s_1 \dots s_m$ are the scores, for a *single sample*, of m different classifiers. Also in this case, we can train a prior-weighted logistic regression model

$$J(\boldsymbol{\alpha}, b) = \sum_{i=1}^{n} \xi_i \log\left(1 + e^{-z_i(\boldsymbol{\alpha}^T s + \beta)}\right) \ , \quad \xi_i = \begin{cases} \frac{\pi_T}{n_T} & \text{if } z_i = +1 \quad (c_i = 1) \\ \frac{1-\pi_T}{n_F} & \text{if } z_i = -1 \quad (c_i = 0) \end{cases}$$

and compute the fusion mapping as

$$f_{fusion}(s_1, s_2, \dots s_m) = \boldsymbol{\alpha}^T \boldsymbol{s} + \beta - \log \frac{\pi_T}{1 - \pi_T} = \boldsymbol{\alpha}^T \boldsymbol{s} + \gamma \ , \quad \gamma = \beta - \log \frac{\pi_T}{1 - \pi_T}$$

In `Data/scores_1.npy` and `Data/scores_2.npy` you can find the scores of two classifiers for a given set of samples, with corresponding labels provided in `Data/labels.npy`. These scores correspond to the validation set of a larger training set.

Table 1: Raw scores — minimum and actual DCF — $\pi_T = 0.2$ — full dataset

	System 1		System 2	
	minDCF	actDCF	minDCF	actDCF
Raw scores	0.204	0.298	0.311	0.328

Analyze the scores of the two systems. For both systems, visualize the Bayes error plots for minimum and actual DCF (Figure 1). As target application we select $\pi_T = 0.2$. Compute minimum and actual

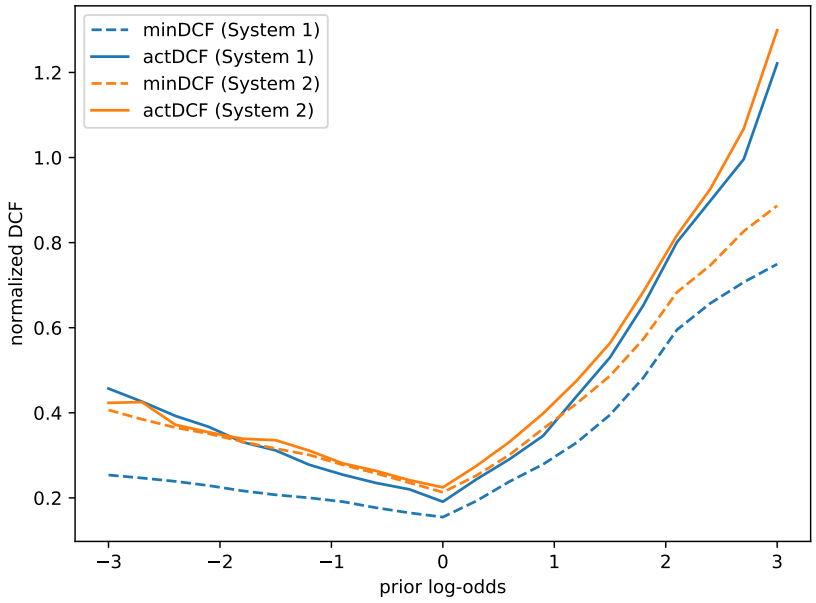


Figure 1: Single fold — actual and minimum DCF of original raw scores — full dataset

DCF for both systems (Table 1). We can observe reasonably good calibration for system 2 for low target prior applications, but bad calibration for high target prior application and overall bad calibration for system 1. We also observe that system 1 has better discrimination potential than system 2, but the calibration loss of system 1 brings its performance close to that of system 2.

Calibration — Single-fold approach

To estimate the calibration parameters we require a labeled training set (calibration training set). To assess the quality of the transformation, we can then compute actual and minimum DCFs of calibrated scores. To avoid bias in our results, the calibration training set and the calibration validation set should be different. We start analyzing a simple set-up based on our usual split of training-validation samples. In this case, the split is applied to our scores dataset (which may, for example, be one of the validation sets we employed in previous laboratories).

Split the set in two partitions. These partitions should contain similar distributions of the scores. If the scores set is sufficiently large, we can simply shuffle the scores of each class and then take a set of the scores of each class for training and the remaining part for validation. For this laboratory, we simply take every third score for calibration training, and the remaining two thirds as calibration validation. The split is obtained with

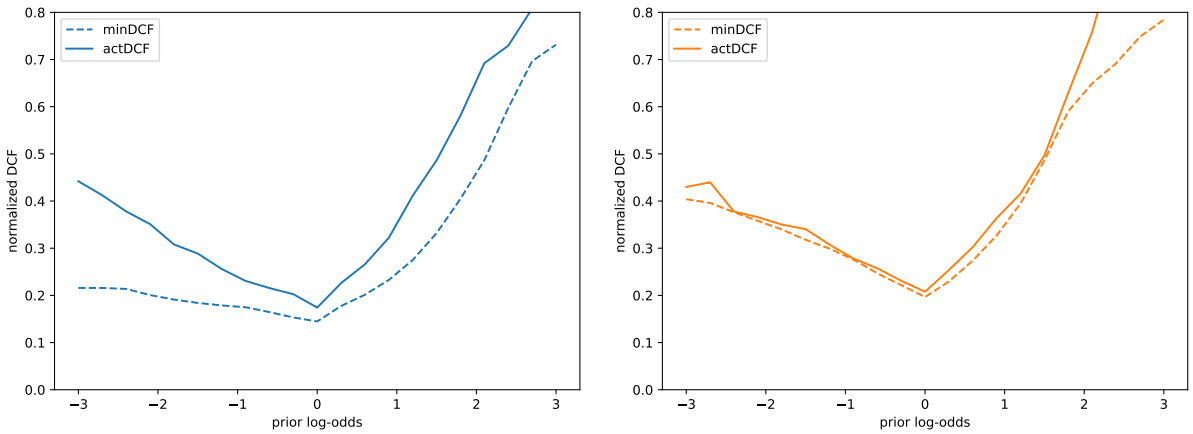
```
scores_sys_1 = numpy.load('Data/scores_1.npy')
scores_sys_2 = numpy.load('Data/scores_2.npy')
labels = numpy.load('Data/labels.npy')

SCAL1, SVAL1 = scores_sys_1[::3], numpy.hstack([scores_sys_1[1::3], scores_sys_1[2::3]])
SCAL2, SVAL2 = scores_sys_2[::3], numpy.hstack([scores_sys_2[1::3], scores_sys_2[2::3]])
```

`SCAL1, SVAL1` are the calibration training and calibration validation set for system 1, and similarly for `SCAL1, SVAL1` and system 2.

Since the calibration validation set is a subset of the whole dataset of scores, we start re-evaluating the performance of the systems on this new validation set

NOTE: we cannot directly compare the results with those of the previous section, since we changed the validation set



(a) System 1

(b) System 2

Figure 2: Single fold — actual and minimum DCF of original raw scores — calibration validation set

Table 2: Single fold — minimum and actual DCF — $\pi_T = 0.2$ — calibration validation and evaluation sets

	System 1		System 2	
	minDCF	actDCF	minDCF	actDCF
Calibration validation dataset				
Raw scores	0.182	0.274	0.311	0.329
Calibrated scores	†	0.206	†	0.336
Evaluation dataset				
Raw scores	0.205	0.290	0.280	0.284
Calibrated scores	†	0.225	†	0.312

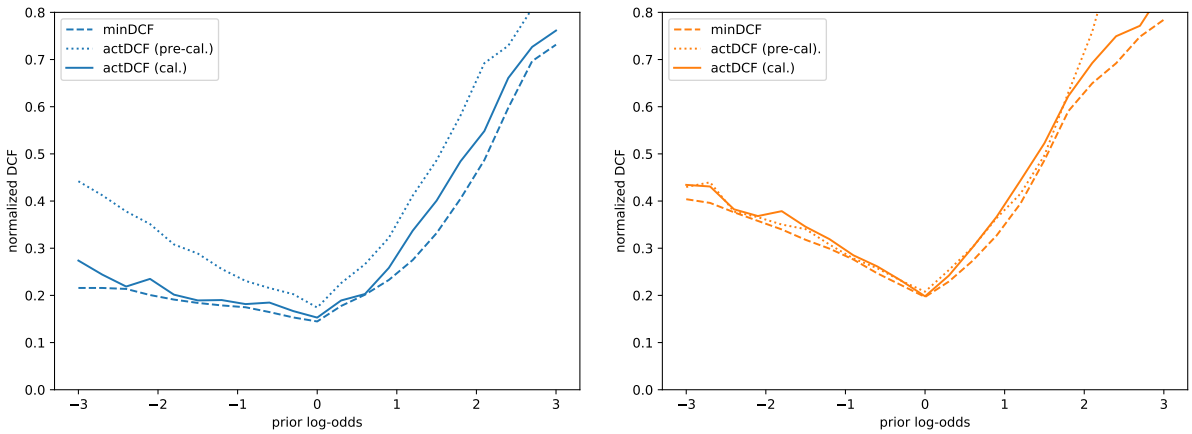
† Same as raw scores — affine calibration cannot change minDCF

In Figure 2 and Table 2 you can find the Bayes error plots and DCF values for the non calibrated (raw) scores. We observe that the results are similar, but not identical, to those computed over the whole set.

Train a calibration model for both systems (independently), using the target application $\pi_T = 0.2$ as prior for the logistic regression, *using the calibration train* split. Compute the calibrated scores for the *calibration validation* set. Evaluate the calibration effectiveness by computing actual DCF and Bayes error plots for the calibrated scores. you can also find the results in Figure 3 and in Table 2. We observe that, for system 1, calibration is effective for the whole set of applications shown in the Bayes error plot, including the target one, even though relevant miscalibration is still present in some regions of the plot. For system 2 calibration was not necessary, and indeed the results are similar to the original ones.

NOTE: If re-using the logistic regression code, pay attention to correctly reshape the logistic regression inputs and the calibrated scores — in the labs we assumed scores are 1-D arrays, whereas logistic regression works with 2-D arrays. In our case, we would need to reshape the scores to a $1 \times n$ 2-D `ndarray` before training the model

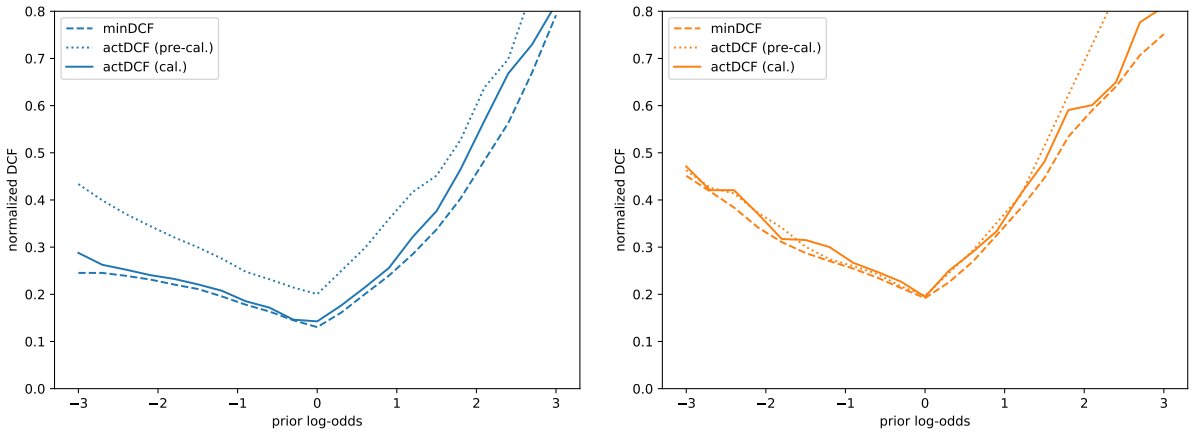
In `Data/eval_scores_1.npy` and `Data/eval_scores_2.npy` you can find the scores of the two systems for an additional, held-out evaluation set. The corresponding labels are in `Data/eval_labels.npy`. Evaluate the calibration model on the evaluation set. For this, you must simply apply the trained model to the evaluation scores, and compute the metrics over the calibrated evaluation scores (you must *not* train anything using the evaluation scores). The results are shown in Figure 4 and Table 2. We can observe that our calibration was not effective for system 2, and actually cost us almost 10% more than using the raw scores directly. For system 1, on the other hand, our approach was effective.



(a) System 1

(b) System 2

Figure 3: Single fold — minimum DCF, actual DCF of calibrated scores (cal.), and actual DCF of original (raw) scores (pre-cal.) — calibration validation set



(a) System 1

(b) System 2

Figure 4: Single fold — minimum DCF, actual DCF of calibrated scores (cal.), and actual DCF of original (raw) scores (pre-cal.) — evaluation set

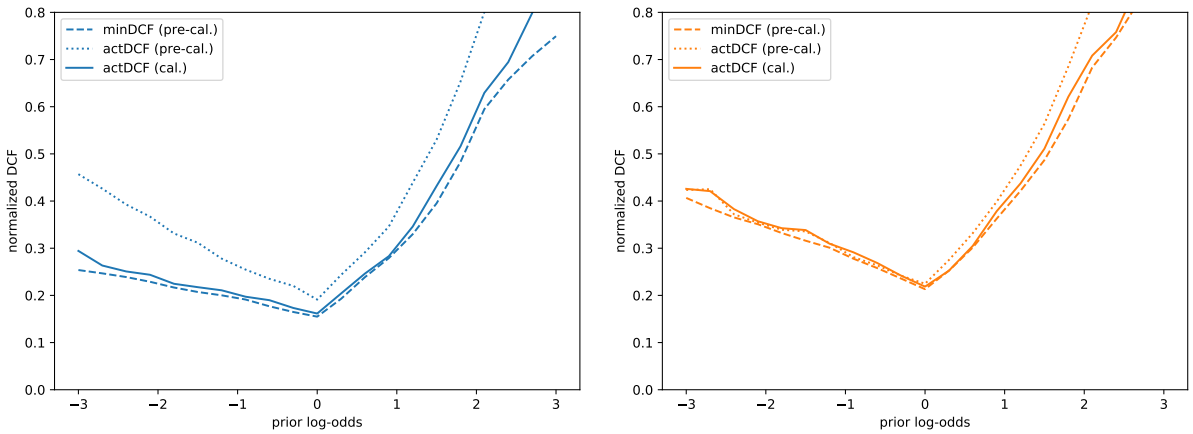
Calibration — K-fold approach

We now turn our attention to a K-fold scheme for calibration training. In this case, our reference scores are the original, non-calibrated scores of the whole scores dataset.

Divide the dataset in K folds. Here we will use $K = 5$. As for single-fold splits, we should first shuffle the scores, to ensure that we have similar distributions of scores over the different folds. For the provided dataset we can skip this step, and simply compute each fold as `S[idx::KFOLD], L[idx::KFOLD]`, where `S` and `L` are the 1-D array of scores and of labels, `idx` is the fold index and `KFOLD = 5`.

Iteratively train models $\mathcal{M}_1 \dots \mathcal{M}_5$, each leaving out one of the five folds during training, and apply them on the held-out fold. This will provide 5 arrays of scores `SCAL1` ... `SCAL5`, with corresponding labels given by `L[0::5]` ... `L[4::5]`.

Stack together the arrays of scores to build a single, 1-D pooled array `SCAL = numpy.hstack([S1 ... S5])` (pseudo-code), and build the corresponding label array `LCAL = numpy.hstack([L[0::5] ... L[4::5]])` (pseudo-code). Compute the actual and minimum DCF, and visualize the Bayes error plots for the pooled scores. Results are given in Figure 5 and Table 3.
When using a K-fold approach, we use the validation results to make all decisions on the model (model



(a) System 1

(b) System 2

Figure 5: K-fold — minimum DCF, actual DCF of calibrated scores (cal.), and actual DCF of original (raw) scores (pre-cal.) — calibration validation set

Table 3: K-fold — minimum and actual DCF — $\pi_T = 0.2$ — calibration validation and evaluation sets

	System 1		System 2	
	minDCF	actDCF	minDCF	actDCF
Calibration validation dataset				
Raw scores	0.204	0.298	0.311	0.328
Calibrated scores	†0.204	0.219	†0.315	0.326
Evaluation dataset				
Raw scores	0.205	0.290	0.280	0.284
Calibrated scores	‡	0.225	‡	0.283

† May differ from raw scores — each fold is affine-calibrated on its own, the overall transformation for the whole dataset is *not* affine anymore.
‡ Same as raw scores — the calibration model for *application* data is again an affine transformation.

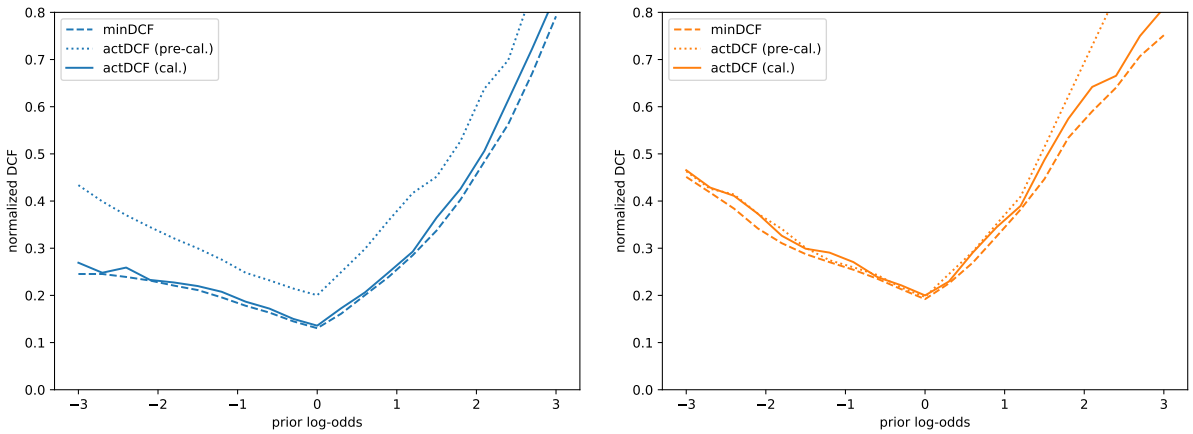
selection, hyper-parameters, ...). However, once we have selected the methodology, we still have K models, and we cannot directly choose any of them by simply comparing their performance — although we could compute the metrics for each fold, since different folds contain different data we would not know if the performance differences are due to the model or the data itself. A possible solution to select a robust model for our task consists in training a final additional model $\mathcal{M}$ using the whole dataset, and the methodology / hyperparameters we selected earlier.

Train the final model $\mathcal{M}$ using the whole calibration set. We can then evaluate whether this model would have been effective using the *evaluation* set. Apply the model $\mathcal{M}$ to evaluation data, and compute the performance metrics on the latter dataset. The results are shown in Figure 6 and in Table 3. We can observe that, compared to the single-fold approach, with k-fold we were able to estimate a more robust calibration model. Both systems are well calibrated, and we did not introduce calibration loss for system 2. These results may be explained by the fact that the calibration model has been trained with a larger dataset than in the single-fold approach.

Score-level fusion

Compute the score-level fusion for both the single-fold and k-fold approaches. The fusion can be obtained by extending the calibration approaches to work with 2-D matrices of scores

```
numpy.vstack([scores_sys_1, scores_sys_2]
numpy.vstack([eval_scores_sys_1, eval_scores_sys_2])
```

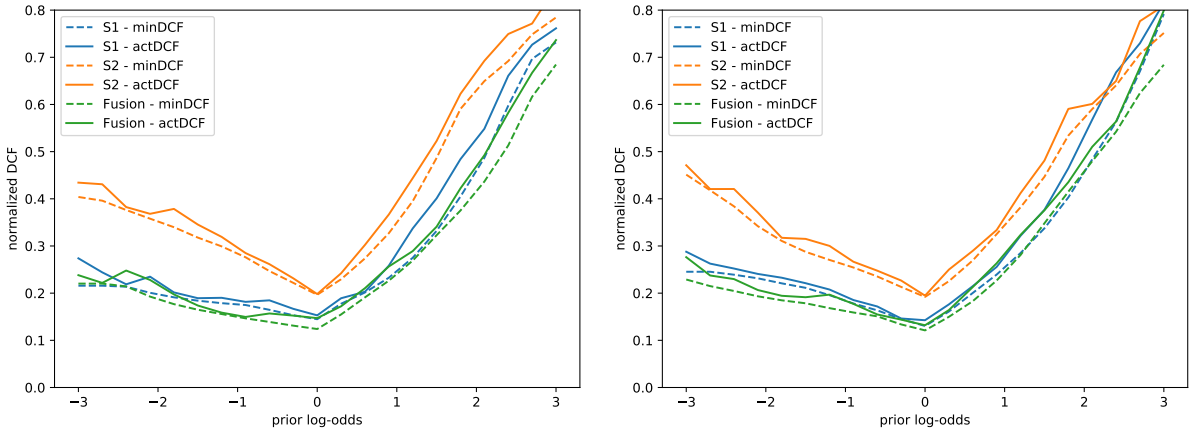


(a) System 1 (b) System 2

Figure 6: K-fold — minimum DCF, actual DCF of calibrated scores (cal.), and actual DCF of original (raw) scores (pre-cal.) — evaluation set

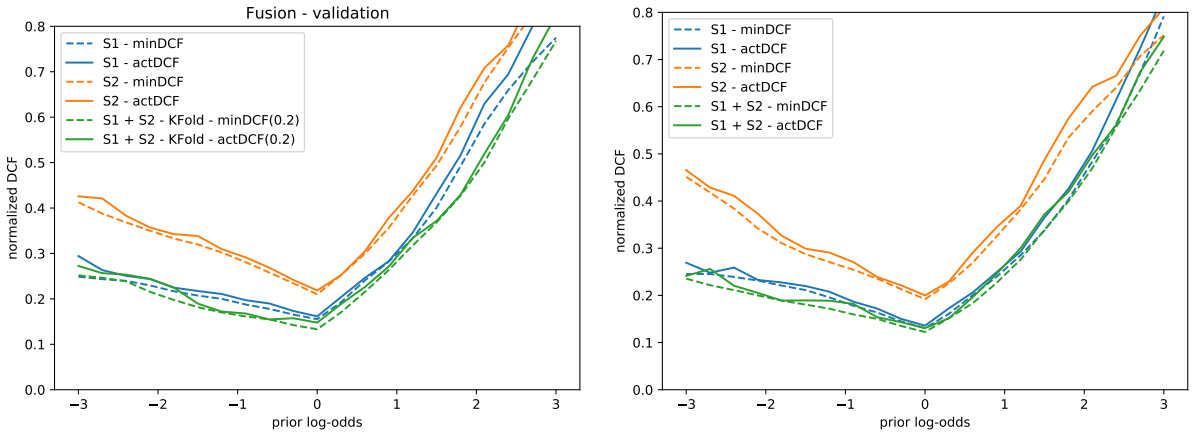
where `scores_sys_1` and `scores_sys_2` are the full sets of training scores and `eval_scores_sys_1` and `eval_scores_sys_2` are the evaluation set scores.

The results are given in Figure 7 for the single-fold approach (calibration validation set on the left and evaluation set on the right), and in Figure 8 for the k-fold approach. For the considered application, fusion results are also given in Table 4 for both approaches. Note that direct comparison of the single and k-fold metrics on the *validation* set is not particular meaningful, since the two approaches are evaluated on different validation sets. We observe that, once more, the K-fold approach proves the most effective (lowest actual DCF — remember that minimum DCF represents the optimal DCF if we were to chose the optimal threshold for the evaluation set, but our system is indeed achieving the actual DCF cost). Fusion allows us to further reduce the cost with respect to a single system set-up.



(a) Calibration validation set (b) Evaluation set

Figure 7: Single fold — score-level fusion



(a) Calibration validation set

(b) Evaluation set

Figure 8: K-fold — score-level fusion

Table 4: K-fold — minimum and actual DCF — $\pi_T = 0.2$ — calibration validation and evaluation sets (note: direct comparison of the single and k-fold metrics on the *validation* set is not particular meaningful, since the two approaches are evaluated on different validation sets)

	Single-fold		K-fold	
	minDCF	actDCF	minDCF	actDCF
Calibration validation dataset				
System 1 (cal.)	0.182	0.206	0.204	0.219
System 2 (cal.)	0.311	0.336	0.315	0.326
Fusion	0.161	0.161	0.177	0.185
Evaluation dataset				
System 1 (cal.)	0.205	0.225	0.205	0.225
System 2 (cal.)	0.280	0.312	0.280	0.283
Fusion	0.175	0.196	0.178	0.189

Project

Calibration and fusion

Consider the different classifiers that you trained in previous laboratories. Depending on the course, these will be either

01URTOV: GMM, logistic regression and SVM

01HERUU: neural networks and logistic regression

Optionally, you can also consider MVG models, but, as you should have seen, these models perform significantly worse on this dataset, and therefore can be discarded for this part of the project. For each of the main methods compute a calibration transformation for the scores of the best-performing classifier you selected earlier. The calibration model should be trained using the validation set that you employed in previous laboratories (i.e., the validation split that you used to measure the systems performance). Apply a K-fold approach to compute and evaluate the calibration transformation. You can test different priors for *training* the logistic regression model, and evaluate the performance of the calibration transformation in terms of actual DCF for the *target* application (i.e., the training prior may be different than the target application prior, but evaluation should be done for the target application). For each model, select the best performing calibration transformation (i.e. the one providing lowest actual DCF in the K-fold cross validation procedure for the target application). Compute also the minimum DCF, and

compare it to the actual DCF, of the *calibrated* scores for the different systems. What do you observe? Has calibration improved for the target application? What about different applications (Bayes error plots)?

Compute a score-level fusion of the best-performing models. Again, you can try different priors for *training* logistic regression, but you should select the best model in terms of actual DCF computed for the *target* application. Compute also the minimum DCF of the resulting model. How is the fusion performing? Is it improving actual DCF with respect to single systems? Are the fused scores well calibrated?

Choose the final model that will be used as "delivered" system, i.e. the final system that will be used for application data. Justify your choice.

Evaluation

We now evaluate the final delivered system, and perform further analysis to understand whether our design choices were indeed good for our application. The file `Project/evalData.txt` contains an *evaluation* dataset (with the same format as the training dataset). Evaluate your chosen model on this dataset (note: the evaluation dataset must not be used to estimate anything, we are evaluating the models that we *already* trained).

1. Compute minimum and actual DCF, and Bayes error plots for the delivered system. What do you observe? Are scores well calibrated for the target application? And for other possible applications?

2. Consider the three best performing systems you selected earlier, and their fusion. Evaluate the corresponding actual DCF, and compare the actual DCF error plots. What do you observe? Was your final model choice effective? Would another model / fusion of models have been more effective?

3. Consider again the three best systems. Evaluate minimum and actual DCF for the target application, and analyze the corresponding Bayes error plots. What do you observe? Was the calibration strategy effective for the different approaches?

4. Now consider one of the two (01HERUU) or three (01URTOV) approaches (we should repeat this part of the analysis for all systems, but for the project you can consider only a single method). Analyze whether your training strategy was effective. For this, consider all models that you trained for the selected approach (e.g., the different hyper-parameters / kernel combinations that you considered if you chose SVM, the different regularization coefficients for logistic regression, ...). Evaluate the minimum DCF of the considered systems on the evaluation set, and compare it to the minimum DCF of the selected model (it would be better to analyze actual DCF, but this would require to re-calibrated all models, for brevity we skip this step). What do you observe? Was your chosen model optimal or close to optimal for the evaluation data? Were there different choices that would have led to better performance?