Laboratory 8

Confusion matrices and accuracy

Confusion matrices are a tool to visualize the number of samples predicted as class i and belonging to class j. A confusion matrix is a $K \times K$ matrix whose elements $M_{i,j}$ represent the number of samples belonging to class j that are predicted as class i.

We start considering the Gaussian classifiers (MVG) on the IRIS dataset. We make use of the dataset split that we employed in Laboratory 5 (the 2-to-1 split, not the K-fold version, see 5_Lab.pdf).

For the moment we assume that mis-classification costs are all equal to 1. We have seen that, in this case, optimal prediction correspond to predicting the class which has maximum posterior probability. Compute the confusion matrix for the predictions of the MVG classifier on the IRIS dataset. You should get

		Class		
		0	1	2
Prediction	0	19	0	0
	1	0	15	0
	2	0	2	14

If you repeat the same process for the Tied covariance classifier you would get

		Class		
		0	1	2
Prediction	0	19	0	0
	1	0	16	0
	2	0	1	14

Given the limited number of errors, a detailed analysis of the IRIS dataset is not very interesting. We thus turn our attention to the problem of Laboratory 6. You can use your version of the classifier for tercets or use the one provided as solution to Laboratory 6. The class-conditional likelihoods and the corresponding tercet labels are provided in `Data/commedia_ll.npy` and `Data/commedia_labels.npy`. Compute the confusion matrix for decisions based on uniform prior and cost assumptions (i.e. maximum posterior class probability decisions). You should get

		Class		
		0	1	2
Prediction	0	210	113	61
	1	137	191	111
	2	53	98	230

Optimal Bayes decision

We now focus on making optimal decisions when priors and costs are not uniform. Optimal Bayes decisions are decisions that minimize the expected Bayes cost, from the point of view of the recognizer $\mathcal{R}$.

Although in the lectures we denoted classes with $k = 1 \dots K$, in the following we use indices $k = 0 \dots K-1$. This has the advantage that indexing follows the same convention used in numpy matrices, and we have the same representations for binary and multiclass problems.

For a K-class problem, let

$$\boldsymbol{\pi} = \begin{bmatrix} \pi_0 \\ \vdots \\ \pi_{K-1} \end{bmatrix}$$

denote the prior class probabilities. We already discussed how to compute class posterior probabilities:

$$P(C=c|x,\mathcal{R}) = \frac{f_{X|C,\mathcal{R}}(x|c)\pi_c}{\sum_{k=0}^{K-1} f_{X|C,\mathcal{R}}(x|k)\pi_k}$$

We define the cost matrix

$$\boldsymbol{C} = \begin{bmatrix} 0 & C_{0,1} & \dots & C_{0,K-1} \\ C_{1,0} & 0 & \dots & C_{1,K-1} \\ \vdots & \vdots & \ddots & \vdots \\ C_{K-1,0} & C_{K-1,1} & \dots & 0 \end{bmatrix}$$

$C_{i,j}$ represents the cost of predicting class i when the actual class is j.

When we classify a new sample, we do not know its real class, however we can compute the expected Bayes cost of predicting class c according to the posterior class probabilities provided by the recognizer $\mathcal{R}$:

$$\mathcal{C}_{x,\mathcal{R}}(c) = \sum_{k=0}^{K-1} C_{c,k} P(C=k|x,\mathcal{R})$$

The optimal Bayes decision consists in predicting the class c^* which has minimum expected Bayes cost:

$$c^* = \arg\min_c \mathcal{C}_{x,\mathcal{R}}(c)$$

For binary tasks with classes $\mathcal{H}_T = 1$, $\mathcal{H}_F = 0$, we have two costs $C_{0,1} = C_{fn}$ and $C_{1,0} = C_{fp}$, i.e. the costs of false negatives and false positive errors. The expected Bayes cost for predicting either of the two classes are

$$\mathcal{C}_{x,\mathcal{R}}(0) = C_{0,0}P(C=0|x,\mathcal{R}) + C_{0,1}P(C=1|x,\mathcal{R}) = C_{fn}P(C=1|x,\mathcal{R})$$
$$\mathcal{C}_{x,\mathcal{R}}(1) = C_{1,0}P(C=0|x,\mathcal{R}) + C_{1,1}P(C=1|x,\mathcal{R}) = C_{fp}P(C=0|x,\mathcal{R})$$

We predict the class c^* that has **minimum** cost $c^* = \arg\min_c \mathcal{C}_{x,\mathcal{R}}(c)$. For the binary task, we can also express c^* as

$$\begin{aligned} c^* &= \begin{cases} 1 & \text{if } \log \frac{C_{fn}P(C=1|x,\mathcal{R})}{C_{fp}P(C=0|x,\mathcal{R})} > 0 \\ 0 & \text{if } \log \frac{C_{fn}P(C=1|x,\mathcal{R})}{C_{fp}P(C=0|x,\mathcal{R})} \leq 0 \end{cases} \\ &= \begin{cases} 1 & \text{if } \log \frac{\pi_1 C_{fn} f_{X|C,\mathcal{R}}(x|1)}{(1-\pi_1)C_{fp} f_{X|C,\mathcal{R}}(x|0)} > 0 \\ 0 & \text{if } \log \frac{\pi_1 C_{fn} f_{X|C,\mathcal{R}}(x|1)}{(1-\pi_1)C_{fp} f_{X|C,\mathcal{R}}(x|0)} \leq 0 \end{cases} \\ &= \begin{cases} 1 & \text{if } \log \frac{f_{X|C,\mathcal{R}}(x|1)}{f_{X|C,\mathcal{R}}(x|0)} > -\log \frac{\pi_1 C_{fn}}{(1-\pi_1)C_{fp}} \\ 0 & \text{if } \log \frac{f_{X|C,\mathcal{R}}(x|1)}{f_{X|C,\mathcal{R}}(x|0)} \leq -\log \frac{\pi_1 C_{fn}}{(1-\pi_1)C_{fp}} \end{cases} \end{aligned}$$

i.e., we can compare the log-likelihood ratio

$$r(x) = \log \frac{f_{X|C,\mathcal{R}}(x|1)}{f_{X|C,\mathcal{R}}(x|0)}$$

with threshold

$$t = -\log \frac{\pi_1 C_{fn}}{(1-\pi_1)C_{fp}}$$

Binary task: optimal decisions

Write a function that computes optimal Bayes decisions for different priors and costs starting from binary log-likelihood ratios. The log-likelihood ratios can be computed from the classifier conditional probabilities for the two classes. File `Data/commedia_llr_infpar.npy` contains the LLRs for the inferno-vs-paradiso task. The corresponding labels are in `Data/commedia_labels_infpar.npy` . We assume that label $\mathcal{H}_T$ is inferno and $\mathcal{H}_F$ is paradiso. The function should receive the triple (π_1, C_{fn}, C_{fp}), corresponding to the cost matrix

$$\boldsymbol{C} = \begin{bmatrix} 0 & C_{fn} \\ C_{fp} & 0 \end{bmatrix}$$

and to prior class probabilities $(1-\pi_1, \pi_1)$.

To verify that your predictions are correct, you can find below the confusion matrix that you should obtain with decisions made with different costs and priors.

		Class	
		0	1
Prediction	0	293	96
	1	109	304

$$\pi_1 = 0.5$$
$$C_{fn} = C_{0,1} = 1$$
$$C_{fp} = C_{1,0} = 1$$

		Class	
		0	1
Prediction	0	271	80
	1	131	320

$$\pi_1 = 0.8$$
$$C_{fn} = C_{0,1} = 1$$
$$C_{fp} = C_{1,0} = 1$$

		Class	
		0	1
Prediction	0	257	75
	1	145	325

$$\pi_1 = 0.5$$
$$C_{fn} = C_{0,1} = 10$$
$$C_{fp} = C_{1,0} = 1$$

		Class	
		0	1
Prediction	0	302	113
	1	100	287

$$\pi_1 = 0.8$$
$$C_{fn} = C_{0,1} = 1$$
$$C_{fp} = C_{1,0} = 10$$

We can observe that

- When the prior for class 1 increases, the classifier tends to predict class 1 more frequently
- When the cost of predicting class 0 when the actual class is 1, $C_{0,1}$ increases, the classifier will make more false positive errors and less false negative errors. The oppposite is true when $C_{1,0}$ is higher.

Binary task: evaluation

We now turn our attention at evaluating the predictions made by our classifier $\mathcal{R}$ for a target application with prior and costs given by (π_1, C_{fn}, C_{fp}).

At evaluation time, we use the real labels of the test set to evaluate the decisions c^*. As we have seen, we can compute the empirical Bayes risk (or detection cost function, DCF), that represents the cost that we pay due to our decisions c^* for the test data. The risk can be expressed as

$$\begin{aligned} DCF_u = \mathcal{B} &= \sum_{k=0}^{K-1} \frac{\pi_k}{N_k} \sum_{i|c_i=k} \mathcal{C}(c_i^*|k) \\ &= \pi_1 C_{fn} \cdot FNR + (1-\pi_1) C_{fp} \cdot FPR \end{aligned}$$

where c_i is the actual class for sample i and c_i^* is the predicted class for the same sample. FNR and FPR are the false positive and false negative rates. These can be computed from the confusion matrix $\boldsymbol{M}$. Remember that $\boldsymbol{M}_{i,j}$ represents the number of samples of class j predicted as belonging to class i. We can then compute

$$FNR = \frac{FN}{FN+TP} = \frac{M_{0,1}}{M_{0,1}+M_{1,1}} \ , \quad FPR = \frac{FP}{FP+TN} = \frac{M_{1,0}}{M_{0,0}+M_{1,0}}$$

Write a function that computes the Bayes risk from the confusion matrix corresponding to the optimal decisions for an application (π_1, C_{fn}, C_{fp}) (use the same values both for computing the cost and for computing the decisions). For the previous four configurations, you should get

(π_1, C_{fn}, C_{fp})	DCF$_u$ ($\mathcal{B}$)
(0.5, 1, 1)	0.256
(0.8, 1, 1)	0.225
(0.5, 10, 1)	1.118
(0.8, 1, 10)	0.724

The Bayes risk allows us comparing different systems, however it does not tell us what is the benefit of using our recognizer with respect to optimal decisions based on prior information only. We can compute a normalized detection cost, by dividing the Bayes risk by the risk of an optimal system that does not use the test data at all. We have seen that the cost of such system is

$$\mathcal{B}_{dummy} = \min(\pi_1 C_{fn}, (1 - \pi_1) C_{fp})$$

Compute the normalized DCF for the aforementioned configurations. You should get

application point

(π_1, C_{fn}, C_{fp})	DCF
(0.5, 1, 1)	0.511
(0.8, 1, 1)	1.126
(0.5, 10, 1)	2.236
(0.8, 1, 10)	0.904

We can observe that only in two cases the DCF is lower than 1, in the remaining cases our system is actually harmful.

Minimum detection costs

We have seen during the lectures that, for binary tasks, we can measure the contribution to the cost due to poor class separation and the contribution due to poor score calibration: our classifier does not produce outputs that represent log-likelihood ratios, thus the theoretical threshold is not optimal anymore (note that this happens even for generative models — for example, the model parameters may not be consistent between training and test population).

Scores can be *re-calibrated* by using a (small) set of labeled samples, that behave similarly to the evaluation population and were not used for model training (i.e. a validation set). Alternatively, we can compute the optimal threshold for a given application on the same validation set, and use such threshold for the test population (K-fold cross validation can be also exploited to extract validation sets from the training data when validation data is not available).

Even if we do not have available such data, it may still be interesting knowing how good the model would perform if we had selected the best possible threshold. To this extent, we can compute the (normalized) DCF over the test set using all possible thresholds, and select its minimum value. This represents a lower bound for the DCF that our system can achieve (minimum DCF in the following).

To compute the minimum cost, consider a set of thresholds corresponding to $(-\infty, s_1 \dots s_M, +\infty)$, where $s_1 \dots s_M$ are the test scores, sorted in increasing order. For each threshold t, compute the confusion matrix on the test set itself that would be obtained if scores were thresholded at t, and the corresponding normalized DCF using the code developed in the previous section. The minimum DCF is the minimum of the obtained values.

(π_1, C_{fn}, C_{fp})	min DCF
(0.5, 1, 1)	0.506
(0.8, 1, 1)	0.752
(0.5, 10, 1)	0.842
(0.8, 1, 10)	0.709

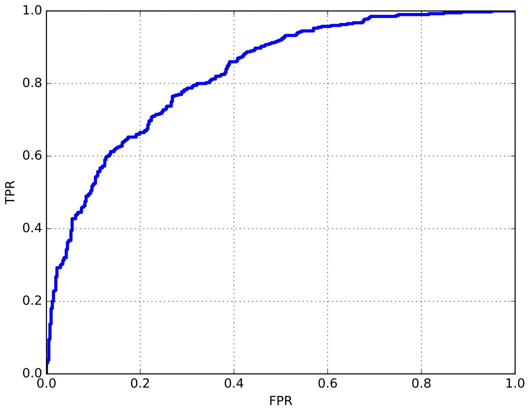
With the except of the first application, we can observe a significant loss due to poor calibration. This loss is even more significant for the two applications which had a normalized DCF larger than 1. In these two scenarios, our classifer is able to provide discriminant scores, but we were not able to employ the scores to make better decisions than those that we would make from the prior alone.

ROC curves

ROC curves are a method to evaluate the trade-off between the different kind of errors for our recognizer. ROC curves can be used, for example, to plot false positive rates versus true positive rates as the threshold varies. Note that these plots do not account for errors due to poor selection of the threshold (i.e. mis-calibration), since they simply plot how the error rates change when we modify the threshold on the evaluation set.

The most commonly used ROC curves plot true positive rates against false positive rates. You can compute true positive rates from false negative rates as TPR = 1 - FNR. The ROC curve consists of points (TPR(t), FPR(t)) where t is the threshold. By sweeping all possible thresholds we can obtain the coordinates of the ROC curve.

Plot the ROC curve for the inferno-vs-paradiso scores. For each threshold, compute the confusion matrix and extract the FNR and FPR as you did in the previous section. Compute TPR as 1 - FNR. Plot the curve that contains, on x-axis, all the FPRs, and on the y-axis all TPRs. You should obtain:



Bayes error plots

The last tool that we consider to assess the performance of our recognizer consists in plotting the normalized costs as a function of an effective prior $\tilde{\pi}$. We have seen that, for binary problems, an application (π_1, C_{fn}, C_{fp}) is indeed equivalent to an application $(\tilde{\pi}, 1, 1)$. We can thus represent different applications with different values of $\tilde{\pi}$. The normalized Bayes error plot allows assessing the performance of the recognizer as we vary the **application**, i.e. as a function of prior log-odds $\tilde{p} = \log \frac{\tilde{\pi}}{1-\tilde{\pi}}$ (note that the prior log-odds are the negative of the theoretical threshold for the considered application).

Compute the Bayes error plot for our recognizer. Consider values of $\tilde{p}$ ranging, for example, from -3 to $+3$. You can generate linearly spaced values with `effPriorLogOdds = numpy.linspace(-3, 3, 21)` (21 is the number of points we evaluate the DCF at in the example, i.e. the number of values of $\tilde{p}$ we consider). For each value $\tilde{p}$, compute the corresponding effective prior

$$\tilde{\pi} = \frac{1}{1 + e^{-\tilde{p}}}$$

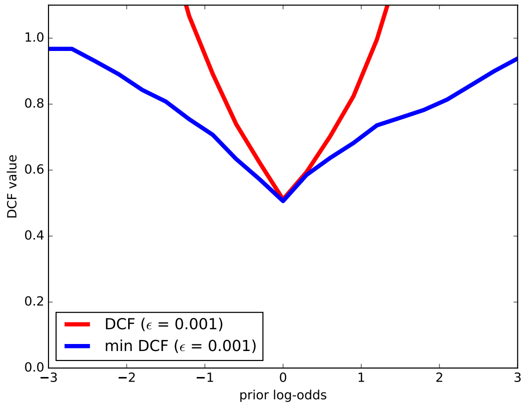
Compute the normalized DCF, and the normalized minimum DCF corresponding to $\tilde{\pi}$. Plot the computed values as a function of log-odds $\tilde{p}$: the x-axis should contain the values of $\tilde{p}$, the y-axis the corresponding DCF. To obtain the DCF, you can re-use the previous code: for each value of $\tilde{\pi}$, compute the DCF for the application $(\tilde{\pi}, 1, 1)$.

You can plot both DCF and minimum DCF over the same figure by using

```
matplotlib.pyplot.plot(effPriorLogOdds, dcf, label='DCF', color='r')
matplotlib.pyplot.plot(effPriorLogOdds, mindcf, label='min DCF', color='b')
matplotlib.pyplot.ylim([0, 1.1])
matplotlib.pyplot.xlim([-3, 3])
```

where `dcf` is the array containing the DCF values, and `mindcf` is the array containing the minimum DCF values.

The Bayes plot should look like



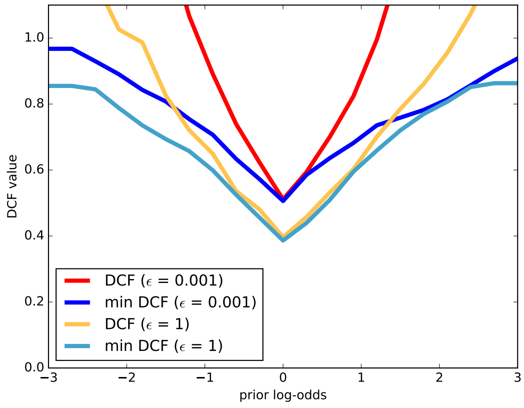
Comparing recognizers

We now employ the tools that we have seen to compare the recognizer obtained with pseudocounts $\varepsilon = 0.001$ and $\varepsilon = 1$ (see Laboratory 6).

You can find the LLRs for the second model in `Data/commedia_llr_infpar_eps1.npy`. You should get the following DCFs for the applications we considered:

	$\varepsilon = 0.001$		$\varepsilon = 1.0$	
(π_1, C_{fn}, C_{fp})	DCF	min DCF	DCF	min DCF
(0.5, 1, 1)	0.511	0.506	0.396	0.386
(0.8, 1, 1)	1.126	0.752	0.748	0.695
(0.5, 10, 1)	2.236	0.842	1.053	0.839
(0.8, 1, 10)	0.904	0.709	0.658	0.604

[optional] and the Bayes error plots:



We can conclude that the model with $\varepsilon = 1.0$ is superior over a wide range of applications (lower DCF), and also produces better calibrated scores (lower gap between DCF and minimum DCF).

Multiclass evaluation

We now consider multiclass problems. In contrast with binary problems, the target application cannot be parametrized by a single value. This makes the analysis more difficult, since we do not have a single optimal threshold anymore, but decisions take more complex expressions. It is thus also more difficult to separate the classifier ability to discriminate the classes from the loss due to poorly calibrated conditional likelihoods. For these reasons, we restrict our analysis to confusion matrices and Bayes costs.

Files `Data/commedia_ll.npy` and `Data/commedia_labels.npy` contain conditional likelihoods and labels for the evaluation population. In this case we have three classes, so the cost matrix has the form

$$\boldsymbol{C} = \begin{bmatrix} 0 & C_{0,1} & C_{0,2} \\ C_{1,0} & 0 & C_{1,2} \\ C_{2,0} & C_{2,1} & 0 \end{bmatrix}$$

and the prior probabilities can be represented by a vector

$$\boldsymbol{\pi} = \begin{bmatrix} \pi_0 \\ \pi_1 \\ \pi_2 \end{bmatrix}$$

with $\pi_1 + \pi_2 + \pi_3 = 1$.

The Bayes cost of classifying a pattern $\boldsymbol{x}_t$ as belonging to class c can be computed as

$$\mathcal{C}_{\boldsymbol{x}_t,\mathcal{R}}(c) = \sum_{k=0}^{K-1} C_{c,k} P(C = k|\boldsymbol{x}_t, \mathcal{R})$$

We can compute the vector of costs

$$\overline{\mathcal{C}}_{\boldsymbol{x}_t,\mathcal{R}} = \begin{bmatrix} \mathcal{C}_{\boldsymbol{x}_t,\mathcal{R}}(0) \\ \mathcal{C}_{\boldsymbol{x}_t,\mathcal{R}}(1) \\ \mathcal{C}_{\boldsymbol{x}_t,\mathcal{R}}(2) \end{bmatrix}$$

from the vector of posterior class probabilities

$$\boldsymbol{P}_{\boldsymbol{x}_t,\mathcal{R}} = \begin{bmatrix} P(C = 0|\boldsymbol{x}_t, \mathcal{R}) \\ P(C = 1|\boldsymbol{x}_t, \mathcal{R}) \\ P(C = 2|\boldsymbol{x}_t, \mathcal{R}) \end{bmatrix}$$

as

$$\overline{\mathcal{C}}_{\boldsymbol{x}_t,\mathcal{R}} = \boldsymbol{C}\boldsymbol{P}_{\boldsymbol{x}_t,\mathcal{R}}$$

We can then predict the class that has the minimum cost.

Compute the optimal class for cost matrix and prior vector

$$\boldsymbol{C} = \begin{bmatrix} 0 & 1 & 2 \\ 1 & 0 & 1 \\ 2 & 1 & 0 \end{bmatrix} \quad , \quad \boldsymbol{\pi} = \begin{bmatrix} 0.3 \\ 0.4 \\ 0.3 \end{bmatrix}$$

$\boldsymbol{C}$ encodes larger errors due to classifying an inferno tercet as belonging to paradiso or vice versa.

Suggestion 1: You can directly work with the matrix of class posterior probabilities for all test samples. Let $\widehat{\boldsymbol{P}}$ be such matrix, the corresponding matrix of costs for all classes and all samples can be computed as $\widehat{\boldsymbol{C}} = \boldsymbol{C}\widehat{\boldsymbol{P}}$. The optimal class can be obtained using `numpy.argmin(..., axis=0)` (note that we take the minimum)

Suggestion 2: You can compute the normalized DCF by computing the Bayes cost of a classifier that always selects the class with minimum prior cost. The cost of classifying a sample as belonging to class c according to the prior probabilities is

$$\mathcal{C}_{\mathcal{P}}(c) = \sum_{k=1}^{K} C_{c,k}\pi_k$$

The cost of a "dummy" system whose decisions are based on the prior alone is the minimum of these costs. In vector form, you can compute the cost of the "dummy" system, i.e. the normalization term required for computing normalized DCF, as `numpy.min(numpy.dot(C, vPrior))`, where `vPrior` is a column vector containing prior probabilities

Suggestion 3: Given optimal class assignments, compute the confusion matrix, and the mis-classification ratios for each class $R_{i,j} = \frac{M_{i,j}}{\sum_i M_{i,j}}$. Once you have the matrix R of mis-classification ratios, you can compute the detection cost by simply computing

$$DCF_u = \mathcal{B} = \sum_{j=1}^{k} \pi_j \sum_{i=1}^{k} R_{i,j} C_{i,j}$$

The normalized cost is obtained by dividing the DCF$_u$ value by the cost of the "dummy" system (see previous suggestion)

For $\varepsilon = 0.001$ you should get

$$\boldsymbol{M} = \begin{bmatrix} 205 & 111 & 56 \\ 145 & 199 & 121 \\ 50 & 92 & 225 \end{bmatrix} \ , \quad \text{DCF}_u = 0.560 \ , \quad \text{DCF} = \frac{\text{DCF}_u}{0.6} = 0.933$$

i.e., slightly better than using just the prior information system.

With $\varepsilon = 1.0$ you should get

$$\boldsymbol{M} = \begin{bmatrix} 216 & 77 & 31 \\ 146 & 236 & 143 \\ 38 & 89 & 228 \end{bmatrix} \ , \quad \text{DCF}_u = 0.485 \ , \quad \text{DCF} = \frac{\text{DCF}_u}{0.6} = 0.808$$

For an application with uniform class priors and costs, $\boldsymbol{\pi}_k = \frac{1}{3}$, $C_{i,j} = 1 \ \forall i \neq j$, you should obtain

	DCF$_u$	DCF
$\varepsilon = 0.001$	0.476	0.714
$\varepsilon = 1.0$	0.415	0.623